

Technical Report No: 2003/05

Towards a semantics of Problem Frames

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>



Towards a semantics of Problem Frames

Jon G. Hall Lucia Rapanotti
The Open University
Milton Keynes
MK7 6AA
{J.G.Hall,L.Rapanotti}@open.ac.uk

Abstract

This paper presents a framework for understanding Problem Frames [5, 6], their components, their frame concerns, and the flexibility that is inherent in their definition.

Problem Frames classify software development problems. Of particular utility is their intuitive graphical notation that facilitates communication between software designer and problem owner. This provides costs as well as benefits and, as is the case with many such graphical notations, a (formal) semantics is needed to underpin meaning. To the best of our knowledge a semantics of Problem Frames is missing from the literature. In this paper we begin the definition of such a semantics.

Our semantics places Problem Frames within the framework for Requirements Engineering of Zave and Jackson and its subsequent formalization in the Reference Model of Gunter et al.

1 Introduction

Problem frames [5, 6] classify software development problems. They structure the analysis of the world in which the problem is located — the problem domain — and describe what is there and what effects one would like a system located therein to achieve. With its emphasis on problems rather than solutions, the problem frame approach leverages the understanding of a problem class, to allow the problem owner with their specific domain knowledge to drive the Requirements Engineering process. Of particular utility is the intuitive graphical notation that facilitates communication between software designer and problem owner.

As with other graphical notations, their intuitive nature provides costs as well as benefits. These include: the difficulty of precise interpretation of artifacts and

being unable to determine correctness and completeness of a description. Other graphical notation communities have provided a formal semantics to underpin meaning to counter some of these costs. However, to the best of our knowledge a semantics of Problem Frames is missing from the literature. And yet, given the above, the benefits that could accrue would appear to make one such valuable. In this paper we begin the definition of a semantics of Problem Frames that we intend to fulfill this role. As Problem Frames are a graphical notation, we also introduce a Problem Frame Description Language (*PFDL*) over which our semantics is defined.

Our semantics places Problem Frames within the framework for Requirements Engineering of Zave and Jackson [10] and its subsequent formalization in the Reference Model for requirements and specifications given by Gunter *et al.* [2]. In particular, we intend with our semantics to clarify the relationship between a Problem Frame-based process for requirements engineering and the completion of the requirements engineering phase (within a particular software development project) as defined in [10].

The paper is organized as follows. Section 2 outlines the Problem Frames framework. A comprehensive introduction to PFs is beyond the scope of this paper and can be found in [6]. Section 3 introduces the Problem Frame Description Language (*PFDL*). Section 4 recalls the Reference Model of [2]. Section 5 introduces the semantics. Finally, Section 6 concludes the paper.

2 The Problem Frames Framework

In this section we review some of the basic elements of the Problem Frames (PFs) framework. To focus our review we will describe PFs development as might be applied to the following problem, illustrated in Figure 1.

A computer system is required to control the catalyst unit and cooling system of a chemical reactor. An operator issues commands for activating or deactivating the catalyst unit; based on such commands, the system instructs the unit accordingly and regulates the flow of cooling water. Attached to the system is a gearbox: whenever the oil level in the gearbox is low, the system should ring a bell and halt execution.

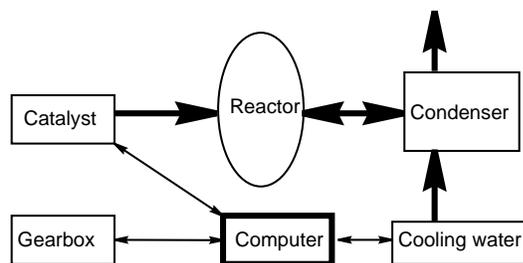


Figure 1. The Chemical Reactor schematic (from [1])

This problem is a simplified version of a chemical reactor described in [1, 7].

