ISSN 1744-1986

Technical Report N° 2004/26

A Constructive Approach to Problem Frame Semantics

Zhi Li
Jon G. Hall
Lucia Rapanotti

10th December 2004

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 Constructive Approach to Problem Frame Semantics

Zhi Li, Jon G. Hall, Lucia Rapanotti*

3rd December 2004

Abstract

The Problem Frame approach (PF) is an effective requirements engineering tool for analysing and structuring software development problems. PF has a graphical notation that is easy to use and understand. PF imposes only loose constraints on the features of the language used to describe its components and, with only slight restrictions, Hoare's CSP can be used. Although not generally applicable, in this paper we show that CSP can provide a constructive way to arrive at a solution to a PF expressed problem.

The work is situated in the Problem Frame Semantics for Software Development of Hall, Rapanotti and Jackson.

1 Introduction

The formation of Software Engineering (SE) was led by the so-called "software crisis" in 1968, when requirements analysis and definition were reviewed as a potentially high-leverage but neglected area. By the mid-1970s, the review by Bell et al. [BT76] had produced plenty of data, confirming that "the rumoured 'requirements problem' are a reality". The recognition of the critical nature of requirements has established Requirements Engineering (RE) as an important sub-field of Software Engineering [GMB94].

Understanding the problem before providing the solution is one of the most important approaches that support incorporating the requirements engineering process within the software engineering process. Jackson and Zave's work on *Domain Descriptions* [JZ93] and *Four Dark Corners of Requirements Engineering* [ZJ97] have given reasons why understanding the problem world is fundamental to RE.

The Problem Frame approach (PF) provides a useful way for people to understand and solve software problems [Jac01]. They are useful techniques throughout the entire software process, because they can relate software requirements and architectures in an iterative twin-peak fashion [Nus01, HJL+02]. The benefits of using PFs in RE are more than just understand the problem — they can help relate RE activities and software design activities.

*{Z.Li, J.G.Hall, L.Rapanotti}@open.ac.uk, Computing Department, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK

The motivation of developing a semantics for Problem Frames is that a syntax (graphical notations) without an unambiguous semantics can cause difficulties in communication [Pet95]. A formal semantics can help with rigourous reasoning and avoid ambiguities, thus providing rules for diagrammatic transformation and promising tool support for Problem Frames. A formal semantics for PF has been suggested by Hall *et al*. [HRJ04]. The framework provides a set-based characterisation of the solutions that satisfy a particular problem, but does not give constructive methods for finding a solution.

In this paper we show that the combination of PF and the quotient operator $\backslash\backslash$ of [LS95] provides a constructive method of determining a solution to a problem expressed in PF notation. In addition, we show how different languages can work together in PF, and how one can move between the problem frame notation and the description of domains. We also provide clarification and examples of the semantics in action.

This paper is only part of our long-term research of building a solid and robust foundation of PF to underpin their graphical notation.

This paper is organised as follows. Section 2 gives a brief account of related work. Section 3 gives an introduction to the formal semantics of Problem Frames, to the weakest-environment Calculus for Communicating Processes and their relationship to each other. Section 4 illustrates their linkage with an example of software development using Problem Frames. Section 5 evaluates the results and provides pointers to new developments.

2 Related work

PFs have existed for over 10 years now, but there have been very few attempts to formalise their graphical notation. Some early work of [BKNS97] characterises the properties of Jackson's early frames (the Translation frame, Control frame, Information Systems frame, Workpiece frame and Connection frame) in terms of typical types of domains and requirements, and for two of them, the design of the solution. This might be used to determine the solution form. [CR99] take this approach further, providing a formalisation of the Translation frame and Information System frame using CASL (the Common Algebraic Specification Language, [CoF99]. [Del02] proposes a semantic approach which uses UML as a means of describing a meta- model as a semantics for problem frames. The semantics describes context diagrams, problem diagrams and problem frames, but does not capture the meaning of other elements, such as correctness of a machine specification with respect to its environment and requirements.

