Unifying Analysis Tools with Rosetta

Nicolas Frisby
System Level Design Group
Information Technology and Telecommunications Center
University of Kansas
nfrisby@ittc.ku.edu

Abstract

The Rosetta system specification language will require a variety of analysis capabilities to assist system designers. The language’s generality prohibits the development of a single analysis tool. It is proposed, instead, to leverage the existing analysis tools and create an analysis environment unified around the Rosetta language. A semi-automated tool, the Rosetta Nexus, will generate tool-specific analysis models and a correspondence with the original Rosetta specifications.

1 Introduction

System-level properties cross the traditional component boundaries of system architectures. A component cannot safely engage with another via cryptographic protocols without assuming that the other component is benevolent; both components must be trustworthy for the communication to remain secure. A compiler targeting a microprocessor component in a system may naively attempt to lower energy consumption by offloading computation to a co-processor, but the chip-to-chip communication overhead may in fact increase energy consumption at the system level. Components cannot be designed and analyzed in isolation to establish system-level properties. Furthermore, some intuitively disparate properties are actually inter-related. For instance, different implementations of an algorithm incur different energy consumptions on a microprocessor.

System-level properties and interactions are difficult to express and analyze. Rosetta is a specification language designed for expressing system-level properties. The syntax and semantics are undergoing standardization, but no Rosetta analysis tools exist. The proposed research effort seeks to develop an analysis environment capable of predicting and verifying system-level properties in Rosetta specifications.

2 Background

Interacting system-level properties are a challenge for existing specification languages and analysis tools, both in terms of expressive and analytic capabilities. The Rosetta methodology expresses a component’s various behaviors independently, in accord with the traditional separation of design perspectives between, say, functional behavior and energy consumption. The models in these two specification domains are combined to form a complete model for the component. All relevant interactions are described as part of the combination. The Rosetta syntax and semantics have been developed around this methodology, and so too will be the analysis environment.

The primary unit of specification within Rosetta is the facet, a model of a particular aspect of a component. Facets model hierarchical components via inclusion, a facet may contain sub-components modeled by other facets. The containing facet can then specify the inter-connections establishing the sub-components’ interdependence. Combining facets via inclusion is referred to as horizontal composition, in which facets represent separate components. A term algebra exists for combining multiple facets into a more comprehensive facet modeling a single component, a technique named vertical composition. The central operator of the algebra is the facet product, which asserts that any component satisfying the resulting facet must satisfy both original facets. For example, the functional behavior model and energy consumption model for the same component are specified separately and combined with the facet product.

Every facet relies on semantics central to the specification domain for that aspect of the component, such as some unit of energy consumption or a heat dissipation law. Accordingly, a distinguished kind of facet, a Rosetta domain, introduces a comprehensive vocabulary for a particular specification domain along with axioms. Rosetta domains collect the concepts with which Rosetta facets collect assertions about components. To accomplish this, every facet extends a domain and every domain extends some
simpler domain. The capability of combining facets from different domains provides Rosetta’s support for specifying heterogeneous systems.

Though Rosetta is useful as an expressive specification language with a formal semantics, this paper outlines a plan to automate domain-specific analyses of Rosetta specifications. The double entendre is not accidental; the domain-aware analysis tools under design will rely on the extension relationships among domains and facets as an organizing principle. In accord with the Rosetta methodology of separately treating and then combining traditional design perspectives, the Rosetta analysis environment will apply existing analysis tools to the corresponding perspectives of the system-level analysis.

3 Approach

The existing analysis tools support a varied spectrum of abstractions, ranging from the equations over Boolean variables of Boolean satisfiability (SAT) solvers to the higher-order logic of a theorem prover. Intermediate abstractions are supported by SAT Modulo Theories (SMT) solvers, SAT solvers enriched with theories for specific mathematical structures (like linear integer arithmetic and bitstrings), and finite-state and infinite-state model-checkers. Some tools are designed for a specific domain, like simulators for estimating the energy consumption of microprocessors. All of these tools have benefited from years of dedicated research, testing, and performance tuning.

Reproducing all of those capabilities for Rosetta would be too costly. Instead, the research proposed in this paper is to encapsulate these powerful analysis tools as part of an analysis environment for Rosetta by translating suitable Rosetta models back and forth between the appropriate tools. The Rosetta Nexus will be the linchpin of the environment.

