A RAISE Specification Framework and Justification Assistant for the Duration Calculus

Anne Haxthausen [*] IT/DTU

Xia Yong [†] UNU/IIST

July 21, 1998

1 Introduction

RAISE is a product consisting of a development method [RMG95], an associated formal specification language (RSL) [RLG92] and a collection of computer-based tools. RAISE has turned out to be useful in the development of many kinds of software systems. However, RSL has no "real-time" features, and hence, it is difficult to specify real-time applications using RSL.

The Duration Calculus (DC) [ZHR91] is a formalism which can be used for that. However, DC is "just" a logic, and for practical purposes it would be nice to have a richer language providing modules, facilities for declaring symbols to be used in DC formulas etc. The goal of our work is two-fold: (1) to extend RSL with real-time features, and (2) to provide a specification language and tools support (e.g. a syntax and type checker, a justification assistant, etc.) for DC.

A first step towards this goal, could be to combine DC and RSL achieving the power of both. So far, we have encoded DC in RSL and set up a proof assistant tool to verify (encoded) DC formulas using the RAISE Justification tools.

This paper first introduces a RAISE specification framework for DC, and then explains how the justification assistant, called DC/RJ$^-$, is set up. Finally, a conclusion and topics for future work are given.

2 A RAISE specification framework for DC

The base of our RAISE specification framework for DC is a library of RSL modules providing a language of encoded DC formulas and their semantics. Using this library one can declare DC symbols such as state variables and write (encoded) DC formulas in RSL. Below, we give a introduction to the encoding in RSL.

[*]Department of Information Technology, Building 344, Technical University of Denmark, DK-2800 Lyngby, Denmark; ah@it.dtu.dk

[†]UNU/IIST, P.O.Box 3058, Macau; xy@iist.unu.edu

2.1 Encoding of DC in RSL

State assertions, terms and formulas are encoded as values of the following RSL types:

```
type
  StateAssertion = {| e : Time → Bool • is_finitely_variable(e) |},
  Term = Interval → Real,
  Formula = Interval → Bool
```

where

```
type
  Time = Real,
  Interval = {| (t1, t2) : Time × Time • t1 ≤ t2 |}
```

and *is_finitely_variable(e)* is a predicate ensuring the property of finite variability stated in [MZ97].

The DC library provides these types and a value constructor for each DC operator. For instance, for the chop operator we have the following declaration:

```
value
  ⁀ : Formula × Formula → Formula
  f1 ⁀ f2 ≡
    (λ (t1, t2) : Interval •
       ∃ t : Time • t1 ≤ t ∧ t ≤ t2 ∧ f1(t1, t) ∧ f2(t, t2)),
```

Hence, a DC formula $F1 \frown F2$ can be encoded as $f1 \frown f2$, where $f1$ and $f2$ are the encodings of the sub-formulas $F1$ and $F2$.

A specification in our specification framework contains declarations of DC symbols such as state variables, and of axioms of the form $hold(f)$, where f is an encoded DC formula. An axiom of the form $hold(f)$ restricts the possible interpretations of the declared DC symbols to those that make the formula f true in any interval. (Note that this does *not* mean that f is valid).

The full library can be found in [Xia97].

2.2 Example

The requirements to the gas burner in [ZHR91] can be formulated in the encoded language as follows:

```
scheme BURNER_1 =
  extend DC with
  class
    value
      /* declarations of state variables */
      Gas, Flame : StateAssertion,
      Leak : StateAssertion = and(Gas, not(Flame))
    axiom
      [Req] hold(implies(length ≥ lift(60.0), lift(20.0) * dur(Leak) ≤ length))
  end
```

where and, not, implies, length and dur are the encoded symbols corresponding to the DC operators $\wedge$, $\sim$, $\Rightarrow$, $\mathbf{l}$ and $\int$, and, lift(r) is the encoding of a real number r.

