A Dual Spring System Case-Study Model in Rosetta

Peter J. Ashenden
Dept. Computer Science
Adelaide University
Adelaide, SA 5005,
Australia
petera@cs.adelaide.edu.au

Perry Alexander
The University of Kansas
EECS Dept. / ITTC
2291 Irving Hill Rd.
Lawrence, KS 66044, USA
alex@ittc.ukans.edu

David L. Barton
Averstar
1593 Spring Hill Road,
Suite 700
Vienna, VA 22182-2249,
USA
dlb@averstar.com

Abstract

This paper describes a design case-study undertaken as part of the language design validation for the Rosetta System Level Description Language. The system under consideration is a dual- spring mechanical system. A physical model is constructed, describing the static characteristics of springs and the requirements and constraints applying to the system. It is demonstrated that the declarative modelling style used in Rosetta is a powerful modelling tool. Furthermore, the formal semantic basis of the language allows analysis of models and interfaces with other tools.

2. Introduction

Rosetta [1,2,3] is a system-level design language, being developed by two of the authors, Alexander and Barton, as part of the SLDL Initiative [4] currently sponsored by Accellera (formerly VHDL International) and ECSI. The SLDL Initiative was originally sponsored by the EDA Industry Council, and moved under the auspices of VI and ECSI in 1999. Rosetta addresses a need for a language in which designers can specify the requirements and constraints on a system that spans multiple design domains. Requirements describe functional behaviour that a system must exhibit, and constraints describe operational bounds within which the system must remain. The different design domains include digital and analogue electronic subsystems, and the mechanical, optical, fluidic and thermal subsystems with which they interact.

Various computational models are most appropriate to describe aspects of systems in different domains. Example computational models include finite and infinite state-based, discrete event, discrete time, and continuous time. The Rosetta language is extensible to allow designs to be expressed using each of these computational models, and to allow expression of the interactions between descriptions in each computational model. The language achieves this extensibility be being based on a formal semantic underpinning and by including facilities for reflection. In combination, these features allow definition of syntax and semantics of new Rosetta domains for expressing requirements and constraints using different computational models.

This paper describes a model of a dual-spring mechanical system, developed as a design case study in Rosetta. The design was originally specified informally using a combination of English language and mathematical equations. The information in this specification formed the basis of a Rosetta specification, in which the requirements and constraints are expressed formally in a manner that is amenable to analysis and computation. The model is expressed in the *logic* domain in terms of continuous mathematical equations that describe the mechanical system.

2. Overview of the Rosetta language

The key language feature in Rosetta for describing a system is a *facet*. A facet captures one aspect of a system, describing properties of interest in one chosen domain. A given system may be described by a number of different facets, each characterising the system in a different domain. The various facets may be combined to provide a multi-faceted description, which might be instantiated as part of a larger system. In this section, we present a very brief overview of facets to enable the reader to understand the spring model that follows.

A facet, in its textual rendition, is a named unit comprising a parameter list, a collection of declarations, a domain identifier, and a collection of labelled terms. An illustrative example of a requirements facet for a sorter device is:

```
facet trigger(x::in real; y::out
bit) is
   s:: bit;
begin continuous-time
   t1:     if s = 1 then
           if x >= 0.4 then s' = 1
else s' = 0 endif;
        else
           if x =< 0.7 then s' = 0
else s' = 1 endif;
        endif;
   t2: y@t+10ns = s;
end trigger;
```

The facet is named trigger, and represents the behaviour of a Schmidt trigger device. The two parameters, x and y, each are of type bit. The parameter modes, in and out, are simply assertions whose meaning is defined in the domain of the facet. The domain is identified after the keyword begin, continuous-time in this example, representing the predefined continuous-time domain. The declaration before the keyword begin names the variable s and specifies its type as bit. The two terms t1 and t2 are assertions about the values of the parameters and the variables that must be true at all times. The variable t is implicitly defined in the continuous time domain to refer to the "current" time, and each term in the facet is implicitly universally quantified over all times. It is important to note that the operator "=" is an equality operator, not assignment. Thus the terms should be read as statements about what must be true at all times, rather than as operational definitions of events.

