A Practical Semantics of Design Facet Interaction: A White Paper

Perry Alexander
The University of Kansas
Information and Telecommunications Technology Center
2291 Irving Hill Rd
Lawrence, KS 66044
(785) 864-7741
alex@ittc.ukans.edu

September 1, 2000

Introduction

This white paper is intended to describe a theoretical framework for defining the interactions between different views or facets of a system or component. For purposes of this exposition, a view represents a single model of a system expressed from a domain specific perspective. More concretely, a view is a model that represents one aspect of a system's behavior. When represented in this fashion, a complete system is represented as the coproduct of all its constituent models.

Take as one example models of the power constraints and functional behavior of a simple Schmidt trigger expressed in a Larch-like syntax:[1]

```
model schmidt-function(x:in real; z:out bit)
introduces
  if x =< 1.0 then x' = 0.0 else z'=z endif;
  if x >= 4.0 then x' = 5.0 else z'=z endif
end model schmidt-function;

model schmidt-power;
  p: real;
introduces
  p =< 5.0mW
end model schmidt-power
```

Both models apply to the definition of a Schmidt trigger. The first from a functional correctness perspective and the second from a power consumption perspective. Thus, it is natural to say that:

```
schmidt-trigger = schmidt-function ∧ schmidt-power;
```

[1] A Larch-like syntax for models is used here for exposition purposes. The underlying technique applies independent of specific representation.

Specifically, that the Schmidt trigger model employs properties expressed in both the function and power models. The $\wedge$ connective is used to indicate that the Schmidt trigger exhibits both functional and power consumption properties. Mathematically, the composite `schmidt-trigger` model is the *coproduct* of the original `schmidt-function` and `schmidt-power` models. Thus, the $\wedge$ operation forms the coproduct of the original specifications.[2]

It is important to note that each of the original function and power models are expressed in different semantic domains. The functional model is expressed using a state-based representation. Specifically, the value of the trigger's output in the next state (`z'`) is determined by the output in the current state (`z`) and the current input (`x`). The trigger's power consumption model is a relational expression involving the milliwatts (`mW`) physical quantity. Information from each domain, functional requirements and constraints, is expressed using its own vocabulary and underlying semantics.

The challenge of engineering systems is combining semantically disparate models in a predictive fashion. In the Schmidt trigger example, the two models are *intellectually distant*. Information in the functional model does not have surface connections to information in the power consumption model. However, experience shows that switching in the Schmidt trigger may have noticeable effects on power consumption. Information from the model describing the device's function interacts with information describing constraints. These interactions between information described in two domains are called *domain interactions*. As systems become increasingly complex, domain interactions become increasingly critical in assessing system correctness and must be modeled during the design process.

The key to modeling interactions is the ability to move information from one domain to another. We assert that there exist two basic approaches for achieving this task: (i) model everything in a single semantic domain; or (ii) model the movement of information between domains. The first approach involves the concept of a *universal semantics* or *super-semantics*. Using this approach, all models are written with, or transformed into a single semantic basis. Thus the term universal semantics. Although many such universal semantic systems have been proposed, this approach has proven unwieldy and impractical in practice. A universal semantics is necessarily a least common denominator of all semantic domains. When transforming data into the universal semantics, designers are forced to give up design abstractions that make design feasible in their original domains.

Using a circuits example, imagine performing mixed analog/digital circuit simulation in VHDL-AMS [1] by transforming both digital and analog constructs representations into SPICE models. The result would in fact simulate, but at the expense of any further understanding of phenomenon in the digital domain. Abstractions such as bits, states and clocks are eliminated from the representation to perform the simulation. The simulation results are expressed in a domain that is not familiar to the digital designer. The desired outcome is to understand the impacts of composition with the analog model *in the digital domain*.

The second approach, and the approach used in this work, is to model when and how domains interact using concepts borrowed from institution theory [2].[3] Formally, an institution is a logic of logics. It describes when theorems from one logical domain imply theorems in another. Instead of reducing all logics to a single semantic framework, the institution defines when theorems in one logic imply theorems in another. Applying the concept to general model interaction precisely addresses the domain interaction problem described previously.

[2]The coproduct is generally thought of as disjunction. We have purposefully selected the conjunction symbol here as it reflects the combining of models. It's use is a simple matter of syntax.

[3]Please note that we are not completely loyal to all aspects of institution theory.

Revisiting the circuits example, actual VHDL-AMS simulators work in just this way. Two simulators, one for analog subsystems and one for digital subsystems, operate on appropriate subsets of the design. Events are exchanged when one simulator performs an operation of interest to the other. Although the institution-based approach is far more general, the principle is the same. When something is true in one domain that impacts another, information is exchanged to reflect the interaction.