Given a Rosetta specification, the Nexus will generate both a set of models for the specific analysis tools representing the same system and a correspondence between the generated models and the original Rosetta specification. The Nexus improves the accessibility of the analysis tools’ interfaces by encapsulating them behind a more expressive and uniform Rosetta analysis environment, and thereby also achieves analysis support for Rosetta. Properties checked by one tool can be used as premises for another, using Rosetta as the intermediate human-readable form; the Nexus becomes a semi-automated analysis assistant. There are three main challenges for the Nexus.

1. As Rosetta proceeds through standardization, syntax and semantics will change. Interfaces to the analysis tools will also change. The Nexus must be modular to minimize adaptations.

2. Many Rosetta domains will correspond to an existing domain-specific analysis tool or theory. The Nexus must be extensible, so power-users, beyond the core Rosetta support team, can develop back-ends for new analysis tools.

3. As a general specification language, Rosetta permits the use of rich functional language abstractions. No analysis tool will support them all.

**Modular monadic semantics (MMS)** [7] is the vehicle of choice for achieving modularity. MMS accommodates plug-and-play language semantics: language constructs and semantics become re-usable components. The Haskell library InterpreterLib [9] implements MMS and adds two facilities beyond the basic MMS initial algebra semantics. **Generic programming** [6] is a structure-centric form of polymorphism that drastically reduces program size by recognizing algorithmic patterns and confining the details of the algorithm to a limited number of structural cases. **Algebra combinators** (akin to traversal combinators [5]), extend the re-use capabilities of MMS with a focus on software engineering benefits. For example, one static analysis can consume the results of another without involving a data structure to retain them. A final benefit of using MMS and generic programming is that the mathematical foundations of both works will streamline future validation of the Nexus.

The environment will be deployed as a library for the Haskell programming language to achieve extensibility. Users of the library will re-use the syntactic and semantics components in order to construct back-ends for the relevant analysis tools. Doing so will require understanding the Rosetta semantics, but the software engineering effort will be minimized by the language processing aptitude and facilities of Haskell and InterpreterLib.

Mitigating the gap between the expressive power of Rosetta and the analytic power of the analysis tools will be the primary research challenge. Just as an Integrated Development Environment (IDE) combines program analysis tools such as compilers, project managers, and debuggers, the Nexus will combine specification analysis tools. This Nexus’s task is more challenging than a conventional IDE’s because no single analysis tool will be able to service a general Rosetta specification. Instead, tools will support classes of Rosetta specifications as codified by Rosetta domains. Still, the Rosetta expression language, common to all facets regardless of domain, permits sophisticated abstractions unsupplied by most analysis tools. Uses of these abstractions must be re-expressed into abstractions tools can handle, and, naturally, this must happen on a tool-by-tool basis. General patterns may emerge as abstractions are ported to various tools which may be automated within the Nexus, giving it a interactive suggestion capability. Though too domain- and tool-specific to list here, the transformations
performed by the Nexus are anticipated to be comparable to the whole-program inter-procedural control-flow analyses of conventional programming languages, answering the query “In what ways can this value be used by the rest of the specification?”

Consider a model at an intermediate level of abstraction that describes the state of a system as an integer. Before applying a finite-state model-checker, the specifier must further refine the specification by choosing a finite range for the integer or perhaps by returning to a finite, safe abstraction of the possible states. The Nexus can assist in this refinement by analyzing the possible usage sites of the original integer and suggesting a refactorization. Furthermore, the Rosetta environment will record the refinement process, yielding a list of (possibly already verified) transformations and intermediate models documenting the design process and providing a road-map for certifying the analysis. This facility suggests a similarity between the analysis assistant and proof assistants.

4 Evaluation

The execution of this research plan will deliver a semi-automated analysis assistant for Rosetta specifications. The Rosetta Nexus will have multiple back-ends and analysis results will be translated back and forth between Rosetta and the appropriate tools. To evaluate the success of the system, milestone implementations of the Nexus will be applied to a series of increasingly rich system-level analysis problems.

1. The first milestone is to demonstrably simplify an existing system-level analysis via the Rosetta Nexus. Given an existing moderately-complex model for a specific analysis tool, a Rosetta specification will drive the generation of a new model. The property-level equivalence of the original and generated models and a strong software engineering argument that the Rosetta specification is preferable to the original constitutes success.

2. The second milestone will add transparency to the first. A Rosetta user will be able to invoke an underlying analysis tool without directly interacting with it. For example, the state-exploration capabilities of a model-checker will be encapsulated as a state-exploration capability for Rosetta. This will require a bidirectional correspondence between the original Rosetta specification and the specification generated for the particular analysis tool.

