ABSTRACT

Achieving a proper understanding of the problem space before providing the design in the solution space is one of the basic tenets in requirements engineering. The Problem Frames approach provides a way for people to understand and solve software problems.

Recently, a denotational semantics for Problem Frames was defined to relate the various elements of Problem Frames together. One of the problems of the semantics is that, as denotation, a problem has the set of all satisfying solution specifications. Whereas this is a sensible initial choice, it does not lend itself easily to the construction of solution specifications.

The contribution of this paper is to provide a formal technique which in the context of the given semantics allows for the systematic derivation of software specifications from requirements.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specifications

General Terms
Theory, Design

Keywords
Requirements, Specifications, Problem Frames, CSP

1. INTRODUCTION

Achieving a proper understanding of the problem space before providing the design in the solution space is one of the basic tenets in requirements engineering (RE). The problem space refers to the environment in which the requirements and application domains reside, and the solution space refers to the computer system in which the hardware and software are to be co-designed. In [13] and [4], Jackson and Zave argue that “requirements exist only in the environment” and domains should be described explicitly independent of the system to be built.

The Problem Frames approach (PF) provides a way for people to explore the broader contexts, provide domain descriptions, and solve software problems [3]. The approach can help guide early life-cycle requirements and domain analysis in the problem space so as to derive software specifications in the solution space.

The second and third authors (together with Jackson) have defined a PF semantics [1] that relates the various elements of PF together. The semantics is that, as denotation, a problem has the set of all satisfying solution specifications. Whereas this is a sensible initial choice, it does not lend itself easily to the construction of solution specifications.

In this paper we show how the quotient operator \( \backslash \) of [5] provides a constructive method of determining a solution to a problem expressed in PF notation. Given certain assumptions, the application is quite natural, and can be seen as illustrating what is, perhaps, a deep correspondence between, on the one hand, Jackson’s notion of sharing of phenomena and contextual domains and, on the other, parallel composition as found in many process algebraic frameworks. The assumptions we make in this paper lessen the applicability of Lai’s operator to those situations in which a fully formal description is given to environmental processes, i.e., everything is expressed in a version of CSP process algebra [2], which includes specification statements.

This paper is organised as follows. Section 2 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 3 illustrates their linkage with an example of software development using Problem Frames. Section 4 evaluates the results and provides pointers to new developments.

2. BACKGROUND

2.1 Problem Frames and their semantics

Problem frames are a concretisation of many Jackson’s ideas on requirements specification into a framework for representing, analysing and classifying software problems. This section only provides a brief overview of the aspect of the framework which are more relevant to our work. A much more complete treatment can be found in [3]. Problem frames have inspired a problem oriented approach to requirements engineering in which problems are transformed – in solution-preserving ways – towards solutions. This comple-
ments other approaches such as those based on, for instance, goals [11] [12] and scenarios [7].

In PF, problems are represented through the graphical notation of problem diagrams. This defines the context of a problem by capturing the characteristics and interconnections of the parts of the world the problem is concerned with, as well as a requirement to be satisfied in such a context. Figure 1 gives a simple example of a problem diagram. In the figure, the software problem is to specify a machine \( M \) (the solution) in the given problem context (the collection of given domains \( D_1 \) to \( D_3 \) and their interconnections), so that the requirement \( R \) is satisfied.

![Figure 1: A problem diagram with multiple domains](image)

Problem diagrams also show phenomena shared between the given domains, and between context and machine. Such shared phenomena provide the vocabulary for the problem, and capture a variety of entities of interest to the problem, such as values, events, commands or operations. In Figure 1, the labels on the links between domains, and between domains and machine, name the phenomena that are shared between them; for instance, there is a set of shared phenomena \( c_1 \) shared between \( M \) and \( D_1 \). That such phenomena are controlled by machine \( M \) is indicated by the prefix \( M! \).

Provided separately from the diagram are the designations of phenomena, which ground the problem’s vocabulary in the physical phenomena. It is typical of problem diagrams formed early within problem discovery that the requirement \( R \) is situated very far from the machine; in the figure, \( R \) is connected to domain \( D_3 \) which interacts only indirectly with the machine through \( D_1 \) and \( D_2 \). One of the goals of a problem oriented approach to requirements engineering is to move the requirement closer to the machine; requirements expressed closer to the machine are easier to solve\(^1\) that those far away, as one does not have the complicating behaviour of the intervening domains to cope with. Our goal in this paper is to show – for one particularly simple case – how to move requirements closer to the machine.

Not included in the problem diagram, but an essential part of the problem definition, are the descriptions of the given domains and of the requirement. It is notable that the PF does not prescribe a notation to be used for these descriptions; formal and informal descriptions are equally acceptable, provided they are precise enough to capture the characteristics of interest of the various respective domains.

The purpose of rigour and precision in the analysis of a problem is to determine the specification of a solution that is implementable and that can be argued to satisfy the requirement in the given context. As for domain and requirement descriptions, the specification need not be formal; it should, however, be sufficiently precise to permit further design steps to be undertaken (in this paper we describe one such design step when descriptions are written in CSP). Precision is also required in order to argue that a solution specification indeed satisfies the requirement in the given context. In PF, such an argument is known as a correctness argument; it need not be formal, only sufficient to convince us that the proposed solution specification solves the problem. In other words, correctness should not be interpreted too narrowly: PF acknowledges that there are parts of the real-world that escape formal description, but whose properties need to be taken into account nevertheless. A formal proof is, in most realistic cases, unobtainable. This is widely acknowledged in software development, where testing – something acceptable as correctness within PF – is the de facto form of correctness argument.