The notation s' denotes the value of s at the next instant after the current time. As shown in the example model, this allows descriptions of discontinuities in the value of a variable over time. The notation y@t+10ns denotes the value of the variable y at a time 10ns later than the current time t. This allows expression of temporal relationships between values of variables.

The example model shown above uses simple bit-valued and real-valued types for variables. Rosetta has a very rich type system, including scalar discrete and continuous types, and type constructors for arrays, records, tuples and sequences. Furthermore, it permits definition of higher-order types, namely, types whose values are functions. The type system has well-founded mathematical semantics. This, in conjunction with the formal semantic definition of facets and their constituents, makes formal analysis and verification of Rosetta models feasible.

Facets, such as the one shown above, can be instantiated as terms in other facets, with variables or expressions being associated with the formal parameters. This provides a form of structural composition of models. As an illustration, the Schmidt trigger facet might be instantiated in a sensor model as follows:

```
facet sensor (...) is
   data_in:: real;
   quantized_data:: bit;
   ...
begin continuous
   s1:
trigger(data_in,quantized_data);
      ...
end sensor;
```

Furthermore, different facets of a given system can be composed using facet operators to form a description covering the

different aspects of the system. One of the more common compositions is facet conjunction using the facet operator "and", which requires that all terms of both operand facets be true. Other composition operators include "or" for describing a system that may have several variants, "<=>" for asserting equivalence of two facets, and "=>" (implication) for asserting that one facet is a refinement of a more abstract facet. These and other composition operators are described in more detail in The Rosetta Usage Guide [1].

The domains referred to in the above descriptions are predefined sets of definitions that enlarge the syntactic and semantic vocabulary of the base language for particular kinds of modelling or computation. A domain definition consists of a collection of declarations and terms that are included in facets based on the domain. For example, the continuous facet used in the example model above defines the time variable t, and also defines the syntax and semantics of the "'" and "@" symbols. Rosetta predefines a number of domains that will be used in a wide variety of models. The fundamental domain is called logic, and includes definitions of the basic mathematical types, operators and expressions. Other predefined domains inherit from the logic domain, as shown in the following figure. The arrows represent homomorphism from more- abstract domains to domains which are more concrete in their representation of state or time.

Where a model is composed of facets from different domains, Rosetta *interactions* define the way in which terms in one domain imply properties in other domains. For example, an interaction that deals with conjunction of the logic domain and the continuous time domain might include a rule specifying that each term in the logic-domain facet is true for all time in the continuous-time domain.

While the Rosetta language is extensible through the definition of new domains and interactions, it is anticipated that most designers will only need to use the predefined domains and interactions. Development of new domains and interactions requires a deep understanding of the underlying semantics, and is expected to be performed by a relatively small number of application developers.

3. Overview of the spring example

The dual-spring design example is a case study specified by The US Air Force Materials Directorate to demonstrate an end-to-end Rosetta-based design activity. The case study requires design of a system comprising two coaxial wire springs, one shorter than the other. The inner spring must fit over a 1.0" diameter tube and must fit inside the outer spring. The outer spring must fit inside a 2.5" diameter tube. The maximum uncompressed length of the longer spring is 12.0", and the system must allow deflection of 6.0". For the first 4.0" of deflection, the system must exhibit a rate (force per unit deflection) of 50 lb/in, and for deflection between 4.0" and 6.0", the system must exhibit a rate of 75lb/in.

In addition to these problem-related requirements and constraints, equations are provided that relate the performance properties of a spring (rate and free length) to its physical properties (wire diameter, coil diameter and pitch, number of coils). Further constraints are specified for a spring to ensure that it does not assume a permanent deformation due to excessive compression. These constraints are expressed as inequalities in terms of the materials properties of the spring wire, including its modulus of rigidity, shear stress and shear yield stress.