3. The most ambitious milestone is to work with a combination of disparate analysis tools. Analysis results will be transferred between tools using Rosetta as a common intermediate form. One analysis tool will be a model-checker. The other is less determined, but will preferably be a more domain-specific analysis tool such as an energy-consumption simulator.

4. The most significant contribution of the thesis will be the identification of patterns common to the disparate back-ends of the Rosetta Nexus and a framework encapsulating them. Modular monadic semantics represents an established pattern that contributes extensibility to the framework. Other patterns will be identified during the development of the back-ends, preferably within the semi-automated refinement techniques. The final implementation of the Rosetta Nexus must demonstrate the software engineering benefits of such a framework.

Software-defined radios (SDRs) offer real-world analysis opportunities. Traditional radios are designed as embedded computer systems with digital signal processing capabilities. SDRs raise the specification level to algorithms, architectures, and protocols in such a way that the specifications can target multiple implementation fabrics. SDRs require high assurance of properties from multiple analysis domains such as security and energy consumption and involve components from multiple design domains such as digital logic and analog signal processing. It is precisely the sort of problem domain intended for Rosetta. Other problem domains are being sought as discussions with radio designers formulate the analyses of interest for SDRs.

The planned evaluation is entirely qualitative. Appropriate quantitative metrics, beyond the unfavorable “lines of specification,” remain unidentified.

5 Previous Work

For a thorough discussion of the Rosetta methodology, see [2]. [1] is a smaller case study in power-aware design through heterogeneous specification.

Modular monadic semantics [7] correlates many computing fundamentals to achieve a form of modularity with a strong correspondence to traditional language processing concerns. Syntax and semantics are held separate, but both are plug-and-play compatible components. Representing various computational effects with monadic types enables the qualified type system [3] to act as an automated bookkeeper of semantic properties required of the final denotational domain.

Generic programming [6] also achieves modularity, but does so in a complementary way to modular monadic semantics. The key insight is that many language processing analyses can be described as a function from the syntactic space to a semantic domain with two kinds of cases. The one general case assumes only mild structural properties of the syntactic and semantic domain and handles most
syntactic forms. The other override cases replace the general case with specializations for syntactic cases of interest. For example, the analysis identifying free variables in a program expression can treat most syntactic forms by taking the union of the sub-expressions’ free variables. Only syntactic forms that use or introduce bindings need be handled specifically. The modularity is derived from the general case of the analysis: any syntactic form that is handled by the general case can be added to or removed from the syntactic space or structurally modified without any change necessary for the specification of the analysis.

InterpreterLib implements both modular monadic semantics and an approach to generic programming derived from [4] by specializing the type-level programming to the structure of syntactic functors. This formulation lends itself to combination with modular monadic semantics because of the common focus on the functor structure of the syntactic space. Previous research [9] has found InterpreterLib to simplify code for a number of tools, including a synthesis capability from a basic functional language to VHDL and a plug-and-play semantics for multi-staged programming constructs quote and unquote.

As the research effort begins on the software-defined radio problem domain, the relevant SDR analysis literature will guide the development of the Rosetta specifications and Nexus facilities. The particular analyses requested by the SDR designers will determine which automated abstraction methodologies are appropriate for inclusion within the Nexus. It is anticipated that the Nexus will directly implement some existing methodologies, adapt others to the unique features of Rosetta, and even introduce novel methodologies specific to Rosetta and/or SDRs. The relevant background literature is diverse.

6 Conclusion

Rosetta is an expressive heterogeneous specification language designed for the specification of system-level properties. No analysis tool yet exists that can directly analyze Rosetta specifications. This research proposal suggests a single analysis tool would be in appropriate and, instead, the suite of existing general and domain-specific analysis tools should be adopted as needed on the basis of the specification domain. The Rosetta Nexus will be a semi-automated tool capable of recommending specification refactorizations necessary to generate a tractably analyzable model for specific analysis tools such as SAT or SMT solvers, model-checkers, or theorem provers and other domain-specific tools. Its task necessitates a modular and extensible implementation, which will be achieved through Haskell and the modular monadic semantics-based language processing library InterpreterLib. While the primary objective is the delivery of a useful analysis environment, a secondary research product will be case studies of the modularity and extensibility implementation techniques. Reports on the long-term experience of implementing the analysis environment and applying it to problems such as the design and analysis of software-defined radios will serve as evaluation. The research efforts seek to enable an unprecedented fluidity of analysis results between independent analysis tools.

References


