Technical Report No: 2003/06

A Reference Model for Requirements Engineering

Jon G Hall
Lucia Rapanotti

2003

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk

A Reference Model for Requirements *Engineering*

Jon G. Hall Lucia Rapanotti
The Open University, UK
{J.G.Hall, L.Rapanotti}@open.ac.uk

Abstract

The Reference Model of Gunter et al., 2000, provides a framework for describing and analyzing key software engineering artifacts and their properties. In this paper we propose a reification of this framework in which behaviour is explicitly trace-based. We find that this benefits the formalism in adding structure in ways which are meaningful and practical from an engineering viewpoint. In particular, we develop notions of points of introduction and reachability in the new framework, and show how they strengthen the properties of the Reference Model.

1. Introduction

The Reference Model of Gunter et al., 2000, [1] provides a framework for describing and analyzing key software engineering artifacts and their properties. This model builds upon previous work in the literature from the same authors [6, 10] and others [9]. A major contribution of the framework is the statement of fundamental relationships and properties between the artifacts. These are universal truths which hold independently of any particular notation or application domain. Therefore, the Reference Model can be seen as providing a unifying characterization of software artifacts and their relations.

This fact has important practical ramifications. Should an engineer choose to adopt the framework within a particular software development process, then the analysis of the problem space is greatly simplified by the separation of concerns and the set of proof obligations provided by the framework. There are, however, issues related to the applicability of the framework from the engineering viewpoint, which need some consideration, such as, for instance, its expressiveness and level of abstraction and its accessibility to practitioners.

The formulation of Gunter et al., 2000 is based on a first order logic and makes no explicit reference to time. From a theoretical viewpoint, this provides a powerful abstraction and the necessary generality. From a practical viewpoint, however, it under-constrains the problem space so that there is no clear-cut separation of what is theoretically expressible from what is practically relevant. In practice, this separation has to be explicitly dealt with by a highly skilled engineer, which reduces the utility of the model from a practical viewpoint.

In this paper we propose a reification of the Reference Model with the introduction of time intervals. We claim that such a reification allows the grounding of the framework better to address engineering issues, such as reachability, and expect that it will facilitate the provision of practical tools for the analysis of the problem space.

The paper is structured as follows. In Section 2, we recall the Reference Model of Gunter et al. 2000 [1]. In Section 3, we justify our reification of the model. In Section 4, we provide a formalisation which introduces time and traces of behaviours. In Section 5, we discuss some of the benefits of the reification. Finally, Section 6 concludes the paper.

2. The Reference Model and its properties

The Reference Model (RM) of Gunter et al., 2000, [1] is based on the widely accepted separation between the system and its environment, a number of key artifacts which pertain to the two, and a vocabulary, which is used to describe the environment, the system and their interface. The key artifacts are the following. The domain knowledge, W, is what we know about the environment. The requirement, R, is what the customer requires of a system working within the environment. The specification, S, is a description of the system, which can be used for its implementation. The program, P, is the system implementation. Finally, the programming platform, M, is the machine within which the program runs. The way these key artifacts relate to environment and system is illustrated in Figure 1.

In the RM, the vocabulary (a.k.a. *designation*), is used to provide names to describe the environment, the system and their artifacts. More precisely, the vocabulary is used to name phenomena: typically, states or events. Phenomena are grouped into those which belong to and are controlled by the environment, denoted by e, and those which belong

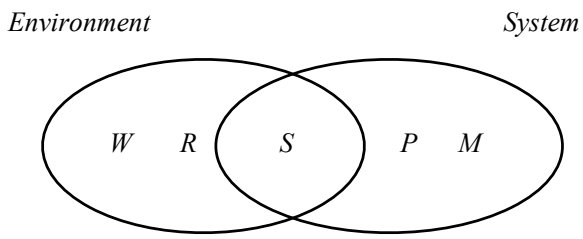

Figure 1. Key artifacts of the Reference Model.

to and are controlled by the system, denoted by s. The two sets e and s are disjoint.

Control is mentioned in the RM, but with no explicit semantics. We use the definition of [10], and return to discuss control in Section 5.1. For the moment, the reader should interpret the notion of control loosely, as something which is exercised by either the environment (for e) or the system (for s).