4. The Rosetta spring model

The Rosetta model for the design example is comprised of two parts, presented in full in the Appendix to this paper. The first part is a parameterised model of a single spring, expressed as a Rosetta facet in the logic domain. We use this domain since we are dealing with a static model, not a dynamic model.

The formal parameters to the facet describe the physical characteristics of the spring: the wire diameter (*d*), the coil inner diameter (*ID*) and outer diameter (*OD*), the

Modulus of Rigidity of the wire (*G*), the number of coils (*N*), and the coil pitch (*p*). Within the facet, a number of internal items are defined, representing the derived quantities mentioned in the spring equations. These include the spring rate (*R*) and desired free length (*L*) used in determining the performance of the spring, and the material properties used in the constraints.

The spring facet models a spring in terms of equations and inequalities that relate these parameters and derived quantities. We use the convention of labelling constraint terms with labels C1, C2, etc., formula terms with labels F1, F2, etc., and assertion terms with labels A1, A2, etc. Term C1 simply relates the inner and outer diameters to the wire diameter. C2 and C3 are constraints on the spring constant and number of coils determined from experience and given in the problem statement. F1 is given in the problem statement as a formula relating several variables as follows:

This is simply translated into Rosetta using predefined mathematical operators. Formulas represented in F2 and F3 are likewise given in the problem statement. *N* is the total number of coils. This, multiplied by the wire diameter, gives the height of the full compressed spring. F4 gives the minimum free (uncompressed) length of the spring (*Lm*): the fully compressed height (*H*) plus the maximum compression that will be applied. However, since the spring rate is non-linear over the last 20% of compression, the desired free length (*L*) is set to be longer than the minimum free length. This is expressed in F5. The next term, F6, expressed the coil pitch (*p*) in terms of the free length, wire diameter and number of active coils. C4 is a constraint that ensures that the spring is "spring-like" rather than being a column of wire.

The remaining terms in the spring facet define the corrected shear force applied to the spring on full deflection (*Sk*) and the shear yield strength of the wire (*Sy*). These terms are transliterations of the formulas provided in the problem statement. Finally C5 is the constraint that prevents the applied shear force exceeding the yield strength. If this constraint is violated, the spring "takes on a set," that is, it suffers a permanent deformation.

The second part of the Rosetta model for the design example is a facet that describes the composite system. The spring_system facet declares a number of variables representing properties of the inner and outer springs, as well as the overall system deflection and applied force. The facet also includes two instances of the single-spring facet, and a collection of equations and constraints that parallel the informal specification provided in the problem statement.

In particular, the facet includes equations that relate the overall system deflection to the deflections of the individual springs, and the overall system rate to the rates of the individual springs. The Modulus of Rigidity of the springs is given in the problem specification, but the other physical characteristics are required to be determined from the required overall rate and the physical constraints. Hence the actual parameters for the spring instances, describing the physical characteristics, are free variables. Their values depend on the rate for each of the springs, determined from the overall rate, and the various constraints that apply to the individual springs and to the composite system.

The apparent complexity of the formulas describing deflection of the system and the individual springs comes about from the fact that the problem does not specify which of the two springs is the longer. For the system rate (the sum of the two spring rates) to change at a deflection of 4", one spring must be 4" longer than the other. The longer spring must have a rate of 50ln/in, and the shorter, engaged when the deflection reaches 4", must have a rate of an additional 25lb/in. In principle, this could be achieved with either spring being the longer. Care was taken in constructing the formula not to bias the solution. If only one alternative is feasible, that fact should be a consequence of the constraints upon the system rather than an *a priori* statement in the model.

The formula F6 and the assertion A1 are not strictly required in the model.

However, they are derived from statements in the problem definition that serve as clarifications of the rate specifications. They can be seen as "sanity checks," and so were included in the model as assertions for this purpose.