In [LS97], the authors extend the notion of weakest environment to processes with state. This provides a generalisation of the work reported in [LS95]. As we do not need to consider state (or rather the notion of state is that of control state, not variable state) we do not need the more sophisticated calculus.

3 Problem Frames semantics and the weakest environment calculus

3.1 Problem Frames semantics

The Problem Frame approach (PF) is a framework for requirements engineering developed through Jackson *et al.*'s long-term research into software development problems. Problem frames [Jac95, Jac01] classify software development problems; they structure the world in which the problem is located (the problem domain) and describe what is there and what effects a system located there must achieve. By emphasising problems rather than solutions, Problem Frames can exploit the understanding of a problem class, allowing a problem owner with specific domain knowledge to drive the Requirements Engineering process.

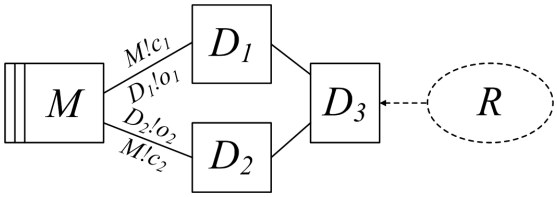

Figure 1: A problem diagram with multiple domains

Hall, Rapanotti and Jackson [HRJ04] provide a formal description of PF that allows a problem diagram—such as that in Figure 1—to be given a precise meaning. Given:

- a real-world description K (the collection of domains D_1, D_2, D_3 in the figure),
- a requirement description R,
- a set of shared phenomena c controlled by the machine ($c_1 \cup c_2$ in the figure), and
- a set of shared phenomena o observed by the machine ($o_1 \cup o_2$ in the figure).

the meaning of a problem diagram is a "challenge to find a satisfying specification" [HRJ04] for the machine M, i.e., to find a member of the set $c, o : [K,\ R]$ defined as

$$\{S : \textit{Specification} \mid \quad S \textit{ controls } c \wedge\ S \textit{ observes } o \wedge\ K, S \vdash_{DRDL} R\}$$

Hall *et al.*'s semantics is propositional in nature: as in the propositional calculus, atomic propositions can be described in any (or even multiple) suitable language(s); for instance, it can accommodate a natural language description of an *operator* such as 'The operator smokes' as well as a first order logical description of a variable update such as $x = 0 \wedge x' = 1$. This matches the multi-paradigm assumption of problem frames [JZ93][Domain Descriptions]. The language (or collection of languages) for the description of domains and requirements is named *DRDL* (for Domain and Requirements Description Languages). Any notion of correctness, such as the $\vdash_{DRDL}$ in

the definition of a challenge, is made with respect to (more or less formal) correctness notions in *DRDL*. In this paper, we choose *DRDL* to consist of two semantically related languages[1] that appear in Lai's weakest environment calculus [LS97]. Specifically, they are Hoare's CSP and *Spec*, a propositional logic-based language for the description of specifications (these are described later).

The notation for a challenge is intentionally reminiscent of the notation for formal refinement of [Mor94], but the reader will note that any non-set based notion of a refinement relation is currently missing from the semantics of [HRJ04]. This leads us to look for constructive methods to complete a problem diagram's challenge; part of the contribution of this paper is to look to CSP and *Spec* as possible description languages for PF, and to use the tools of CSP to solve a challenge constructively.

3.2 Weakest-environment calculus

Lai *et al*'s work in the weakest-environment calculus [LS95] extends to CSP parallel composition the notion of refinement. I.e., given a specification S and a CSP process term P, Lai's work allows us to solve for X in

$$P \| X \ sat \ S \tag{1}$$

Indeed, Lai *et al* give us more, defining the weakest environment to be the specification $P \setminus\setminus S$, named the *quotient of P by S*, such that $P \| (P \setminus\setminus S)$ *sat* S. $P \setminus\setminus S$ is actually the weakest such specification, i.e., any Q *sat* $(P \setminus\setminus S)$ will be a solution of (1).