2.1 Problem Diagrams

Within the PFs framework, a *problem diagram* defines the shape of a problem by capturing the characteristics and interconnections of the parts of the world it is concerned with, and allowing the expression of concerns and difficulties that are likely to arise in discovering its solution. A problem diagram for our chemical reactor problems is shown in Figure 2.

The components are:

- An *Operation Machine*: the *machine domain*, i.e., the software system and its underlying hardware, to be built.
- Other boxes (*Cooling System, Catalyst, etc.*): *given domains* representing parts of the world that are relevant to the problem.
- A dotted oval *Control*: the *requirement*, i.e., what has to be true of the world for the machine to be a solution to the problem.
- The connections between the various elements (indicating *shared phenomena*), including events, operations and state information. In Figure 2,

for example, the connection between the *Operation Machine* and the *Cooling System* is annotated by the set d containing the two phenomena $increase_water_{act}$ and $decrease_water_{act}$, both controlled by the *Operation Machine*. Control is indicated by an abbreviation of the domain name followed by $!$, in this case $OM!$.

2.2 Problem Frames

One of the aims of the PFs framework is to identify basic classes of problem that recur throughout software development. Each such class should be captured by a *problem frame* that provides a template for the problem class. The template included: the topology of the problem (represented as a problem diagram); the characteristics of the domains involved (as annotations to the diagram); a way of building and discharging a correctness argument for the problem class. [6] identifies five *basic* problem frames, including the *commanded behaviour*, *information display* and *required behaviour* frames that we use here for illustration.

The topology of *commanded behaviour* frame is shown as a problem diagram in Figure 3.

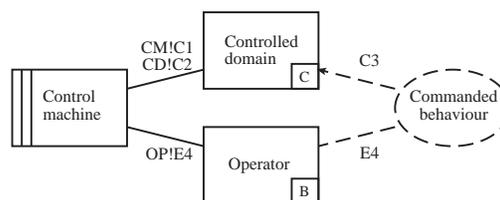
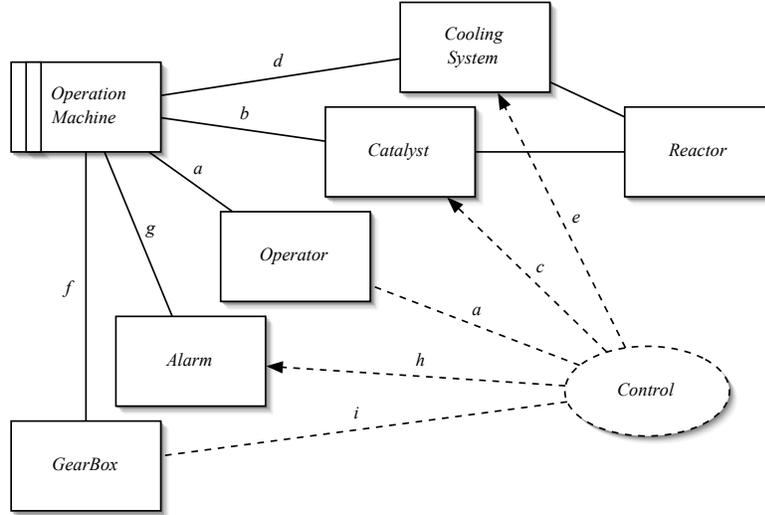


Figure 3. The Commanded Behaviour frame

Typically:

- the *commanded behaviour* frame is used when there is some part of the physical world whose behaviour is to be controlled in by some operator. The class of problem represented is by the *commanded behaviour* frame is then to build a machine that will accept the operator's commands and impose control accordingly;
- the *required behaviour* frame is used when there is some part of the physical world to be controlled in accordance with requirements;
- the *information display* frame is used when the machine needs to obtain some information (e.g., state or behaviour) about some part of the world and present it at the required place and in the required format.



