

# 1 Introduction

Traditionally, software development has been largely a manual endeavor. Informal natural language requirements have been manually captured, a design satisfying the requirements has been manually derived, and code implementing the design has been manually coded. Recently, there has been a move away from this traditional view of software development to a new paradigm commonly called *model-based development*. In this paradigm, the development effort is centered around a formal description of the proposed software system—the ‘model’ in model-based development. For validation and verification purposes, this *formal model* can then be subjected to various types of analysis, for example, completeness and consistency analysis [Heimdahl and Leveson, 1996b, Heitmeyer et al., 1996] model checking [Grumberg and D.E.Long, 1994, Clarke et al., 1999], theorem proving [Archer et al., 1998, Bensalem et al., 1999], and test case generation [Gargantini and Heitmeyer, 1999, Offutt et al., 1999, Rayadurgam and Heimdahl, 2001]. There are currently several commercial and research tools that attempt to provide these capabilities—for example, the commercial tools Esterel Studio (with its graphical notation Safe State Machines) and SCADE Studio from Esterel Technologies [Esterel, 2004], Statemate from i-Logix [Harel et al., 1990], Simulink and Stateflow from The Mathworks Inc. [MathWorks, 2004], and SpecTRM from Safeware Engineering [Leveson et al., 1999]; examples of research tool are SCR [Heitmeyer et al., 1995], RSML<sup>-e</sup>[Thompson et al., 1999], and Ptolemy [Lee, 2003].

Our **long-range goal** has for the last decade been to dramatically increase the quality and productivity of software development for critical control systems by centering the development around fully formal models extensively supported by tools. These tools must allow engineers to specify system requirements in an appropriate and familiar notation and to *effectively* analyze the specifications to ensure that safety critical properties are satisfied. Based on years of experience with model-based development and formal modeling in various critical systems domains (air traffic control, flight guidance, munition fuzing, cardiac phasing, etc.) we believe that the following conjectures describe the fundamental obstacles that must be overcome for model-based development to have the dramatic real-world impact we envision. These conjectures form the basis for the research outlined in this proposal.

**Conjecture 1:** No modeling language will be universally accepted, nor universally applicable. Even closely related domains, such as avionics and medical technology, have justifiably different and entrenched views of what notations and features a modeling language should have to be suitable for their domains. However, certain *classes* of languages do have wide appeal, for example synchronous languages such as, Safe State Machines [Esterel, 2004], SCADE [Esterel, 2004], and SCR [Heitmeyer et al., 1995].

**Conjecture 2:** No verification and validation tool will satisfy all of a user’s analysis needs. Analysis tools are quite specialized and new sophisticated analysis tools are constantly emerging. Somehow mating a wide and growing collection of analysis tools with a variety of modeling languages (from Conjecture 1) is inevitable.

**Conjecture 3:** To make progress in model-based development, practicing engineers must evaluate proposed solutions on practical problems; if proposed theories, methods, and tools do not solve real problems, they are of little or no value. Therefore, the methods and tools must be flexible enough to easily adapt and be improved based on what is learned from using the tools on real world problems.

The **objective of this application**, which is the next step towards our goal, is to develop the foundation for building extensible and flexible modeling language processing tools that can satisfy the following three goals derived from the above conjectures—successful modeling tools must (1) allow easy extension and modification of formal modeling notations to meet the domain specific needs of a class of users, (2) allow easy construction of high-quality translations from the modeling notations to a wide variety of analysis tools, and (3) enable controlled reuse of tool infrastructure to make tools extensions cost effective.

Currently, we are aware of no tools that support the easy customization (or complete redefinition) of a modeling language necessary to accommodate the needs and likes of stakeholders in different domains. Some modeling notations, notably UML [Rumbaugh et al., 1998], attempt to appeal to different domains by being incredibly rich and incorporate a wide variety of constructs. The richness of such languages, however, makes them unsuitable for most formal analysis. In a different approach, an intermediate model notation is used to accommodate the integration of a wide variety of analysis tools—various modeling formalisms are translated to the intermediate notation that can in turn be mapped into suitable analysis tools. With an intermediate notation valuable information from the source language, for example, model structure, may be

lost in the translation to the intermediate notation and there is a risk for exponential growth in model size in each translation step.

The **central hypothesis** of our proposal is that an *extensible language* implemented using attribute grammars with forwarding [Van Wyk et al., 2002] can serve as a *host-language* for (1) a plethora of domain specific *language-extensions* that can be combined to construct new *modeling-languages* suitable for different audiences and domains, (2) the translation of these extension constructs into their semantically-equivalent representation in the host-language and the host-language translation into various target-languages, and (3) the direct translation of a language-extension construct to an analysis tool’s language when the default translation provided through the host-language is inadequate. An extensible host-language is similar to an intermediate language but avoids its pitfalls. Briefly, an attribute grammar production specifying a domain specific extension defines its *semantically equivalent* host-language expression that can be used to define its “default” semantics and “default” translations to target-languages. If translating an extension to its host-language representation and then translating this to a target-language results in the loss of important information, the language construct can explicitly specify its own translation to the target-language, bypassing the default and lossy intermediate translation through the host-language, and thus retain any important information that would benefit the translation, yielding the high-quality translations that we seek. For example, if a host-language does not support template types, but a modeling-language defined through extensions does (to allow for the modeling of generic channels in an architectural model for example), the template type extension should specify explicitly its translation to a theorem prover like PVS [Shankar et al., 1993], which supports template types. We may use the default translation to the model checker NuSMV [NuSMV, 2004], provided by the host-language, since NuSMV does not support template types. Thus, we can reuse the translations from the host-language to the target-languages when they are appropriate and bypass them for the language constructs where a direct translation yields a higher-quality implementation in the target-language. This is not possible with a set of monolithic translators nor with translation through intermediate languages.