The sets e and s are further partitioned into phenomena which are hidden and visible:

$$e = e_h \cup e_v \text{ and } e_h \cap e_v = \varnothing$$

$$s = s_h \cup s_v \text{ and } s_h \cap s_v = \varnothing$$

where phenomena in e_h are hidden, and in e_v are visible, outside the environment. Similarly for s_h and s_v.

The RM assumes that the specification, S, lies in the environment's and the system's common vocabulary (see Figure 1). Therefore, S can only use terms denoting phenomena in $s_v \cup e_v$. On the other hand W and R can use terms denoting phenomena in $e \cup s_v$.

The RM introduces a factorization of the artifacts which allows for the specification S to represent the combination of P and M, that is the program and the machine within which the program executes. This factorization allows one to focus on the relationships between problem and solution through the specification and without worrying about any particular implementation details. Also, thanks to a transitive condition introduced in the RM, such relationships extend to P and M. Therefore, for brevity, in the reminder of the paper, we will omit considerations of P and M and will work in the environment part of the framework.

The universal relationships between software artifacts expressed in the RM are captured by the following properties:

I. Consistency of the domain knowledge:

$$\exists\, e\, s \bullet W$$

II. Adequacy:

$$\forall\, e\, s \bullet W \wedge S \Rightarrow R$$

III. Relative consistency:

$$\forall\, e_v \bullet (\exists\, e_h\, s \bullet W) \Rightarrow (\exists\, s \bullet S) \wedge \forall\, s \bullet (S \Rightarrow \exists\, e_h \bullet W)$$

Intuitively, property I states that the (description of the) environment must make sense in some configuration. This is an obvious starting point: we can't hope to specify a sensible system for a non-sensible world.

Property II embodies the proof obligation that we are specifying the right system, that is one that satisfies the requirements. Again, we should not aspire to specify systems that don't satisfy our customers. This second property only makes sense if the first holds, that is if we start with a sensible environment.

Property III is less intuitive. Let us decompose it into two parts. The first part is:

$$\forall\, e_v \bullet (\exists\, e_h\, s \bullet W) \Rightarrow (\exists\, s \bullet S) \qquad \text{III(a)}$$

This says that whenever there is a behavior satisfying the environment W, then there must be a behavior (which has the same visible events e_v) satisfying the specification S.

The second part is:

$$\forall\, e_v \bullet (\exists\, e_h\, s \bullet W) \Rightarrow \forall\, s \bullet (S \Rightarrow \exists\, e_h \bullet W) \quad \text{III(b)}$$

This means that whenever there is a behavior satisfying the environment W, then for all behaviors s satisfying the specification S, there must be a behaviour satisfying the environment W. This last condition has profound implications on the nature of the relationship between environment and machine. We will return to this point in Section 4.

Although not explicit in the original paper, the RM appears to assume complete, i.e., potentially infinite, traces of behaviour. Moreover, property III does not work locally on traces, i.e., through property III it is possible for two traces, not sharing a prefix, to be related. This we find counter-intuitive for a behavioural model—the behaviour of a system evolves through time and surely should only be constrained by properties that can act in time. This is one of the foundations of the process algebraic approach [8] and our primary concern in this paper.

3. Reifying the Reference Model

In this section we discuss our main motivations in reifying the RM.

For illustration, let us introduce the following example. Consider a nuclear reactor and a controller which monitors and regulates the kernel temperature. For this purpose, W can be described by two equations. The first one relates to

the kernel temperature, which is goverend by the laws of thermodynamics:

$$temp(t) = \int_0^t react(x) - loss(x)dx + temp_0$$

The second equation relates to the sensor, which samples and discretises the temperature at 1 second intervals, but melts above $20,000^oK$ and fails unsafe:

$$sensor(t) = \begin{cases} \lfloor temp(\lfloor t \rfloor) \rfloor & temp(t) \leq 20,000^oK \\ 0 & \text{otherwise} \end{cases}$$

Assume the requirement *R* to be that the controller prevents the temperature from rising above $15,000^oK$, that is:

$$\forall\, t \bullet temp(t) \leq 15,000^oK$$

Finally, assume that the controller specification *S* can be described by a single expression:

$$sensor(t) > 10,000^oK \Rightarrow \forall\, T > t \bullet react(T) = 0$$

which otherwise does not constrain *react*.