Using the institution-based approach, special models called *domain interactions* are written that specify when properties in one design domain imply properties in another. The universal semantics and its necessary transformations are avoided. Designers specify systems using their own domain specific abstractions by extending an underlying domain theory. The domain interaction theories simply define circumstances when information in one theory implies information in another. Furthermore, the results of model composition can always be expressed using vocabulary and semantics defined for the original domains. This is an important result as it allows domain engineers to understand how composing heterogeneous systems models impacts their specific domain.

Modeling Interactions

The focus of the research described here is *providing a semantics for representing how models from different domains interact under composition.* Having described the problem and provided a qualitative look at the proposed solution, the following sections outline a mechanism for formally defining interactions between systems engineering models. The semantics of interaction modeling described here will take two models, M_j and M_k, and express interactions between those models under composition, $M_j \wedge M_k$. Although other composition operators exist, $\wedge$ is the most prevalent and useful in work performed thus far.

Models, Domains and Terms

An atomic model, M_k, is a pair, (D_k, T_k), where D_k is the *domain* of M_k and T_k is the *term set* of M_k. The domain of a model is its semantic basis. The term set of a model is a set of terms that extend its domain to describe a more specific system. Thus, D_k provides meaning and inference capabilities to terms expressed in T_k. We say that a term, t, is a consequence of a model if $M_k \vdash_{D_k} t$, where $\vdash_{D_k}$ is inference as defined by domain D_k. Specifically, a term follows from a model if it can be inferred using the model's inference mechanism. The theory, Θ_k, of a model, M_k, is defined as the closure with respect to domain specific inference:

$$\Theta_k = \{t \mid M_k \vdash_{D_k} t\}$$

The complete calculus for models is beyond the scope of this white paper and is taken largely from existing model theoretic research. It is sufficient for this effort to understand that each model consists of a semantic domain model and a presentation extending that domain. Inference mechanisms are defined within the domain and closure with respect to inference defines the the model's theory. These concepts and definitions are taken directly from the formal semantics literature.

Interactions

A composite model is a set of models that are simultaneously true. Given two models M_j and M_k, we define $M = M_j \wedge M_k$ as:

$$M = \{(D_j, T_j \cup I(M_j, M_k)), (D_k, T_k \cup I(M_k, M_j))\}$$

where I is an *interaction function* defining the domain interaction. $I(M_j, M_k)$ defines a set of terms, called the *interaction term set* or simply *interaction set*, in the semantic domain of M_j. The interaction set defines the impact of M_k on M_j using terms defined in the semantic domain D_j. Under composition, the terms of M_j are unioned with the interaction term set to augment the original model with interaction results. Again, the key to the approach is that the interaction term set is expressed in the affected domain. In a design flow, composing models in this way corresponds to putting a design in its operational environment.

The projection of a composite model into a domain, π_D, is the atomic model with that domain resulting from the interaction. While composition combines models, projection pulls them back apart maintaining the effect of the interaction. Specifically:

$$\pi_D(M) = M_k \Leftrightarrow M_k \in M \wedge D = D_k$$

The projection function retrieves domain aspects of a composite model specific domain. To find the projection, the composition is formed using the interaction function and the projection with respect to the domain in question is extracted. If there is no model in the composition associated with D, then the projection is undefined. In a design flow, taking projections in this way corresponds to assessing the results of putting a design in its operational environment from one particular perspective.

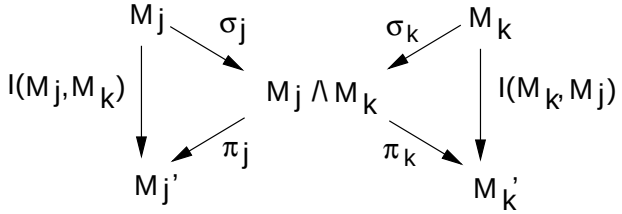

Figure 1: Category theoretic representation of models, interactions and projections.