$a : OP!\{open_catalyst, close_catalyst\}$ $e : CA!water_level$
 $b : OM!\{open_catalyst_{act}, close_catalyst_{act}\}$ $f : GB!request_service$
 $CA!\{is_open_{sen}, is_closed_{sen}\}$ $g : OM!ring_bell$
 $c : CA!\{open, closed\}$ $h : AL!bell_ringing$
 $d : OM!\{increase_water_{act}, decrease_water_{act}\}$ $i : GB!oil_level$
 $CS!\{is_rising_{sen}, is_falling_{sen}\}$

Figure 2. The Chemical Reactor Problem Diagram

A problem frame (whether basic or not) is represented by an *annotated* problem diagram with standard topology, and standard domain and phenomena names and markings. For the commanded behaviour frame, for instance:

- The *Controlled domain* is a *causal domain* (the C annotation in the figure), its phenomena being physical and causally related (indicated by C on the arcs).
- The *Operator* issues commands as *event* phenomena (the E annotation), shared with the *Control machine* and controlled by the operator (prefix *OP!*). The operator is assumed to be *biddable*, i.e., generate the events in E4 spontaneously (the B annotation).
- The requirement of the commanded behaviour frame, *Commanded behaviour*, determines the response of the Controlled domain to the Operator's commands, E4. The required behaviour is expressed in terms of phenomena C3 (typically distinct from both C1 and C2).
- In the links between the requirement and the domains, a dotted line indicates that its phenomena

are *referenced* by (i.e., an object of) the requirement, while a dotted arrow indicates that the phenomena are *required* (i.e., a subject for the requirement).

The *frame concern* identifies the form of the correctness argument that must be made (see [6]). For the commanded behaviour frame, the form of the argument will exploit explicitly stated causal properties of the controlled domain in showing that the machine behaviour in terms of the phenomena C1 and C2 will cause the required behaviour of the controlled domain in terms of the phenomena C3.

2.3 Problem Decomposition

Most real problems are too complex to fit basic problem frames. They require, rather, the structuring the problem as a collection of (interacting) sub-problems, each of which is smaller and simpler than the original.

Within the PFs framework, we would identify sub-problems from their problem frame templates and instantiate the corresponding sub-problem diagrams. Our semantics provides for the expression of the relationship between problems and sub-problems, and so here we briefly illustrate this process for the chemical

reactor.

The chemical reactor problem can be decomposed into three distinct sub-problems:

- a commanded behaviour frame allowing the operator to control the catalyst;
- a required behaviour frame regulating the water flow for cooling; and
- an information display frame for issuing a warning (and halting the system) when there is an oil leak in the gearbox.

To be able to apply the problem frames as templates, domains, and domain and phenomena markings of the problem must match those of the frame. For the commanded behaviour frame the instantiation is given in Figure 4; that for the required behaviour frame in Figure 5; that for the information display frame in Figure 6.

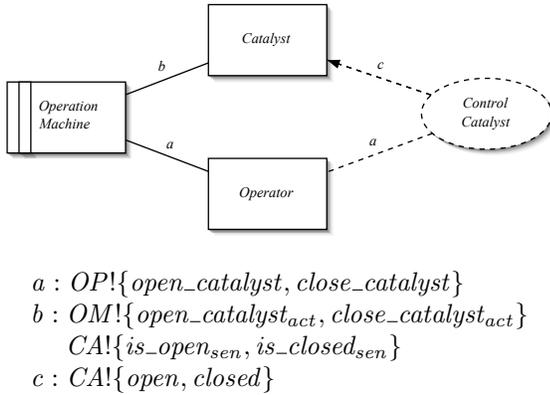


Figure 4. The Control Catalyst sub-problem, an instantiated commanded behaviour frame

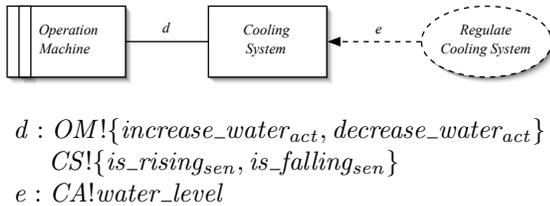


Figure 5. The Regulate Cooling System sub-problem, an instantiated required behaviour frame