5. Evaluation of the model

The dual-spring system case study is one of the first modelling problems specified externally to the language development team to be attempted as part of the language validation process. It was undertaken largely by the first author after briefly reviewing the preliminary documentation on the language. Most of the effort in developing the model was spent in understanding the English-language and equation-based informal specification. Thereafter, expression in Rosetta was relatively straightforward. The major difficulties lay in determining which properties of the springs should be specified as parameters and which as exported properties. This question is still not clear, and we expect it will be treated as a matter of modelling style.

An important benefit of the declarative nature of the Rosetta description became evident during development of the model. The spring facet was developed from the perspective of starting with the physical parameters of the spring (e.g., wire diameter, coil diameter, etc.) and deriving the behavioural properties (rate, free length). However, when the facet was instantiated in a system model, the behavioural properties were given and the physical properties were to be determined. The declarative nature of the equations makes it possible to "drive the model backwards" in this way. Were the model expressed in an operational style, with variables being assigned from inputs, such an approach would be much more difficult.

In order to validate the Rosetta model, the first author prepared an Excel spreadsheet that encapsulated the formulas in the model. In fact, two spreadsheets were developed, one structured with the inner spring being the longer, and the other structured with the outer spring being the longer. This was done to make the Excel model manageable and solvable. The author attempted to solve the dual spring problem in each variation by using the solver facility in Excel. This facility allows the user to specify goals on dependent variable, constraints on independent variable, and to find values of independent variables that imply the goals. The author was not able to find a feasible solution to the dual-spring problem as specified. It remains unclear whether this is because the problem really is overconstrained, or whether the Excel model is too simplistic to allow identification of a feasible solution.

In addition to the Excel translation, an interface between Rosetta and the MATLAB environment has been developed. The MATLAB translation system takes parsed output from the standard Rosetta parser and generates equations suitable for evaluation in the MATLAB environment. An evaluation script was developed to evaluate the model over it operational environment. The model resulting from the automatic transformation evaluated favourably with respect to the hand generated Excel result.

6. Conclusion

The design case study described in this paper demonstrates the effectiveness of Rosetta as a specification and constraint language for describing physical systems such as the dual-spring system. In principle, the formal semantic basis of the language enables automatic tool-based analysis, such as checking for inconsistency, and automatic solution of equations.

While the power of Rosetta has been demonstrated in the domain of continuous, static systems, it is by no means limited to this domain. Other predefined domains allow description of the dynamic behaviour of system, including continuous-time, discrete-time and state-based models. it is expected that further case studies will demonstrate the use of the language for requirements and constraint specification in these domains. However, one of the most exciting capabilities will be the expression, with a formal semantic basis, of the interactions between these domains. This

remains one of the most significant contributions of the Rosetta language development effort.

References

[1] Alexander, P,. C. Kong, and D. Barton, "The Rosetta Usage Guide," University of Kansas Technical Report available at http://www.ittc.ukans.edu/Projects/rosetta/

[2] Alexander, P., C. Kong, and D. Barton, "The Rosetta Functional Requirements Specification Domains," *Proceedings of the Hardware Description Language Conference (HDLCON'00)*, March 2000, Los Angeles, CA.

[3] Alexander, P., R. Kamath, D. Barton, "System Specification in Rosetta," *IEEE Engineering of Computer Based Systems Workshop and Symposium,* April 2000, Edinburgh, UK.