And, the two design decisions can be formulated as follows:

```
scheme BURNER_2 =
  extend DC with
    class
      value
        Gas, Flame : StateAssertion,
        Leak : StateAssertion = and(Gas, not(Flame)),
        Des1 : Formula = allsub(implies(everywhere(Leak), dur(tt) ≤ lift(1.0))),
        Des2 : Formula =
         allsub
           (implies
             (somesub(everywhere(Leak)) ⁀ somesub(everywhere(not(Leak))) ⁀
                somesub(everywhere(Leak)),
              dur(tt) ≥ lift(30.0)
             )
           )

        axiom [DES1] hold(Des1),  [DES2] hold(Des2)
      end
```

where allsub, somesub and everywhere are the encoded symbols corresponding to the DC operators $\Box$, $\Diamond$ and $[\![.]\!]$

2.3 A DC+RSL syntax extension

We plan (cf. section 5) to make a DC+RSL extension of RSL with shorthands for declaring state variables and DC requirements so that the specifier should not bother with the encoding. For instance, a DC+RSL shorthand for the BURNER_1 module shown above could be

```
scheme BURNER1 =
  class
    state variables
      Gas, Flame: Bool,
      Leak : Bool = Gas ∧ (∼ Flame)
    DC requirements
      [Req] l ≥ 60.0 ⇒ 20.0 ∗ ∫Leak ≤ l
  end
```

3 The DC/RJ$^-$ justification assistant

Now, one could in principle use the RAISE justification tools to do proofs about encoded DC formulas employing the semantic definitions of the encoded formula constructors. However, for efficiency reasons we prefer to do proofs at the "syntactical level" using the DC proof system. Therefore, we have encoded the DC proof system as a collection of RAISE theories that can be employed by the RAISE justification tools.

For instance, the DC axiom

$$(F1^\frown F2)^\frown F3 \Leftrightarrow F1^\frown(F2^\frown F3)$$

is encoded as the following RAISE theory:

theory A2 :
  axiom [CHP_ASSOC]
    in DC ⊢ ∀ F1, F2, F3 : Formula • (F1 ⌢ F2) ⌢ F3 ≡ F1 ⌢ (F2 ⌢ F3)
end

The theories comprise:

1. Encoded basic rules (the 9 axioms of IL, 6 inference rules of IL, and 2 induction rules of DC etc., but not the 6 axioms of DC as they are already included in the definition of DC for the axiomatisation of the duration operator.)

2. Encoded theorems of DC, IL and classic Modal & Temporal Logic, Sentence Logic and First Order Predicate Logic, etc.

All these theories can be found in [Xia97].

The RAISE justification tool set together with the encoded language and theories is called 'the DC/RJ$^-$ justification assistant'. The structure of DC/RJ$^-$ is shown in figure 1, where the arrows indicate the use hierarchy.

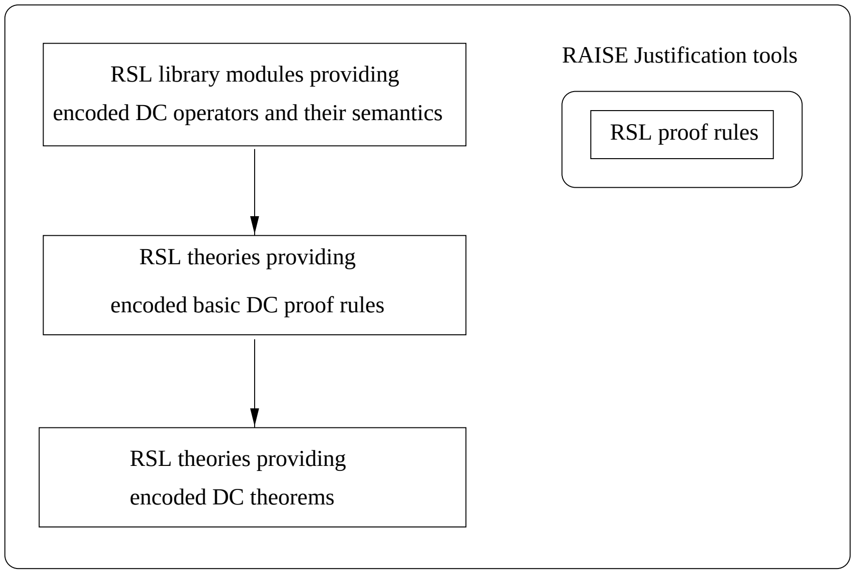
RSL library modules providing encoded DC operators and their semantics
RSL theories providing encoded basic DC proof rules
RSL theories providing encoded DC theorems
RAISE Justification tools
RSL proof rules

Figure 1: The structure of DC/RJ$^-$

3.1 Soundness of axioms and correctness of theorems

We have checked the soundness (wrt. the semantic definitions) of some of the (encoded) basic rules, and believe that we can prove the soundness of all of them.

The (encoded) theorems can be proved from the encoded basic rules except in a few cases, where we need to employ semantic definitions of the formula, term and state assertion constructors.