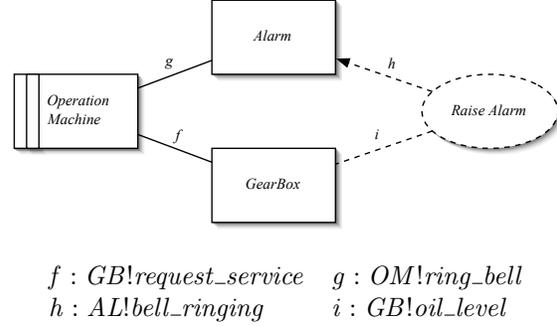


Figure 6. The Raise Alarm sub-problem, an instantiated information display frame

3 PFDL: a Description Language for Problem Frames

In this section we introduce a textual notation for problem frames, the *Problem Frame Description Language* (or *PFDL*). The purpose of this notation is to provide a (textual) syntactic domain on which to define our semantic function. Note that, for brevity of this presentation, only the basic elements of the PFs framework are captured in the *PFDL*.

We will assume that some language (or a collection of languages) has been chosen for the description of domains, phenomena, requirements and specifications. This we refer to as the *Domain and Requirement Description Language* (or *DRDL*, for short). Our semantics assumes of *DRDL* the absolute minimum. For instance, we assume that *DRDL* allows the representations of phenomena, can distinguish controlling influence over them from observation of their values, and can mark phenomena and domain appropriately. We also assume that *DRDL* comes equipped with some notion of reasoning, which we will refer to as \vdash_{DRDL} (or \vdash when context makes this clear). Note that we do not require the reasoning to be formal in any proof theoretic sense. It may, for instance, be that the reasoning takes a software engineering form, such as testing; other forms of reasoning. We will return to this point in Section 5.

3.1 Problem Diagrams

The basic components of problem diagrams are domains (including machine domains), requirements and their connectivity through arcs and annotations. To be able to express this in *PFDL*, we assume the following

given sets¹:

[PFDomain], [PFMachine],
[PFRequirement], [PFPhenomena]

For the grounding of a problem, both domains and requirements are named and described in *DRDL*:

PFDomainName
 ::= PFDomain \rightarrow *DRDL*
PFDomainDescription
 ::= PFDomain \rightarrow *DRDL*

PFRequirementName
 ::= PFRequirement \rightarrow *DRDL*
PFRequirementDescription
 ::= PFRequirement \rightarrow *DRDL*

Phenomena are also named and described in the *DRDL*:

PFPhenomenaDesignation
 ::= PFPhenomena \rightarrow *DRDL*

Phenomena are partitioned in sets of phenomena that can be either controlled or observed by each domain and the machine:

PFControlled
 ::= PFDomain \cup PFMachine \rightarrow PFPhenomena
PFObserved
 ::= PFDomain \cup PFMachine \rightarrow PFPhenomena

Similarly, some phenomena can be either required or referenced by the requirement:

PFRequired
 ::= PFRequirement \rightarrow PFPhenomena
PFReferenced
 ::= PFRequirement \rightarrow PFPhenomena

We will bundle all annotating functions into a *problem diagram annotation*:

PFAnnotation ::=
(PFDomainName, PFDomainDescription, ...
PFRequired, PFReferenced)

A problem diagram is a set of domains, a single machine, a single (set of) requirement(s), and an annotation, i.e.:

PFProblemDiagram ::=
(\mathbb{P} PFDomain, PFMachine,
PFRequirement, PFAnnotation)

For a concrete example of a *PFDL* description, the *PFDL* description of the chemical reactor problem diagram is shown in Figure 7.

¹The notion of a given set is as in the formal notation Z [9].