In this paper, *Spec*, defined below, is the description language we shall choose to represent requirements; CSP process terms and *Spec* will be used to describe domains.

Note that, although (1) is always satisfied by the quotient, it is not always the case that $P \setminus\setminus S$ defines a process, in the CSP sense. [LS97] provides some examples. However, Lai *et al*'s construct's importance to us is that it provides a (in some sense) canonical solution to a challenge, at least when domains are described in the CSP family of notations.

3.3 CSP as a DRDL

As already mentioned, we use CSP and *Spec* to describe domains, and *Spec* to describe requirements. That a specification language such as *Spec* can describe requirements is not anathema: in Jacksonian terms, a specification is simply a requirement that speaks the language of the machine interface; from [GGJZ00] these are the sets e_v and s_v; in our language they are, respectively, the sets o and c.

A specification is a predicate with free variables *tr* (representing traces) and *ref* (representing refusals) and a notion of a communication alphabet. A simple example of a specification is $Spec_1$ defined as the predicate

$$tr = \langle\rangle \wedge ref \subseteq A$$

[1] The relationship between correctness arguments when semantically unrelated domain description languages are used is not considered in this paper.

with alphabet $\alpha Spec_1 = A$ (we will see other examples later in this paper). As stated in [LS95], implication provides a complete partial order on specifications, and satisfaction can be defined between specifications so

$$Sp \; sat \; Sq \quad \text{if and only if} \quad Sp \Rightarrow Sq$$

That $stop_A$, a process that does nothing, satisfies $Spec_1$ is written $stop_A \; sat \; Spec_1$. The satisfaction relation *sat* is the notion of correctness we require for CSP, i.e., $\vdash_{DRDL} = sat$.

A specification P is a process if the following conditions on *tr* and *ref* are met [LS95]:

P1. $P(\langle\rangle, \{\,\})$

P2. $P(tr^\frown ur, \{\,\}) \Rightarrow P(tr, \{\,\})$

P3. $Y \subseteq X \wedge P(tr, X) \Rightarrow P(tr, Y)$

P4. $P(tr, X) \wedge \neg \; \exists v : val(e) \bullet P(tr^\frown \langle c.v\rangle, \{\,\}) \Rightarrow P(tr, X \cup \{c\})$

It is an important property in PF that, from a domain's description, one should be able to distinguish those visible phenomena that are controlled by the domain from those that are observed by it; this amounts to the property [ZJ97] that only a domain that controls a phenomenon should be able to change it. As pointed out in [ZJ97] full CSP does not have this property, but it is not difficult to check, of any collection of domain descriptions, whether it holds or not: for a CSP process[2] P define $P! = \{c \mid c!v \, appears \, in \, P\}$; a collection of CSP-described domains $P_1, ..., P_n$ are said to be *control determined* if and only if $P_i!$ are pairwise disjoint.

For such a collection of domain descriptions, we define $\iota : \bigcup_i \alpha P_i \mapsto \{P_1, ..., P_n\}$ to be the (partial) function that associates with a channel the process term that controls it. Given a problem diagram, if we include the machine M, the associated ι can be made total by adding $\iota_M(c) = M$ whenever $\iota(c)$ is undefined[3]. There is one corollary: the machine, as belonging to a problem diagram, must have assigned to it an alphabet. As part of the parallel composition of a problem diagram, the machine must also have an alphabet: clearly $\alpha M = \text{dom}(\iota_M) \cup observed$, where $observed \subseteq \text{dom}(\iota)$, i.e., those channels it must control together with those it may observe.