3.2 Application of DC/RJ$^-$ to an example

In section 2.2 we formulated in the RSL encoded DC language the requirements and design decisions for the gas burner example [ZHR91]. We have used DC/RJ$^-$ to prove that the design decisions in BURNER_2 are refinements of the requirements in BURNER_1, i.e. that the following theory is true:

theory BURNER21_DEV :
  axiom $\vdash$ BURNER_2 $\preceq$ BURNER_1
end

The proof can be found in [Xia97].

4 Conclusions

In this paper we have presented a RAISE specification framework and a proof assistant (DC/RJ$^-$) for DC based on an encoding of DC in RSL.

4.1 Comparison with related work

Other proof assistants have been created for DC: DC/P$^-$ [MXW96] and PC/DC [Ska94]. These are both based on an encoding in PVS, while our DC/RJ$^-$ is based on an encoding in RSL. Our strategy for encoding is similar to that used for PC/DC, but differs in certain aspects from that used for DC/P$^-$: First order predicate logic is embedded in DC/P$^-$, while PC/DC and DC/RJ$^-$ use the built-in first order predicate logic of PVS and RSL, respectively.

An advantage of using RSL rather than PVS is the fact that RSL is not just a requirement specification language, but a wide spectrum language that can be used in all phases of software development ranging from requirement specification to design of imperative and concurrent programs. When the RSL process algebra has been extended with time as proposed in next section, we will for instance be able to specify a concurrent program in RSL for the gas burner example. A further advantage of RSL that we exploited in the gas burner example, is its associated method for stepwise refinement with a notion of development relations. On the other hand, an advantage of using PVS rather than RSL is that its proof assistant has more powerful decision procedures than the RAISE justification tool. That would have been useful in our proof of the development relation BURNER21_DEV.

A Z specification framework similar to DC+RSL has been proposed by Engel [Eng94].

5 Future work

Topics for future work include:

- Designing syntax and context rules for the DC+RSL language and giving semantics to it by defining a transformation to the DC encoding in RSL.
- Making a syntax and type checker for DC+RSL and a parser which can make the encoding of DC+RSL into RSL. A decoder (unparser) of encoded proofs would also be useful.
- Extending the RSL process algebra with timing constructs, obtaining a timed process algebra a la Timed CSP [Dav93] and Timed CCS [Wang91], and give this timed RSL language a DC semantics. Giving it a DC semantics would provide a base for defining what it means for a timed RSL process to satisfy a DC requirement (expressed in the DC+RSL language).
- Case studies.

6 Acknowledgements

The ideas of encoding and subsequent usage of the RAISE justification tools were originally suggested by Søren Prehn, during his assignment to UNU/IIST.

We would like to thank Zhou Chaochen, Chris George, Anders P. Ravn, Xu Qiwen and P.K.Pandya for discussions on DC, RAISE and proof tools, and Bo Stig Hansen for comments on a draft version of this paper.

References

[Dav93] Jim Davies. *Specification and Proof in Real-Time CSP.* Cambridge University Press, Distinguished Dissertation Series, Cambridge, 1993.

[Eng94] Marcin Engel. *Specifying Real-Time Systems with Z and the Duration Calculus.* In proceedings of the Eighth Z User Meeting, Springer-Verlag, 1994.

[MXW96] Mao Xiaoguang, Xu Qiwen, and Wang Ji. *Towards a Proof Assistant for Interval Logics.* UNU/IIST Report No. 77, UNU/IIST, International Institute for Software Technology, P.O. Box 3058, Macau, 1996.

[MZ97] Michael R. Hansen, and Zhou Chaochen. Duration Calculus: Logical Foundations. *Formal Aspects of Computing*, 1997.

[RLG92] The RAISE Language Group. *The RAISE Specification Language.* The BCS Practitioners Series. Prentice Hall Int., 1992.

[RMG95] The RAISE Method Group. *The RAISE Development Method.* The BCS Practitioners Series. Prentice Hall Int., 1995.

[Ska94] Jens Ulrik Skakkebæk. *A Verification Assistant for a Real-Time Logic.* PhD thesis, Department of Computer Science, Technical University of Denmark, 1994.

[Wang91] Wang Yi. *A Calculus of Real Time Systems.* PhD thesis, Department of Computer Sciences, Chalmers University of Technology, Göteborg, Sweden, 1991.

[Xia97] Xia Yong. *DC/RJ$^-$: A Justification Assistant for Duration Calculus.* UNU/IIST Report No. 126, UNU/IIST, International Institute for Software Technology, P.O. Box 3058, Macau, 1997.

[ZHR91] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1991. Revised June 3, 1992.