3.2 Problem Frames

As already mentioned, basic problem frames are templates for classes of problems. [6] provides a template language enriched over that for problem diagrams by the introduction of domain and phenomena markings. These markings must be matched by problem diagram elements for the problem frame template to apply.

Each domain can be marked as either biddable, lexical or causal (which are distinct):

PFDomainMarking
 ::= PFDomain \rightarrow {B, X, C}

Each (set of) phenomena can be marked as either causal, symbolic or event (also distinct):

PFPhenomenaMarking
 ::= PFPhenomena \rightarrow {C, Y, E}

The diagram of a problem frame is called a frame diagram and is a problem diagram augmented with marking for domains and phenomena:

PFFrameDiagram ::=
(PFProblemDiagram,
PFDomainMarking, PFPhenomenaMarking)

A Problem Frame provides a *frame concern*, i.e., a template for correctness argument which is common to all problem of that class. We assume that frame concerns are taken from another given set:

[PFFrameConcern]

and so may define:

PFProblemFrame ::=
(PFFrameDiagram, PFFrameConcern)

A *PFDL* representation of commanded behaviour frame of Section 2.2 is given in Figure 8. The reader will note that the phenomena controlled by the operator are event phenomena (marking E), while all other phenomena are causal (marking C); such markings will define the applicability of the problem frame to a problem diagram.

4 The Reference Model

The Reference Model [10, 2, 4] 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

	Name	Controlled/Required	Observed/Referenced
<i>M</i>	<i>Operation Machine</i>	<i>open_catalyst_{act}, close_catalyst_{act}</i> <i>increase_water_{act}, decrease_water_{act}</i> <i>ring_bell</i>	<i>is_open_{sen}, is_closed_{sen}</i> <i>is_raising_{sen}, is_falling_{sen}</i> <i>request_service</i> <i>open_catalyst, close_catalyst</i>
<i>D₁</i>	<i>Catalyst</i>	<i>is_open_{sen}, is_closed_{sen}</i> <i>open, closed</i>	<i>open_catalyst_{act}, close_catalyst_{act}</i>
<i>D₂</i>	<i>Cooling System</i>	<i>is_rising_{sen}, is_falling_{sen}</i> <i>water_level</i>	<i>increase_water_{act}, decrease_water_{act}</i>
<i>D₃</i>	<i>Reactor</i>	\emptyset	\emptyset
<i>D₄</i>	<i>Operator</i>	<i>open_catalyst, close_catalyst</i>	\emptyset
<i>D₅</i>	<i>GearBox</i>	<i>request_service</i> <i>oil_level</i>	\emptyset
<i>D₆</i>	<i>Alarm</i>	<i>bell_ringing</i>	<i>ring_bell</i>
<i>R</i>	<i>Control</i>	<i>open, closed</i> <i>water_level</i> <i>bell_ringing</i>	<i>open_catalyst, close_catalyst</i> <i>oil_level</i>

Figure 7. The Chemical Reactor problem diagram in tabular form

	Name	Marking for Domain	Marking for Controlled/Required Phenomena	Marking for Observed/Referenced Phenomena
<i>M</i>	<i>Control Machine</i>	--	<i>C1</i>	<i>C2, E4</i>
<i>D₁</i>	<i>Controlled Domain</i>	<i>C</i>	<i>C2, C3</i>	<i>C1</i>
<i>D₂</i>	<i>Operator</i>	<i>B</i>	<i>E4</i>	<i>C1</i>
<i>R</i>	<i>Commanded Behaviour</i>	--	<i>C3</i>	<i>E4</i>

Figure 8. PFDL description of the commanded behaviour frame

the environment, the system and their interface. From a requirement engineering’s viewpoint, the key artifacts are the following²: the domain knowledge, K , is what we know about the environment; the requirement, R , is what the customer requires of a system working within the environment; and the specification, S , is a description of the system, which can be used for its implementation.