If domains in PF are described as CSP processes or specifications, then we need a CSP interpretation of their connection through the sharing of phenomena. The CSP parallel composition operator, $\|$, fits the bill nicely: indeed, after [ZJ93], the interpretation is quite natural: CSP parallel composition is, essentially, conjunction with channel phenomena — channel communications such as $c!5$ and $c?x$ — shared (and unhidden).

With these interpretations, we may, for a problem diagram consisting of CSP-described domains, C_1, ... , C_n, requirement R with a topology that allows the machine

[2] A similar definition holds for terms in the Specification language *Spec*.

[3] This just says that anything that isn't controlled in the environment of the machine must be controlled by the machine.

to control c and observe o, write the solution machine as any in the set

$$
\begin{array}{l}
c, o : [C_1 \| ... \| C_n, R] \\
\quad = \quad \{S : \mathit{Specification} \mid \\
\qquad\quad S \text{ controls } \mathit{dom}(\mathit{iota}_M) \wedge S \text{ observes } \mathit{observed} \wedge C_1 \| ... \| C_n \| S \textit{ sat } R\}
\end{array}
$$

3.4 Lai's quotient

Lai *et al* provide a closed predicate definition for the weakest environment of a process. Given a process P, requirement R and a communication alphabet for the desired solution A with process P, Lai defines $P \setminus\setminus R(tr, ref)$ with alphabet $\alpha R \setminus \alpha P \cup A$ as the specification:

$$
\begin{array}{rl}
P \setminus\setminus R(tr, ref) \triangleq & \forall\, ur : traces(R) \,\forall\, rep \subseteq \alpha P \\
& \quad (tr = ur \upharpoonright \alpha\,(P \setminus\setminus R) \\
& \quad \wedge\, P(ur \upharpoonright \alpha P, rep)) \\
& \quad \Rightarrow R(ur, rep \cup ref)
\end{array}
$$

This complex definition will be expanded (and explained) by the example that follows. Essentially, it says that a trace/refusal set pair of the quotient must contribute to the traces and refusals of the requirement (the *ur* and *rep*) in the right measures.

That $P \setminus\setminus R$ is a process[4] is not trivial; indeed, there are situations in which $P \setminus\setminus R$ is not a process[5]. $P \setminus\setminus R$ may not, therefore, be a CSP solution to the challenge, but a process that satisfies it will be. In any case we can write:

$$
\begin{array}{rl}
c, o : [K, R] \quad = \quad & \{S : \mathit{Spec} \\
& \quad \mid S! = \mathit{dom}(\mathit{iota}_M) \\
& \quad \wedge\, S? = \mathit{observed} \\
& \quad \wedge\, S \textit{ sat } K \setminus\setminus R\}
\end{array}
$$

The combination of Problem Frames with Lai's weakest environment provides us with our first example of a constructive determination of a solution machine. In the next section we present an example of the combination of PF with CSP, to show a) how the two notations work together, b) the deeper semantic relationship between the two operators, at the level of the correctness argument.

4 Linking the two frameworks

The example we present is not new, it appears in Lai *et al*'s paper as an example of the use of the quotient operator. We present the example as it would be done in PF, i.e., as a problem statement, the application of a problem frame (the Required Behaviour frame [Jac01]), the solution derived thereby through the application of the quotient operator, and the discharge of the correctness argument for the problem frame.

That the example is simple also means that the reader will not be overwhelmed by notation.

[4] I.e., that it satisfies conditions P1-P4.

[5] This is the technical difficulty alluded to above.

4.1 The problem statement

The problem is to build a machine to interact with a one-place buffer subsystem so that their composition is a two-place buffer. In PFs framework, a problem diagram for this problem might be that of Figure 2.

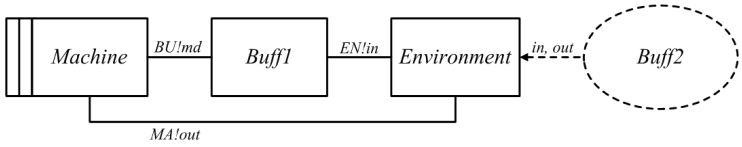

Figure 2: Problem diagram before transformation

The reader will note that the problem diagram contains an environment; Lai's paper does not consider an environment — CSP as a formalism, allows channel communications to 'dangle' in a way that PF cannot.