3.1. Points of introduction

From an engineering perspective it is legitimate to ask at which point in the life of the reactor is it appropriate to introduce the controller. A sensible assumption on initial conditions is that, when *S* is introduced to *W*, the temperature is much less than $10,000^oK$. Clearly, one would not wish to introduce the controller when the reactor is out of control (or even close to being out of control), as this would certainly prevent the requirements from being satisfied. There is also the obligation to establish that the requirements are not incompatible with the environment, that is it is possible for the environment to reach at least one state in which the requirements are satisfied. We argue that such points are those at which it makes sense to introduce the specification to the environment. In this way, we come to a new formulation of the first property of the RM that could be expressed intuitively as follows:

A there are points in time when the environment makes sense and the requirements are satisfied.

We call those times *points of introduction*. This property is much easier to discharge than the original property I which required a complete behaviour to be found, more than just an initial state, and so is much easier in practical engineering terms.

3.2. Pruning the space of analysis

Assuming that the specification will only be introduced to the system at a possible point of introduction, certain environment states become 'unreachable'. For instance, in the example, should the controller be introduced to the reactor when the temperature is well below $10,000^oK$, a state in which the kernel temperature is above $15,000^oK$ should never be reachable. Should we, therefore, have to consider establishing the requirements for behaviours that extend these unreachable behaviors? If we were to use property II of the RM, this is exactly what would be required. Another condition suffices from a practical viewpoint, that is for property II to hold at the moment the controller and reactor come together and at each subsequently reachable state—until either the reactor is shut down (so the environment ceases to exist) or the controller is decommissioned. Property II could therefore be formulated intuitively as:

B at every point reachable from the chosen point of introduction, environment and system together satisfy the requirements.

This intuitive reformulation has the immediate practical appeal of reducing the space of analysis to those situations which are meaningful from an engineering viewpoint, in particular to when the system is introduced to the environment and from that time onwards.

3.3. Reachability

There is a further simplification of the space of the analysis that we can make. This has to do with the notion of reachability mentioned above. To develop such a notion of reachability, we look at property III of the RM. We recall that this property expresses mutual consistency between *W* and *S*. Specifically, it raises an obligation to prove that whenever a visible phenomena occurs in the environment, then the specification must behave sensibly, and this behaviour must not falsify the environment's constraints. We claim that the co-constraining of *W* and *S* produces a reachability structure in the problem space, that can be exploited to discharge proof obligations in an inductive fashion. Only behaviours on this structure are required to be considered in the establishment of the requirements *R*. In other words, when we look to show that $W \wedge S \Rightarrow R$, we are actually only requiring that this holds from an actual initial condition, for all configurations *reachable under W and S*. In general, this is strictly weaker than the statement of the RM which asks that requirements are established at all configurations that satisfy *W* and *S*.

This notion of reachability feels a rather important addition to the reference model from an engineering perspective. In particular, it provides a restricted form of satisfaction in property II of the RM. A precise definition of the reachability structure requires us to introduce some formalism, which we do in the following section.

4. Introducing time

In the RM, the environment exists as an *indicative* description, that is something that is given and cannot be changed (at least not because of the needs of software development). For instance, W could embody some physical laws, or the protocol of a legacy software system. On the other hand, a specification is *optative*, that is it expresses what one would *like to be true* of the built system. In this sense, the given environment description is fixed in the design process, and any introduced or designed system must therefore work within the constraints placed on it by the given environment. Later we make precise this notion of immutability in the design process.

In this section, we consider the specification of a digital system S to control a digital and/or continuous real-world environment W, where control is seen as constraining the possible behaviour of W. We begin from the RM's third property focusing on the ways in which S can be required to perform. For illustration, let us consider another simple example.