Category theoretic relationships between various models and projection operations are shown in Figure 1. In this figure, M_j and M_k represent the original models, $M_j \wedge M_k$ represents the conjunction, and M_j' and M_k' represent projections of $M_j \wedge M_k$ back into the original domains D_j and D_k. $M_j \wedge M_k$ forms a coproduct with the original models. The σ arrows represent injection morphisms between each original model and the result of the conjunction. $M_j \wedge M_k$ also forms a product with the domain specific models formed by the interaction. The π arrows represent projection morphisms between the conjunction and domain specific representations. By definition a morphism exists between each original model and the projection result. These morphisms are defined as $\pi_j \circ \sigma_j$ and $\pi_k \circ \sigma_k$ respectively. In the proposed interaction semantics, the interaction functions define these morphisms and the following relationships hold:

$$(D_j, M_j \cup I(M_j, M_k)) = (\pi_j \circ \sigma_j) M_j$$

$$(D_k, M_k \cup I(M_k, M_j)) = (\pi_k \circ \sigma_k) M_k$$

In Figure 1 these morphisms are represented by arrows between M_j and M'_j, and M_k and M'_k respectively. Note that they are labeled with their respective interaction function although technically the formal definition is expressed in the previous equations.

A special case of model composition exists when $D_j = D_k$. In this case:

$$M = (D_j, T_j \cup T_k)$$

Specifically, the resulting model is an atomic model with the domain shared by the original models and term set equal to the union of the original term sets. Further, only π_{D_j} is defined. It is not possible to retrieve the original theories from the composition, but our initial research results have not indicated a need for such an operation.

Interaction Application Examples

The semantics of interaction has been successfully applied in the modeling of systems-on-chip domains, limited mechanical systems domains, and is being applied in networking environments. In this paper, two examples are cited demonstrating interactions between: (i) models using different timing domains; and (ii) constraint models and functional models.

Timing Model Interaction

One of the simplest and most powerful interactions defined thus far is the interaction between monotonic logic (mathematics) and a state-based semantics. Two interactions define the relationship defined in the *logic* domain and temporal claims defined in the *state based* domain (called `stateBased`). Assume that logic simply provides a mathematical domain that is monotonic, *i.e.* unchanging. In contrast the state based domain provides the concepts of current and next state. Intuitively, if both models describe the same system, each assertion made in the monotonic (logic) model must be true in every state of the temporal (state based) domain.

The first interaction equation defines the impact of an interaction between logic and state based specifications on the logic domain:

$$I(M_j : stateBased, M_k : logic) = \{t : T_j, s : S \mid t@s\}$$

In this case, the interaction set is defined as the invariant predicates asserted in every possible state in the stateBased domain. The notation $t@s$ is used to represent the term t asserted in state s. The set S is the set of all possible states. Effectively, the interaction asserts each invariant statement in every state. Assuming that $\forall x : integer \cdot P(x)$ is a term in the logic domain, then:

$$\forall s : S \cdot \forall x : integer \cdot P(x)@s$$

holds in the state based domain and the set:

$$I(M_j, M_k) = \{s : S \mid (\forall x : integer \cdot P(x))@s\}$$

defines the interaction. Specifically, that because $\forall x : integer \cdot P(x)$ is monotonic, it must hold in every state.

This interaction is extremely useful in defining system constraints such as power consumption. Many constraints are defined in a monotonic fashion. By composing a constraints model for a component and a functional model using a state based semantics, the interaction asserts that the constraint is true in all states.

The second interaction equation is the dual of the first:

$$I(M_j : logic, M_k : stateBased) = \{t : T_j \mid \forall s : S \cdot t@s \in T_k\}$$

The interaction set is defined as all those terms in the theory of M_k that are true in every state. The property expressed is that if a term is true in every state, that term is invariant over all states. It is, in effect, invariant and can be stated without reference to state. The generation of the interaction set is analogous to the previous example.

Although similar in nature to its dual, this interaction discovers invariant properties in a state-based specification. Although discovering a constraint is an interesting concept, its usefulness arises typically when modeling properties such as safety and liveness conditions.

As noted, this is the simplest of the interactions defining relationships between domains using different temporal semantics. Other currently defined domains provide pair-wise relationships between monontonic, state based, finite state, infinite state, discrete time and continuous time domains. Not all of these domains are isomorphic, thus many interactions define only partial transformations of information.

Just a few interesting interactions useful for defining constraints and requirements include:

- *Monotonic constraints interpreted as moving averages* — Rather than treating monotonic specifications as absolute limits, check moving averages over time. Useful for specifying constraints whose instantaneous values are not as important as values over time.
- *Axiomatic specifications interpreted as assertions in operational specifications* — Preconditions and postconditions specified in the state based domain become assertions checked at the initiation and termination of an operationally specified process. Useful for mapping "black box" requirements onto detailed specifications.
- *Temporal specifications interpreted as temporal constraints in operational specifications* — Like axiomatic specifications, but checked at specific temporal instances. Useful for mapping real time constraints onto detailed specifications.