In terms of [HRJ04], given the development of Section 3, the problem diagram can be represented by the challenge

$$c, o : [K, R] = \{out\}, \{md\} : [Buff1 \parallel Environment, Buff2]$$

4.2 Domain descriptions

Informally, a one-place buffer $Buff1$ repeatedly inputs and faithfully outputs values of some fixed type, ensuring that output lags at most one value behind input. It thus has two external channels, *in* for input and *md* for output, both of that type, so that $\alpha Buff1 = in, md$.

Throughout this paper, we use the following notation for traces:

$< a1, \ \ldots, an >$ is the trace consisting of elements $a1, ..., an$

$\mid tr \mid$ is the length of a trace *tr*

$ur \frown tr$ the concatenation of traces *tr* and *ur*

$ur \leq tr$ indicates that *ur* is a prefix of *tr*

$ur \leq^n tr$ means that *ur* is a prefix of *tr* at most *n* elements shorter, and

$tr \upharpoonright A$ is the trace obtained by removing from *tr* all communications on channels in *A*; it is convenient to write $tr \upharpoonright \{c\}$ as *c*.

Formally, as a *Spec* it can be described as the specification

$$Buff1(tr, ref) \triangleq md \leqslant^1 in \wedge in \notin ref \triangleleft in = md \triangleright md \notin ref$$

Informally, a two-place buffer, $Buff2$, repeatedly inputs and faithfully outputs values of some fixed type, ensuring that output lags at most two values behind input.

Process $Buff2$ thus has two external channels, *in* for input and *out* for output, so that $\alpha Buff2 = in, out$. Formally, it can be described as a specification

$$Buff2(tr, ref) \triangleq (out \leqslant^2 in) \wedge (out \leqslant^1 in \Rightarrow in \notin ref) \wedge (out \neq in \Rightarrow out \notin ref)$$

Informally, the environment *Env* can both offer to input a value to $Buff1$ and retrieve output from the *Machine*. It can offer to participate in any of these two events in any sequence at any time. Formally, it can be described as having alphabet $A \supseteq in, out$ with:

$$Env(tr, ref) \; sat \; true$$

4.3 Problem transformation

From a CSP view, process $Buff1$ and *Environment* (or *Env*) can be merged into one process $Buff1 \parallel Env$, known as $Buff1 + Environment$. Therefore, its problem diagram after transformation:

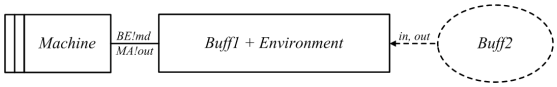

Figure 3: Problem diagram after transformation

Given that $Buff1$ and *Env* can be described in terms of traces and refusal sets, combining them through parallel composition (and hiding) gives the problem diagram shown in Figure 3. This problem diagram matches the Required Behaviour frame of [Jac01], as shown in Figure 4.

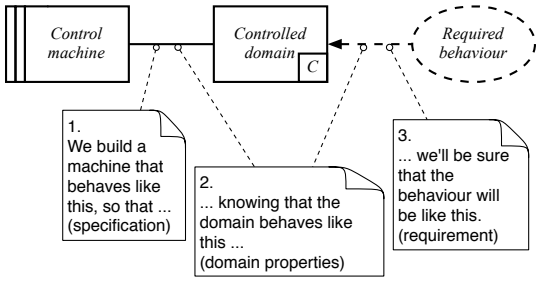

Figure 4: The Required Behaviour Frame with frame concern

4.4 Solving the problem diagram

Applying the quotient provides the following *Spec* as solution to the problem:

$$Machine \triangleq (Env \| Buff1) \setminus\setminus Buff2$$

which is (provably) satisfied by the following machine:

$$Machine(tr, ref) \triangleq (out \leqslant^1 md) \wedge (md \notin ref \lhd md = out \rhd out \notin ref)$$