A three-setting convection heater is used to heat a room. The three settings are $0, 1$ and 2 kW. The heater contains a sensor that samples the temperature of the room every second, remaining constant until the next sample is taken. The room has heat sources other than the heater (the sun, perhaps) and loses heat in many ways (e.g., through the roof and walls, etc.). A model for the room, the sensor and the heater is given by:

$$\mathit{temp}(t) = \int_0^t \mathit{heater}(x) - \mathit{loss}(x)dx + \mathit{temp}_0$$

$$\mathit{sensor}(t) = \lfloor \mathit{temp}(\lfloor t \rfloor) \rfloor$$

$$\mathit{heater}(t) \in \{0, 1, 2\}$$

Our first task is to acknowledge that S will have a point of introduction to its environment, W, the point in time when it first asserts its control over W; any behaviour that occurs during S's interaction with W will be after this point. To be precise, before S is introduced to W we assume that the behaviour of W has been unconstrained or, possibly, constrained by other systems that are replaced by S.

Precisely because of this, the possibility of change in the indicative description of S's environment exists. It is therefore of practical importance to be able to talk of 'before' and 'after' this change. Although the notion is more complex, for the purposes of this paper, we assume only that S is introduced to a world W at time T_S, from which before and after are derivable. It is therefore the *unconstrained* behaviour of W before T_S that we constrain with S.

Moreover, that S is introduced at T_S implies that W existed at T_S, and also before. If necessary[1] we may introduce the time point $T_0 < T_S$ to indicate when W 'came into existence', i.e., the time point at which W as indicative description began to apply. For a designed environment, this is the point in time when its description becomes indicative.

We assume that time, $\mathbb{T}$, is continuous, isomorphic to a connected subset of the reals; given the above it may be an interval bounded below by T_0, but it will certainly contain T_S.

We make explicit the time dependency of the designation, by defining e_h, e_v and s on this interval. e_h is assumed to consist of continuous and/or discrete timed variables (either could be empty): $e_h : \mathbb{T} \to U$ where U is a 'universal' domain that contains all values of interest. e_v and $s : \mathbb{T} \to U$ are assumed to be discrete and finitely variable over any bounded interval. As e_v and s may both contain events, we identify an event with its characteristic subset of $\mathbb{T}$, i.e., those times at which the event occurs.

With reference to the heater example:

$$e_h = \langle \mathit{temp}, \mathit{loss} \rangle$$

$$e_v = \langle \mathit{sensor} \rangle$$

$$s = \langle \mathit{heater} \rangle$$

4.1. Considering behaviours

The detailing of the form of behaviours leads us to be very much more specific as to the relationship between (possibly continuous) behaviours, events and other shared phenomena. Our aim is to consider behaviours as periods of continuous evolution punctuated by discrete events. To this end, we would like to be able to talk about W as a set of traces. Therefore, we associate with W the set

$$\overline{W} = \{ \left\langle \begin{smallmatrix} e_h \\ e_v \\ s \end{smallmatrix} \right\rangle_1 \frown \ldots \frown \left\langle \begin{smallmatrix} e_h \\ e_v \\ s \end{smallmatrix} \right\rangle_n \frown \ldots \}$$

where $\frown$ indicates concatenation, $\left\langle \begin{smallmatrix} e_h \\ e_v \\ s \end{smallmatrix} \right\rangle_i$ is a vector of functions (that we term a *triple*) over an interval I (termed the *supporting interval* for that triple), with e_v and s constant thereon. Because we are interested in the behaviours of W and S together, we will assume that the first I begins with the point of introduction of S, T_S. Moreover, we will assume that each I is left closed, right open and maximal with respect to this property[2]. $\left\langle \begin{smallmatrix} e_h \\ e_v \\ s \end{smallmatrix} \right\rangle_i$ and $\left\langle \begin{smallmatrix} e_h \\ e_v \\ s \end{smallmatrix} \right\rangle_{i+1}$ overlap on the end point of the closure of their respective supporting intervals. We assume that the set $\overline{W}$ is prefix closed.

[1] And if W is a physical system, i.e., one to which only physical laws apply, this might not be strictly necessary.

[2] This prevents problems with multiple values at the end points.