In the design flow, interactions provide information to designers whenever models are composed using the projection operators. When the model composition occurs is a matter of style, however the projection operators deliver back to the domain specific designers the implications of the interaction. Specifically, the designers working with the state based, functional model learn what impacts

constraints have on their design without requiring access to the constraint model. Conversely, the constraints engineer understands the impact of the functional design on constraints issues.

This example is taken from the semantics of a systems level specification language for the systems on chip (SoC) domain. The interaction semantics has been used to successfully model relationships between monotonic, state-based, infinite-state, discrete time, continuous time and constraints domains. In most cases, the interaction specifications are similar in complexity to the given example. This is not universally true, however it does suggest a degree of expressibility in the approach.

Mechanical Design

In an ongoing proof-of-concept demonstration, the interaction semantics is used to model an aircraft redesign problem involving a hydraulic actuator. This is a classic systems engineering problem involving interaction between a hydraulic subsystem and the aircraft power plant. Specifically, a flutter problem associated with a control surface requires increasing stiffness in the surface. Without expounding on the redesign process, several options exist for the designers to consider including: (i) replacement with an electronic actuator; (ii) using a larger actuator cylinder; and (iii) boring out the existing cylinder.

In the actual design situation, the designers chose to bore out the cylinder to obtain more stiffness. This option seemed best because of wing redesign costs associated with other options. Due to the complexities of power consumption relationships, a design constraint violation was not detected. Although increasing the cylinder diameter added stiffness in the scenario where flutter occurred, it also caused instantaneous power consumption to violate power constraints in a different operational scenario.

Using the interaction semantics, models of actuator function and power constraint were developed. Actuator function was expressed in the continuous time domain with minor extensions to address mechanical issues. Power constraints were expressed in the constraints domain. In this particular case, the constraints domain was an extension of the monotonic logic domain. A variant of the interaction described earlier relates monotonic logic specifications with continuous time specifications. In defining simulations of the model, the interaction semantics was used to determine the impact of the power constraint on the functional model; specifically, to define a power constraint meaningful in a continuous time analysis activity. The result was used to show that although the power constraint is not violated in the operational scenario where flutter occurred, it is violated in another equally valid scenario.[4]

It is important to understand that the engineers simulating this redesign problem had not seen an interaction definition and were not required to modify the interaction relationship. The predefined interaction caused the exchange of information from the constraints domain into the simulation domain. The designers simply wrote power and functional specifications, then composed them to model the actuator.[5] Thus the interaction provided information to the modelers without sophisticated knowledge outside modeling domains.

[4] A completed description of this example including MATLAB analysis of the specifications is available at `http://www.ittc.ukans.edu/Projects/SLDG/rosetta`.

[5] It is important to note that this process was not automated, but served only as a proof-of-concept demonstration.

Application to Embedded Software Problems

The interaction semantics has been developed as a part of ongoing systems level design work at The University of Kansas building on work done by the author at The University of Cincinnati. The interaction semantics is not language specific and can be added to other semantics development efforts. No restrictions currently exist on the types of domains referenced by interactions. The interaction semantics has been demonstrated in several examples, two of which were discussed here.

The embedded software application area represents a classical systems engineering problem. This engineering domain integrates a number of intellectually distant domains placing highly complex constraints on software function. We believe it is possible to model such domains and interactions using a variant of the interaction semantics described here. To achieve this task, it is necessary to:

1. Identify domains pertinent to modeling embedded software and its operational environment. This will certainly include a domain for software processes and several constraint domains representing operational environment characteristics.
2. Represent the semantics of each domain in a form useful to embedded software developers and tool vendors. Specific specification and implementation languages are largely immaterial to this process.
3. Define interactions between domains when and where appropriate.

Remaining challenges exist primarily in two areas: (i) definition of specific domains and interactions; and (ii) development of efficient utilization of the semantics. A collection of highly general domains currently exists for describing requirements. For effective application to problems in the embedded software domain, specific domain theories from the embedded software application domain must be written and integrated with existing systems-on-chip oriented domains. Additionally, interactions must be defined to model movement of information between embedded software domains and domains representing its operational environment. We anticipate that both tasks will be achieved by extending existing interaction definitions.

References

[1] Institute of Electrical and Electronics Engineers, Inc., 345 East 47th St., New York, NY 10017. *VHDL Language Reference Manual (Integrated with VHDL-AMS changes)*, April 98. Draft IEEE Standard 1076.1.

[2] J. A. Goguen and R. M. Burstall. Introducing institutions. *Lecture Notes in Computer Science*, 164:221–255, 1984.

[3] B. Pierce. *Basic Category Theory for Computer Scientists.* The MIT Press, 1991.