4.5 Correctness argument of the required behaviour frame concern

As mentioned above the problem fits the Required Behaviour frame. The required behaviour frame has frame concern:

1. We will build the machine to behave like this...
2. ... so that, knowing that the $Buff1 + Env$ works like this ...
3. ... we will be sure that the requirement will be satisfied.

The following argument instantiates this correctness argument for the one-place buffer solution:

1. We will build the machine to behave like this... (it is a one-place buffer):

   a. when it is empty (obviously it cannot output any values yet), it can accept one value from $Buff1$.

   b. when it is full (it cannot accept any more value from $Buff1$), it can store the inputted value for some time (storage function of a buffer), and then output the value to *Env*.

   Substituting the machine's formal definition $Machine(tr, ref) \triangleq (out \leqslant^1 md) \wedge (md \notin ref \lhd md = out \rhd out \notin ref)$ we have:

2. ... so that, knowing that the $Buff1 + Env$ works like this:

   a. Firstly, *Env* retrieves one value from *Machine* through channel *out*, (if it can then the machine *Machine* must have contained a value before), otherwise:

   b. (because the machine is empty now — not ready to give out a value — which prevents the *Env* from retrieving any value) *Env* outputs one value *a* to $Buff1$ through channel *in*. Then either:

   b.i *Env* retrieves value *b* from *Machine* through channel *out* if it can; Then *Env* outputs value c to $Buff1$. Then *Env* goes back to step a.

   or

   b.ii $Buff1$ outputs the value *a* to *Machine* through channel *md*. Then $Buff1 + Env$ goes back to step a.

3. We will be sure that the requirement $Buff2$ (See Figure 5) is satisfied.

   a. First of all, if it is totally empty, it can allow one input; then its internal mechanism shall ensure that the value be transmitted to the second place closer to the output channel from the first place closer to the input channel.

   b. It can either

b.i output the value to channel *out*, and then go back to step a.

or

b.ii allow another input, then outputs it to channel *out*, and then its internal mechanism shall ensure that the value in the first place be moved to the second place through channel *md*. Then it goes back to step b.

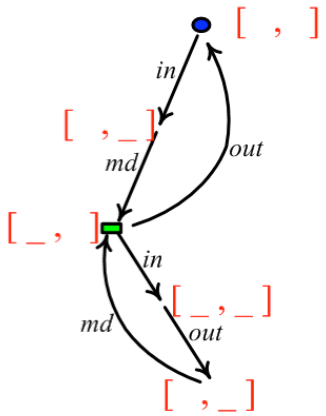

Figure 5: Traces and state transitions of $Machine + Buff1 + Env$

5 Discussion and conclusions

In this paper, we have shown that the combination of PF and the quotient operator $\backslash\backslash$ of [LS95] provides a constructive method of determining a solution to a problem expressed in the problem frame notation, at least when the language for the description of the domain is CSP-based.

In the process, we have shown how different languages can work together in PF, and how one can move between the problem frame notation and the description of domains. We have also provided some clarification and examples of the semantics in action.

We have shown how problem frames can be used when a formal language, CSP, is chosen as domain and requirements description language. This has allowed us to use Lai's quotient operator to investigate the formal semantics of Problem Frames in terms of CSP construct with the correct alphabet. We related the constructed solution to the correctness argument of the Required Behaviour problem frame, and showed that it was a correct solution, as would be expected.

When problem frames have CSP as their description language the interpretation of the sharing of phenomena is naturally that of parallel composition in CSP. This has allowed us to interpret the challenges for a problem diagram in CSP in terms of the parallel composition of its CSP-described domains. This would appear to ground the guidance of [ZJ93] on the use of conjunction as composition.