[4] The VHDL International Systems Level Design Language Committee Web page, http://www.intermetrics.com/sldl

Appendix: The Rosetta spring model

```
facet spring ( d :: real;       // wire diameter
               OD :: real;      // outside diameter of spring
               ID :: real;      // inside diameter of spring
               G :: real;       // Modulus of Rigidity of wire
               N :: real;       // total number of coils
               p :: real        // pitch
             ) is

   export R, L;

   pi :: real is 3.1415926435898;

   D :: real is (OD + ID) / 2;  // mean diameter of spring
   C :: real is D / d;          // spring constant

   n :: real;                   // number of active coils
   H :: real;                   // fully compressed height

   R :: real;                   // Force/length (lb/inch)
   P :: real;                   // force to fully deflect spring

   l :: real;                   // maximum spring displacement
   Lm :: real;                  // minimum free length
   L :: real;                   // desired free length

   Tu :: real;                  // ultimate tensile strength of wire
   Ty :: real;                  // tensile yield strength
   Sy :: real;                  // shear yield strength
   S :: real;                   // uncorrected shear stress
   k :: real;                   // correction factor
   Sk :: real;                  // corrected shear factor

begin logic

   C1: (OD - ID) / 2 = d;       // dependencies between diameter parameters

   C2: C >= 4 and C =< 20;      // empirical constraints on spring constant

   F1: n = (G * d^4) / (8 * R * D^3);
```

```
   C3: n >= 3;                  // empirical constraint on n

   F2: N = n + 2;               // for closed and ground spring

   F3: H = d * N;

   F4: Lm = H + l;
   F5: L = H + (l / 0.8);

   F6: p = (L - 2*d) / n;
   C4: p =< D;                  // pitch can't exceed coil diameter

   F7: Tu = 200000 * D^(-0.14);
   F8: Ty = 0.75 * Tu;
   F9: Sy = Ty * 0.577;

   F10: P = R * l;

   F11: S = (8 * P * D) / (pi * d^3);
   F12: k = (4*C - 1) / (4*C - 4) + 0.615 / C;
   F13: Sk = S * k;

   C5: Sk < Sy;                 // shear force < wire shear yield strength

end spring;


// ---------------------------------------------------------------


facet spring_system is

   outer_d :: real;
   outer_OD :: real;
   outer_ID :: real;
   outer_G :: real is 11.5E6;   // music wire
   outer_N :: real;
   outer_p :: real;
   inner_d :: real;
   inner_OD :: real;
   inner_ID :: real;
   inner_G :: real is 11.5E6;   // music wire
   inner_N :: real;
   inner_p :: real;

   system_L :: real;            // system free length

   system_R :: real;            // system rate
   inner_R :: real;
   outer_R :: real;

   system_deflection :: real;
   inner_deflection :: real;
   outer_deflection :: real;
```

```
   force :: real;

begin logic

   outer_spring: spring( outer_d, outer_OD, outer_ID,
                         outer_G, outer_N, outer_p);

   inner_spring: spring( inner_d, inner_OD, inner_ID,
                         inner_G, inner_N, inner_p);

   C1: outer_OD < 2.5;          // outer spring must fit inside 2.5" tube
   C2: outer_ID > inner_OD;     // outer spring must fit over inner spring
   C3: inner_ID > 1.0;          // inner spring must fit over 1.0" tube

   F1: system_L = max(outer_spring.L, inner_spring.L);
   C4: system_L < 12.0;

   C5: system_deflection >= 0.0 and system_deflection =< 6.0;

   F1: inner_deflection =
          if system_deflection < (system_L - inner_spring.L) then 0
          else system_deflection - (system_L - inner_spring.L);
   F2: outer_deflection =
          if system_deflection < (system_L - outer_spring.L) then 0
          else system_deflection - (system_L - outer_spring.L);

   F3: inner_R =
          if inner_deflection = 0 then 0
          else inner_spring.R;
   F4: outer_R =
          if outer_deflection = 0 then 0
          else outer_spring.R;

   F5: system_R = outer_R + inner_R;

   C6: system_R =
          if system_deflection <= 4.0 then 50
          else 75;

   F6: force = system_R * system_deflection;

   A1: (system_deflection = 0 => force = 0)
       and (system_deflection = 4.0 => force = 200)
       and (system_deflection = 6.0 => force = 350);

end spring_system;
```