In the Reference Model, a *designation* is used to ground terms in the real world[10]. The designation provide names to describe the environment, the system and their artifacts. More precisely, the vocabulary is used to name phenomena: typically, states or events. Importantly, phenomena are grouped into those which belong and are controlled by the environment, denoted by e , and those which belong and are controlled by the system, denoted by s . The sets e and s are further partitioned into phenomena which are hidden and

visible:

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

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

where phenomena in e_h are hidden from the machine, and those in e_v are visible to it. *Vice versa* for s_h and s_v , with respect to the environment.

Note that the RM assumes that the specification, S , lies in the environment’s and the system’s common vocabulary. 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 $e \cup s_v$.

5 Interpreting Problem Frames within the Reference Model

As in refinement [8], in which *pre* and *post* conditions define a ‘challenge’ to develop code to implement them, so a problem diagram, through its environment

²Other artifacts are defined in the model, which pertain to design and implementation. These are beyond the scope of this paper, hence are omitted here.

and requirements, defines a ‘challenge’ to develop a machine specification to discharge them. Whereas refinement is mature, and has formal notation up to the job of representing a challenge³ we, as yet, do not. We will therefore be forced to improvise, notationally, somewhat.

As the basis for our semantics, we use the following definition. Given a world description K , a requirement descriptions R , and sets of shared phenomena c (controlled by the machine domain) and o (observed by the machine domain) over some $DRDL$, we define a ‘challenge to find a satisfying specification’ as

$$c, o : [K, R] = \{S \mid S \text{ controls } c \\ \wedge S \text{ observes } o \\ \wedge K, S \vdash_{DRDL} R\}$$

As such, $c, o : [K, R]$ stands for the set of all specifications that control phenomena in the set c , observe phenomena in the set o , and which, in the context of K are capable of discharging R , as per [10, 2, 4]. As indicated in the notation, the details of \vdash_{DRDL} will be provided by the $DRDL$. We also note that $c, o : [K, R]$ can be empty (as when R is *false*, for instance), in which case there is no satisfying specification, i.e., the system cannot be built. Referring to the original Reference Model, $c = s_v$, i.e., the machine controlled phenomena shared with the environment, and $o = e_v$, the machine visible shared phenomena.

Within the PFs framework, the environment is described through a set of domains. Also, there is a separation between the concept of domain and its description. Therefore, in the semantics, for convenience, we will identify

$$c, o : [\{D_1, \dots, D_n\}, R]$$

and

$$c, o : [DD_1 \wedge \dots \wedge DD_n, DR]$$

where $DD_i = \text{PFDomainDescription}(D_i)$ and $DR = \text{PFRequirementDescription}(R)$.

5.1 Problem Diagrams as challenges

Given a problem diagram $PD = \langle \{D_1, \dots, D_n\}, M, R, A \rangle$ expressed in $PFDL$ over a $DRDL$, we define its semantics as:

$$c, o : [\{D_1, \dots, D_n\}, R]$$

³The reader may be familiar with the refinement notation $w : [pre, post]$ which stands for (all) code that, by changing only variables in w will take as input state satisfying pre and produce state satisfying $post$.

where $c = \text{PFControlled}(M)$ and $o = \text{PFObserved}(M)$.

The reader will note that, discharging the correctness argument associated with the diagram is part of the challenge, specifically that $K, S \vdash_{DRDL} R$, and that S controls c and S observes o .

As an example, assuming the annotation given in Figure 7, the semantics of the chemical reactor problem diagram is the following challenge:

$$\{open_catalyst_{act}, close_catalyst_{act}, \\ increase_water_{act}, decrease_water_{act}, ring_bell\}, \\ \{is_open_{sen}, is_closed_{sen}, is_raising_{sen}, \\ is_falling_{sen}, request_service, \\ open_catalyst, close_catalyst\} \\ : [\{Catalyst, CoolingSystem, Operator, \\ Reactor, GearBox\}, Control]$$

5.2 Problem Frames Semantics

A Problem Frame is a template for a class of problems. By ‘applying’ a problem frame to a problem diagram, we can derive a (sub-)problem diagram whose semantics is a challenge. This allows for a process of problem decomposition in which the sub-problem diagrams are projections of the original problem diagram through a collection of problem frames.

