The Stanford Rapide Project

Overview Of The Rapide Prototyping Project

The Rapide Language effort focuses on developing a new technology for building large-scale, distributed multi-language systems. This technology is based upon a new generation of computer languages, called Executable Architecture Definition Languages (EADLs), and an innovative toolset supporting the use of EADLs in evolutionary development and rigorous analysis of large-scale systems.

The present Rapide-1.0 effort leverages off of Stanford's Rapide-0.2 effort, and on our prior work on applying formal methods to Ada and VHDL.

Rapide is designed to support component-based development of large, multi-language systems by utilizing architecture definitions as the development framework. Rapide allows gradual refinement of architectures into products, and supports testing and maintenance based on automated comparison with formal standard architectures. Rapide adopts a new event-based execution model of distributed, time-sensitive systems -- the "timed poset model." Posets provide the most detailed formal basis to date for constructing early life cycle prototyping tools, and later life cycle tools for correctness and performance analysis of distributed time-sensitive systems.

Objectives

Rapide focuses on introducing new elements into industrial simulation technology as incremental extensions of current tools and methods.

Specification Capabilities For EADLs

The Rapide specification language is planned to allow specification of both constraints on concurrent patterns of events (posets) and real-time constraints. The associated support tools will provide automated formal analysis (both proof theoretic and simulation checking) of highly detailed properties of large-scale distributed system architectures containing both hardware and software components. We plan to utilize formal constraints in developing new techniques of performance analysis of complex prototype systems.

EADL-Based Formal Reference Architectures For Industrial Systems

We plan to define formal reference architectures for particular industrial systems in Rapide (e.g., X/Open DTP and other industry standards), analyze them formally for various critical properties, (e.g., consistency, liveness, durability and real-time constraints), and propose them as standard formal architecture definitions.

Consistency Analysis Tools To Compare Architectures

Rapide support tools will include comparative simulation tools to support consistency analysis between architectures. These tools will enable an actual system to be executed and tested for conformance with a formal reference architecture, thus providing a new technology for industrial standardization.

Architectural Frameworks For Distributed, Multi-Lingual Systems.

A Rapide toolset is being developed to support the use of executable architectures as frameworks for composing industrial-size systems that contain components in different languages (e.g VHDL, Ada, C++, C, Rapide) and can be executed on multiple workstations.

Formal Methods For EADLs

By defining formal mappings of other architecture formalisms (e.g., Honeywell's MetaH, VHDL, Verilog, LOTOS) into Rapide, we will develop general techniques for defining formal semantics for domain specific EADLs and a proof theory for component-based architectures in many EADLs.

Rapide Language

Rapide is a language framework consisting of (i) a type language, (ii) an executable architecture definition language, (iii) a specification language, and (iv) a concurrent reactive programming language. The architecture definition, specification and reactive programming languages also share a pattern sublanguage that provides expressions for reacting to and constraining event-based computations. While these languages satisfy certain compatibility requirements (e.g., they have the same visibility, scoping and naming rules, and underlying event-based execution model), the Rapide language framework is designed to permit alternative programming or specification languages to be used in conjunction with the type language.

The *type language* is based on a single general interface type construct together with inheritance derivations for building new interfaces from existing ones. The form of "derivation" provided resembles inheritance mechanisms in object-oriented languages. All standard kinds of types (e.g., arrays, classes, task types) are particular instances of interface types. The type language is used to define interfaces of components (both hardware and software) of a system.

The *architecture definition language* provides executable features for composing systems out of component interfaces by defining their synchronization and communication interconnections in terms of patterns of events. These features provide powerful new constructs for defining very large, possibly dynamic, architectures. Moreover, event pattern constructs are used to define mappings between architectures. These EADL mappings are implementable to provide new tools for automatically comparing system architectures at differing abstraction levels in hierarchically designed systems, and for monitoring actual systems for conformance with industry standard architectures.

The *constraint language* provides constructs for abstract specification of the behavior of a distributed system, including timing requirements. It is a constraint-based language, based on the timed poset model, that supplies constructs for defining patterns of events that are required (or forbidden) in the computation of a distributed system. Component interfaces and architectures may be given detailed abstract specifications using the specification language.

The *executable language* is a concurrent reactive programming language. It uses types, objects, and expressions of the type language, and provides module and control structures. Its principal constructs are independent (or concurrent) reactive processes that activate when patterns of events occur during execution. These pattern-triggered processes are used (i) to define architecture connections between components, and (ii) to construct behaviors of components by rule-based, reactive programming techniques. The executable language also provides standard kinds of Algol-like control structures, subprograms, exception raising and handling constructs, and timing features. Components may be assigned implementations (modules) using either the Rapide reactive programming language, or languages such as Ada, C++, or VHDL.

7/29/97/lp