Similarly, S can be treated in this way. The differences would be that S does not have access to e_h and so should, in theory, only be vectors of height two. In fact we finesse this point in this paper, and consider a triple for S as having three components, with that representing e_h being $\perp$. Moreover, as we can only be sure that S exists at T_S, the first triple in $\overline{S}$ will have a supporting interval that begins at T_S. $\perp$ will also be used when there is a value we do not care about.

Even though W (resp. S) is a predicate and $\overline{W}$ (resp. $\overline{S}$) a set, when context allows we will write both as W (resp. S).

Figure 2 illustrates the punctuation of continuous behaviour by discrete events for the heater example.

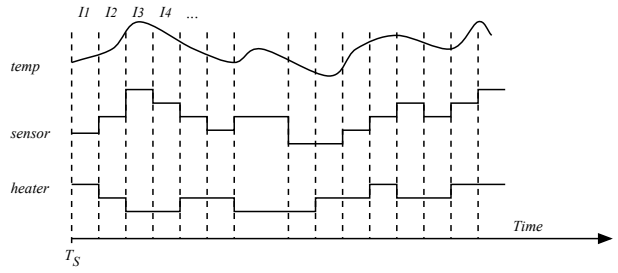

Figure 2. Supporting intervals for the three-setting heater.

The consideration of behaviours as periods of continuous evolution punctuated by discrete events is not new, forming a major contribution of [7]. In fact, our approach is reminiscent of their move from 'super-dense computations' (which, we assert, are required by the RM when e_h has a continuous component) to the simpler and more tractable 'sampling computations'. In essence, we identify e_v and s with the 'important events' of [7], and view the behaviour of the world and controller accordingly.

4.2. Reachability

We can now restate property III in our new framework as Property C. Again there are two parts, Property C(a) states that:

$$\forall p \in W \bullet (\exists \left\langle \begin{smallmatrix} e_h \\ e_v \\ s \end{smallmatrix} \right\rangle \bullet p^\frown \left\langle \begin{smallmatrix} e_h \\ e_v \\ s \end{smallmatrix} \right\rangle \in W \Rightarrow \exists \left\langle \begin{smallmatrix} \perp \\ e_v \\ s \end{smallmatrix} \right\rangle \bullet p^\frown \left\langle \begin{smallmatrix} \perp \\ e_v \\ s \end{smallmatrix} \right\rangle \in S)$$

To unpack this a little: The first quantification (over prefixes of $\overline{W}$) establishes a time point at which e_v changes (the closure of the last interval that supports p). The existential quantification provides a) a period of time (interval) in which e_h changes (continuously and/or discretely), and b) an interval in which an event in s may or may not occur. The implication is that there should be an interval that acts as witness in which the machine S expresses its constraints on W. Note, however, that the length of the interval supporting the respective triples need not be the same.

In essence, if W wishes to progress, then S must accept the e_v and allow that *or some other* behaviour to occur. In logical terms, S must not become *false*. Figure 3 illustrates this property. In the figure, from the prefix p, if there exists a step consistent with W (arc labeled by W), there must exist a step consistent with S (arc labeled by S and W).

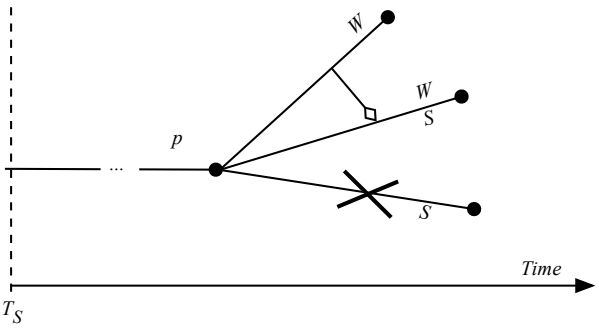

Figure 3. Upper part, Property C(a): If W wants to progress, it must not be prevented from doing so by S, although S may constrain which behaviours of W are possible. Lower part, Property C(b): S cannot add behaviours.