Semantically, we can see the application of a problem frame to a problem diagram as matching topologies, and domain and arc markings. We say that a frame diagram $PF = \langle \langle D, M, R, A \rangle, m \rangle$ matches a problem diagram $PD = \langle D', M', R', A' \rangle$ when there is an injective correspondence $\iota : D \rightarrow D'$ which respects the topology, domain and arc markings.

An problem frame instantiation is then the application of *iota* to $\langle D, M, R, A \rangle$.

5.3 Working with the semantics

Extending the analogy with refinement, working towards an $S \in c, o : [K, R]$ will be an exploration of the design choices at each stage: because we work within all of software—including architectures, *etc*—and not just code, the design space is possibly much larger than that in refinement.

Problem frames, as templates, allow the navigation around the design space through dividing up the problem.

Let us illustrate this idea with an example. From the Chemical Reactor above, we have three sub-problem diagrams corresponding to the required behaviour, commanded behaviour, and information display frames. The decomposition of the Chemical Reactor problem into three sub-problems is illustrated in

$$\begin{array}{l}
c_1, o_1 : [\{Catalyst, Operator\}, Control\ Catalyst] \\
c_2, o_2 : [\{Cooling\ System\}, Regulate\ Cooling\ System] \\
c_3, o_3 : [\{Alarm, GearBox\}, Raise\ Alarm] \\
\hline
c, o : [\{Catalyst, Cooling\ System, Operator, Alarm, Reactor, GearBox\}, Control] \quad \uparrow \text{Decomposition, } \downarrow \text{Recomposition}
\end{array}$$

Figure 9. The decomposition/recomposition relationship between the Chemical reactor problem diagram and its sub-problem diagrams

Figure 9, in which:

$$\begin{array}{l}
c_1 = \{open_catalyst_{act}, close_catalyst_{act}\} \\
o_1 = \{is_open_{sen}, is_closed_{sen}, \\
\quad open_catalyst, close_catalyst\} \\
c_2 = \{increase_water_{act}, decrease_water_{act}\} \\
o_2 = \{is_raising_{sen}, is_falling_{sen}\} \\
c_3 = \{ring_bell\} \\
o_3 = \{request_service\} \\
c = c_1 \cup c_2 \cup c_3 \\
o = o_1 \cup o_2 \cup o_3
\end{array}$$

As for refinement, given a possible decomposition through the application of problem frames, there will be a corresponding composition concern (analogous to refinement’s verification condition) that should be able to convince us of the validity of the recomposition. Our point is that backwards argument identifies the sub-problems that should concern us; the forwards argument requires that composition concerns.

In this chemical reactor example for instance, there is a requirement that when the oil level in the gearbox is low, the machine should sound an alarm and halt (the information display sub-problem). On the other hand, the machine is always required to regulate the cooling system (the required behaviour sub-problem). The composition of these two sub-problem needs to continue to guarantee the safe functioning of the reactor. This concern therefore must be addressed at the composition level.

6 Discussion and Future Work

In this paper we have suggested a semantics for elements of the PFs framework. We have introduced the Problem Frame Description Language (PFDL), over a Domain and Requirements Description Language (DRDL), for the description of problem diagrams and problem frames. We have defined the meaning of a problem frame as a template for problem diagrams, and a semantics for problem diagrams within the Reference

Model, given in terms of *challenges*, i.e., $c, o : [K, R]$, a notion that we have also introduced.

It is worth stressing that our semantics is based on a weak notion of correctness. This differs from that of the Reference Model, which is based on universal quantification over all ‘phenomena’ sequences’. The actual wording used by the authors is that:

A requirement with an environment constraint should always be verified by a demonstration or proof that specified properties, conjoined with domain knowledge, guarantee the satisfaction of the requirement.