In this presentation, we have not been able to explore important issues that arise in PF. For instance, we were forced to include an environment *Env* which could represent any process that can use a two-place buffer. This might be a person, and so biddable; how this should be represented in CSP is unknown to us. An important point to note is that discharging a correctness argument in other situations may require more information to be known of such an environment.

Further work will include the development of other examples in which languages other than CSP are used to describe domains, the formalisation of composition through shared phenomena of domains in problem diagrams, and the investigation of other ways of determining constructively the elements of the solution set of a problem diagram.

References

[BKNS97] D. Bjorner, S. Koussoube, R. Noussi, and G. Satchok. Michael jackson's problem frames: Towards methodological principles of selecting and applying formal software development techniques and tools. In *1st IEEE Int Conf on Formal Engineering Methods*, pages 263–270, Hiroshima, Japan, 1997. IEEE Comp Soc Press.

[BT76] T. E. Bell and T. A. Thayer. Software requirements: Are they really a problem? In *Proceedings of the 2nd international conference on Software engineering*, pages 61 – 68, San Francisco, USA, 1976. IEEE Computer Society Press.

[CoF99] CoFI. CASL the common algebraic specification language summary, version 1. Technical report, The Common Framework Initiative for Algebraic Specification and Development, 1999.

[CR99] Christine Choppy and Gianna Reggio. Using CASL to specify the requirements: a problem specific approach. In *Proceedings of the 14th International Workshop on Algebraic Development Techniques (WADT'99)*, 1999.

[Del02] Gaetan Delannay. A meta-model of Jackson's Problem Frames. Technical report, University of Namur, 2002.

[GGJZ00] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, 2000.

[GMB94] Sol Greenspan, John Mylopoulos, and Alex Borgida. On formal requirements modeling languages: Rml revisited. In *Proceedings of the 16th International Conference on Software Engineering*, pages 135 – 147, Sorrento, Italy, 1994. IEEE Computer Society Press.

[HJL+02] Jon G. Hall, Michael Jackson, Robin Laney, Bashar Nuseibeh, and Lucia Rapanotti. Relating software requirements and architectures using problem frames. In *RE'02*, pages 137–144, Essen, Germany, 2002. IEEE Computer Society Press.

[HRJ04] J. Hall, L. Rapanotti, and M. Jackson. Problem frame semantics for software development. *Journal of Software and Systems Modelling*, 2004.

[Jac95] Michael Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley Publishing Company, 1995.

[Jac01] Michael Jackson. *Problem Frames: Analyzing and Structuring Software Development Problem*. Addison-Wesley Publishing Company, 1st edition, 2001.

[JZ93] Michael Jackson and Pamela Zave. Domain descriptions. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages pages 56–64. IEEE CS Press, 1993.

[LS95] Luming Lai and J. W. Sanders. A weakest-environment calculus for communicating processes. Research report PRG-TR-12-95, Programming Research Group, Oxford University Computing Laboratory, 03-1995 1995.

[LS97] Luming Lai and J. W. Sanders. A refinement calculus for communicating processes with state. In *1st Irish Workshop on Formal Methods*, 1997.

[Mor94] Carroll Morgan. *Programming from Specifications*. Prentice Hall International Series in Computer Science. Prentice-Hall International, 1994.

[Nus01] B. Nuseibeh. Weaving together requirements and architectures. *IEEE Computer*, 34(3):115–117, 2001.

[Pet95] Marian Petre. Why looking isn't always seeing: readership skills and graphical programming? *Communications of the ACM*, 38(6):33 – 44, June 1995.

[ZJ93] Pamela Zave and Michael A. Jackson. Conjunction as composition. *ACM Transactions on Software Engineering and Methodology*, 2(4), October 1993.

[ZJ97] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, Vol. 6 No. 1, January 1997.