Property C(b) is the (almost) symmetric situation with S and W interchanged, the only asymmetry being the universal quantification over e_v:

$$\forall p \in W \bullet \forall \left\langle \begin{smallmatrix} \perp \\ e_v \\ s \end{smallmatrix} \right\rangle \bullet (p^\frown \left\langle \begin{smallmatrix} \perp \\ e_v \\ s \end{smallmatrix} \right\rangle \in S \Rightarrow \exists \left\langle \begin{smallmatrix} e_h \\ \perp \\ \perp \end{smallmatrix} \right\rangle \bullet p^\frown \left\langle \begin{smallmatrix} e_h \\ e_v \\ s \end{smallmatrix} \right\rangle \in W)$$

In essence, all behaviours of S must be 'acceptable' by W: this is illustrated in Figure 3 by labelling that behaviour with both S and W. The figure also indicates that the property prevents behaviours acceptable by S only. Note that the binding of e_v in the second existential quantifier determines the length of its supporting interval, and so that of the witness for the existential quantification.

The effect on W of the control of S is to prime W's reachability tree. This can be seen in Figure 4. The figure illustrates that each future step of the system must map to a step which is acceptable by both W and S, therefore only the latter needs be considered. We therefore see that property C defines a reachability relation over the satisfying states of W, with W's 'natural' reachability relations modified by the influence of S.

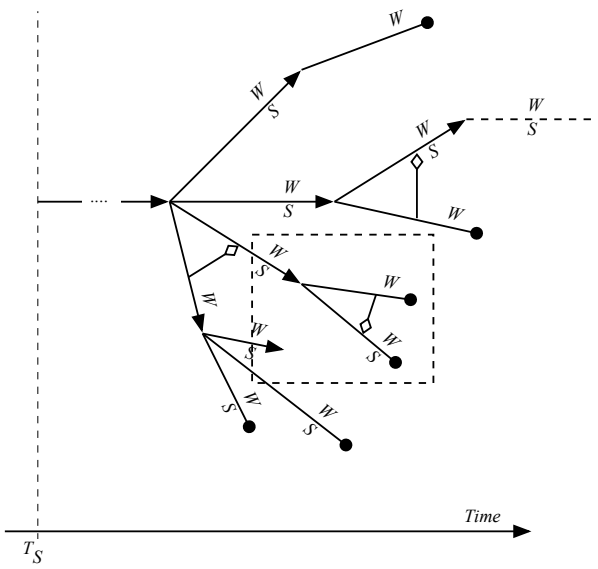

Figure 4. Property C induces a reachability structure on *W* as controlled by *S*.

5. Discussion

Practically, a reference model leads to proof obligations that need discharging. We briefly summarize those arising from the model proposed in this paper.

From property A, the engineer is required to identify possible points of introduction. Practically, this amount to identifying environment's states at which the system can be brought into operation in order to guarantee that the requirements can be met. It also amounts to demonstrating that there exist environment states when the requirements can indeed be satisfied, i.e., that there is no contradiction between environment and requirements.

From property B, the engineer is required to demonstrate that once introduced to the environment, the system satisfies the requirements and continues to do so for each reachable configuration. This proof obligation is facilitated by property C, which defines the reachability structure. The model guarantees that no stepping outside the structure is possible, hence no proof obligation exists beyond it.

5.1. Control and Progress

We use the notion of control of [10], where machine controlled phenomena (i.e., anything in *s*) cannot be under the control of the environment, and *vice versa*. In our model of behaviours, this notion of control applies to the extension of triples: at time *t*, if an event in e_v occurs, *S* cannot prevent it happening, and *vice versa*. Events in e_h are similarly not preventable by *S*. (The notion of prevention here appears related to the refusals of CSP [3].)

Our move to a reachability-based reference model allows us to make the following observation: that over any interval of time, *S* perceives progress in *W* (i.e., progression through its reachability tree) only through the occurrence or change of an e_v. Based on this, over any interval, there are three distinct cases:

- there is an occurrence or change in an e_v because *W* is active and progressing, and *S* can be sure that *W* is existing (i.e., has not become identically *false* as a description);
- there is no occurrence or change in an e_v because, although *W* is active and progressing, such progression does not require a change in e_v;
- there is no occurrence or change in an e_v because *W* has ceased to exist.