In our semantics we relax that notion of proof to its weakest (non-technical) meaning:

facts, evidence, argument, etc. establishing or helping to establish a fact.

We might, with this definition, also admit proofs of the type ‘there is no example that *a system* did not satisfy the requirements’ (a testing ‘proof’); another is the ‘customers were convinced by the arguments of the developers’; another is ‘a formal refinement of the requirements to code was performed, with all verification conditions being discharged’; another is ‘Don Knuth wrote it’; yet another is ‘we’ll fix it in the next release’. Clearly, the nature of \vdash_{DRDL} and that of the requirements are closely related; in the case when the requirements include safety (as would be the case in a safety critical system, for instance), the strength of \vdash_{DRDL} must provide for their discharge.

The development of the $c, o : [K, R]$ needs many things. Most importantly, the availability of a ‘weakest environment predicate transformer’ (analogous to the weakest precondition semantics of refinement) would place the semantic basis of the manipulation of such expressions on a much firmer footing, and even lead us towards a requirements engineering notion of ‘refinement’ that forms a lattice over such objects; given a

problem, the refinement relation \sqsubseteq would justify the relationship, as summarised in Figure 10:

$$problem \sqsubseteq partial_soln_0 \sqsubseteq \dots \sqsubseteq specification$$

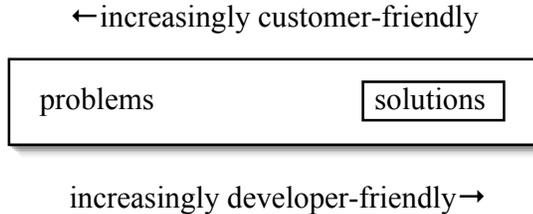


Figure 10. The move between problem and solution domains (after [8, Figure 1.6, page 9])

The import of this paper is to provide a formal semantics for elements of the PFs framework. The extent of the uses to which it can be put is still unexplored and provides a focus for further research. However, some of the immediate benefits of it are already apparent. As for other formal semantics, it can act as an arbiter of communication between two parties and could be used as the basis of tool support for PFs. To be sure, many of the engineering notions that are present in computer science are based on the definition of a mathematical basis provided by a formal semantics.

7 Acknowledgements

Our heartfelt thank you goes to Michael Jackson, Bashar Nuseibeh and Robin Laney for their patient listening and insightful suggestions. We also acknowledge the kind support of our colleagues in the Department of Computing, the Open University.

References

- [1] O. Dieste, A. Silva, Requirements: Closing the Gap between Domain and Computing Knowledge, Proceedings of SCI2000, Vol. II (Information Systems Development), 2000.
- [2] C.A. Gunter, E.L. Gunter, M. Jackson, P. Zave “A reference model for requirements and specifications”, IEEE Software, 17(3):37-43, 2000.
- [3] J.G. Hall, M. Jackson, R.C. Laney, B. Nuseibeh, L. Rapanotti, Relating Software Requirements and Architectures using Problem Frames, IEEE Proceedings of RE’02, Essen, Germany, 2002.
- [4] J.G. Hall, L. Rapanotti, A reference Model for Requirements *Engineering*, Submitted to RE’03, 2003.
- [5] M. Jackson, Software Requirements & Specifications: a Lexicon of Practice, Principles, and Prejudices, Addison-Wesley, 1995.
- [6] M. Jackson, Problem Frames, ACM Press Books, Addison Wesley, 2001.
- [7] N. Leveson, Software Safety: why, what and how, ACM Computing Survey, 18(2):125-163, 1986.
- [8] , C. Morgan, Programming from Specifications. Prentice-Hall International, 1990.
- [9] J.M. Spivey, The Z Notation. Prentice-Hall International, 1988.
- [10] P. Zave, M. Jackson, Four Dark Corners of Requirements Engineering, ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 1, January 1997, pp.1-30.