We plan to test our hypothesis and accomplish the objectives of this proposal by pursuing the following three **specific aims**:

**Define an Appropriate Extensible Host-Language.** Based on our experiences and current industry interest, the class of synchronous languages exemplified by commercial languages such as Safe State Machines, SCADE, and Simulink will play an important role over the next decade or more. Several successful research languages much better suited for high-level modeling, for example SCR and RSML<sup>-e</sup>, also fall into this category. Therefore, we propose to work in this general class of languages. Our first task will be to identify an appropriate host-language. Our initial hypothesis is that the synchronous programming language Lustre [Halbwachs et al., 1991a] will be suitable as a *host-language* and that *language-extensions* that can be combined to create our *modeling-languages* of interest can be successfully built on top of the Lustre host-language.

**Develop Semantic Analysis, Execution, and Debugging Capabilities.** It is crucial that analysis, execution, and debugging of a model can be performed at the level of the modeling-language, not at the lower level of the host-language. We hypothesize that analysis, execution, and debugging capabilities can be effectively defined as a collection of attributes and attribute definitions for the productions in the attribute grammar defining the host-language. These attributes also form a set of “hooks” which language extensions can use to, for example, discover and report semantic errors at the modeling-language level or to specify how an extension is to behave in a debugger.

**Validate our Tools and Techniques on Realistic Systems.** To challenge and evaluate the flexibility of our proposed tools framework and the suitability of the host-language, we propose to accommodate several commercial and research modeling-languages. We plan to accommodate the popular commercial languages SCADE, Safe State Machines, and Simulink; and the research languages SCR and RSML<sup>-e</sup> as modeling-languages through language extensions to the host-language. We will also provide translations for three very different analysis tools; the symbolic model checker NuSMV [NuSMV, 2004], the theorem prover PVS [Shankar et al., 1993], and the bounded model checker in SAL [de Moura, 2004]. In addition, over the years we have established close relationships with industry partners that have given us access to realistic (and often real) case examples on which to evaluate our work. These examples are expressed in the languages mentioned above and we will use these models in our evaluation to guide our work towards scalable solutions.

The **intellectual merit** of the proposed project lies in its new and novel approach to developing robust and cost effective tools for large and evolving sets of modeling-languages and analysis tools. This is a radically different way of thinking about modeling-language design, execution environments, and translation. The conventional wisdom up to this point has been to think of modeling languages and their tool support as monolithic structures—each modeling language would be supported by its own tool, and mappings from modeling tools to analysis tools would go through some common intermediate format. Instead, we think of the tools infrastructure as an amorphous, extensible, and flexible abstract syntax tree defined through attribute grammars with forwarding. The results will be **significant** because a tools framework based on our approach will provide a catalytic infrastructure for the development, use, and evaluation of new modeling-languages and analysis tools.

The **expected outcome** of the project will be (1) a definition of a host-language and suitable attributes that can support a large class of formal modeling-languages, (2) an open implementation of a tools infrastructure supporting extensible modeling-languages through attribute grammars and forwarding, (3) the implementation of several common commercial and research languages as extensions to our host-language, as well as their translation to common analysis tools, and (4) a collection of large case-examples suitable for continued modeling research.

In addition to our research contributions, the **broader impact** of the proposed project promises to have several positive influences on industrial practice and education. If we are successful, we anticipate that commercial tool vendors will adopt our techniques to design their tools for dramatically increased flexibility. They would then be able to cheaply provide language customizations and extensions for a wide variety of domains and fickle customers—customizations and extensions that would be prohibitively costly to accommodate in current tools architectures. In addition, with the forwarding mechanisms we suggest in this proposal, they will be able to easily and cheaply provide translators to new and promising analysis tools suiting their customers’ needs. Furthermore, the technology will allow easier collaboration of researchers in formal modeling and formal analysis. A common and stable tools infrastructure can be maintained and incrementally extended by various research groups; thus, we hope that researchers can focus their efforts on improving formal modeling and analysis rather than the time consuming construction of tools. Finally, in our teaching and advising, we expect to provide educational experiences where the students are both users of the tools and education-oriented modeling-language built by the tools, as well as, contributors to their infrastructure. Our intention is to extensively use and extend the tools when teaching software engineering, formal methods, programming languages, and compilers at both the graduate and undergraduate level.

The team from the University of Minnesota is **particularly well prepared** to undertake the proposed effort. The Minnesota team has extensive experience with practical formal methods [Leveson et al., 1994, Heimdahl and Leveson, 1996a, Miller et al., 2003] as well as building tools to be used in the evaluation of our research [Heimdahl et al., 1998, Thompson et al., 1999, Thompson et al., 2000a]. We have extensive experience in extensible language frameworks [Van Wyk et al., 2002, Van Wyk et al., 2001] and modular language extensions [Van Wyk, 2003, Van Wyk and Johnson, 2004]. The co-PI Van Wyk has a pending CAREER proposal to study extensible *programming* languages. While also based on forwarding attribute grammars, the main focus there is on the integration of data-flow analysis into an attribute grammar framework and the static analysis of language extension specifications to *automatically* detect incompatibilities—two issues not relevant here. The expertise at Minnesota will position us to successfully complete the research outlined in this proposal and radically increase the flexibility of formal modeling tools and, thus, act as a catalyst to dramatically further their penetration and utility in industrial applications.

## 2 Background and Related Work

There has been an immense amount of research done in formal modeling tools, mappings from modeling notations to analysis tools, and translation technology. The coverage in this section is by necessity cursory, but we will discuss our previous experiences (largely funded through NSF) and the core problem, and then present the research efforts that are most closely related to our proposed project.