Of course, over any finite interval, cases 2 and 3 cannot be distinguished. Should *S* be required to stop on the demise of *W*, the desinger of *S* must have some information as to the time when *W* will stop. It is for such an *S* that we coin the term *Design for Control*, or *DfC* for short. DfC arises from the asymmetry described above in the relationship between *W* and the specification. In its most basic form, DfC is the asymmetry induced by the 'real-time existence' of *W* versus the 'machine-time existence' of the *S*. In this sense DfC is the service-based nature of a machine. We believe DfC to be an important corollary of our interpretation of the reference model in that it provides a strong characterisation of *S* in the context of *W* which the original did not.

6. Conclusions

In this paper we have proposed a reification of the Reference Model of Gunter et al., 2000 [1], which explicitly uses time, and which thus can look at the development of behaviours over time. We have shown how this benefits the formalism in adding structure in ways which are meaningful and practical from an engineering viewpoint. In particular, we have developed the notion of 'point of introduction', as establishing the initial conditions for the introduction of a system to an environment, and a notion of reachability of behaviours within the space of problem analysis, based on the co-constraining of *W* and *S*. Our investigation has also led to the identification of a new form of control, 'Design for Control (DfC)', which captures the asymmetry existing between an environment and any machine designed for that environment, and leads to proof obligations to be discharged in the design of such machines.

This work is part of a wider research effort by the authors and colleagues towards the the development of a semantic foundation of Problem Frames [5]. We assert in [2] that a problem diagram with solution, as used in the Problem Frames framework, can be seen as a $W, S \vdash R$ triple, the entailment being justified by correctness argument (provided, for instance, through decomposition to basic Problem Frames with their associated frame concerns). Under this semantics, an unsolved problem diagram is a 'challenge' to find an S together with a correctness argument. As $W, S \vdash R$ triples, solved problem diagrams depend on the Reference Model for their formal properties, and the reification thereof presented in this paper represents a first significant step towards understanding what is required in the discharge of correctness arguments, and from which the importance of our derived notion of reachability from a point of introduction can be seen.

[2] develops a formal semantics that underpins the use of the Problem Frames framework, but much work is still required. In particular, an immediate need is the validation of the framework in the design of realistic digital systems, and the evaluation of their impact on the analysis process.

7. Acknowledgments

Our heartfelt thanks go to our colleagues Michael Jackson, Bashar Nuseibeh and Robin Laney for their patient listening and insightful suggestions. Thanks are also due to Andrés Silva, of the University of Madrid, for his comments on Section 3.2, and to Anthony Hall whose questions make us continue to think. We thank the anonymous referees for their comments, which have helped to improve the paper greatly. We also acknowledge the kind support of our colleagues in the Department of Computing, the Open University.

References

[1] C.A. Gunter, E.L. Gunter, M. Jackson, P. Zave "A reference model for requirements and specifications", IEEE Software, 17(3):37-43, 2000.

[2] J.G. Hall, L. Rapanotti, Towards a semantics of Problem Frames, The Open University, Department of Computing, Research Report No. 2003/5, 2003.

[3] C. A. R. Hoare, Communicating Sequential Processes. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[4] M. Jackson, Software Requirements & Specifications: a Lexicon of Practice, Principles, and Prejudices, Addison-Wesley, 1995.

[5] M. Jackson, Problem Frames, ACM Press Books, Addison Wesley, 2001.

[6] M. Jackson, P. Zave, Deriving Specifications from Requirements: an example, Proc. 17th Int. Conf. on Sofware Engineering, IEEE Computer Society Press, Los Alsamos, California, 1995, pp. 15-24.

[7] Y. Kesten, Z. Manna, A. Pnueli. Verifying Clocked Transition Systems. In Hybrid Systems III, LNCS vol. 1066, pp. 13-40, Springer-Verlag, 1996.

[8] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[9] D.L.Parnas, J. Madey. Functional Documentation for Computer Systems. Science of Computer Programming, Vol. 25, No. 1, Oct, 1995, pp. 41-61.

[10] P. Zave, M. Jackson, Four dark corners of Requirements Engineering, ACM Transactions on Software Engineering and and methodology, Vol. 6, No. 1, Jan. 1997, pp. 1-30.