As well as analysing new problems, PF also provides the means by which recurrent software problems can be categorised for reuse. \([3]\) introduces five initial basic problem classes, including the Required Behaviour frame that we apply in the next section. This class is representative of a class of problems in which a control machine has to be developed to control some part of the context, e.g., the controller of a sluice gate or a lift. Problem frames capture this class of problems in the diagram of Figure 2. The diagram indicates both the form a problem diagram of this class should have and also gives a template for the appropriate correctness argument. (The \( C \) annotation on the controlled domain indicates that for problems of this type such a domain is causal, that is its properties include predictable causal relationships between its phenomena.)

![Figure 2: The Required Behaviour problem class](image)

Hall et al. \([1]\) proposes a formal semantics of problem diagrams, based on the following descriptions:

- 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 \( a \) observed by the machine \( (a_1 \cup a_2 \) in the figure).

All descriptions should be expressed in some chosen domain and requirement description language (DRDL, for short), a (possibly heterogeneous collection of) description language(s). The meaning of a problem diagram is then defined as a “challenge” to find a specification for the machine \( M \) which

\(^{1}\)A more accurate statement is that they are no harder to solve!
satisfies $R$ in the given $K$, or, more precisely, to find a member of the set

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

Like the propositional calculus (for instance, [8, Chapter 1]), Hall et al.'s semantics works with descriptions at two levels: the semantic meta-level combining and allowing the manipulation of 'atomic descriptions' described in the DRDL. For instance, the semantics 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 $x = 0 \land x' = 1$. This matches the multi-paradigm assumption of PF [4]. Notions of correctness, including $\vdash_{DRDL}$ in the definition of a challenge, is made with respect to (the more or less formal) correctness notions in DRDL.

The notation for a challenge is intentionally reminiscent of the notation for a specification statement of [9], but a formal notion of a refinement relation is currently missing from the semantics of [1]. This leads us to look for stepwise constructive methods to complete a problem diagram's challenge; part of the contribution of this paper is to look to CSP and Spec (both of which are described later) as possible description languages for PF, and to use the tools of a CSP version of formal refinement to solve a challenge constructively.

### 2.2 The weakest-environment calculus

Lai et al.'s work in the weakest-environment calculus [6] provides:

- a homogeneous framework for the specification and development of parallel programs which [...] guarantees functional correctness of an implementation as a consequence of development using its laws.

Essentially, CSP is extended with the specification statement for which a selection of sound refinement laws are made with respect to (the more or less formal) correctness notions in DRDL.

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

- $<a_1, \ldots, a_n>$ is the trace consisting of elements $a_1, \ldots, a_n$
- $|tr|$ is the length of a trace $tr$

$ur \leftarrow tr$ 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 $c$.

In addition, for the predicate $p$, we will write

$$a \leftarrow p \leftarrow b \iff (p \Rightarrow a) \land (\lnot p \Rightarrow b)$$

For Lai, a specification is a predicate with free variables $tr$ (representing a trace) and $ref$ (representing the refusal set after that trace) and a notion of a communication alphabet. A simple example of a specification is $Spec_1$ defined as the predicate

$$tr = () \land ref \subseteq A$$

with alphabet $\alpha Spec_1 = A$ satisfied by the process that, in its initial state, refuses to do anything. As stated in [5], 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 with alphabet $A$ that does nothing, satisfies $Spec_1$ is written $stop_A sat Spec_1$.

A specification $P$ is a process if the following conditions on $tr$ and $ref$ are met [5]:

1. $P(\{\}, \{\})$
2. $P(tr^{-} ur, \{\}) \Rightarrow P(tr, \{\})$
3. $Y \subseteq X \land P(tr, X) \Rightarrow P(tr, Y)$
4. $P(tr, X) \land \neg \exists v : val(c) \bullet P(tr^{-} (c, v), \{\}) \Rightarrow P(tr, X \cup \{c\})$

#### 2.2.1 Lai's quotient

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

$$P \parallel R(tr, ref) \triangleq \forall ur : traces(R) \forall \text{rep} \subseteq \alpha P \quad (tr = ur \mid \alpha (P \parallel R) \land P(ur \mid \alpha P, \text{rep})) \Rightarrow R(ur, \text{rep} \cup ref)$$

(i.e., that the trace/refusal set pair of the quotient must contribute to the traces and refusals of the requirement (the $ur$ and $\text{rep}$) in the right measures).

### 3. Applying the Quotient Operator

In this section we will demonstrate how Lai's quotient operator can be extended to Requirements Engineering, and in particular how it can be used to derive in a systematic fashion a specification from a requirement when a PF approach is taken. In our development, we will make use of an example from the literature which we will re-interpret within PF. We will then show how a solution can be derived through the application of the quotient operator, including
the discharge of the correctness argument. That the example is simple means that the reader will not be overwhelmed by notation.

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 [13] that only a domain that controls a phenomenon should be able to change it. As pointed out in [13] 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 $P$ define $P! = \{ c | c! \text{ appears in } P \}$; a collection of CSP-described domains $P_1, ..., P_n$ are said to be control determined if and only if $P!$ 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 $\iota$ can be made total by adding $\iota_M(c) = M$ whenever $\iota(c)$ is undefined$^4$. 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 $\alpha M = \text{dom}(\iota_M) \cup \text{observed}$, 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, $\parallel$, fits the bill nicely: indeed, after [14], the interpretation is quite natural: CSP parallel composition is, essentially, conjunction with channel phenomena — channel communications such as $c!5$ and $c?2$ — shared (and unhidden).

Finally, we must determine the notion of correctness that we will use in discharging our correctness arguments. The obvious candidate is the satisfaction relation sat between processes; indeed, we choose $\vdash_{\text{DRDL}} = \text{sat}$.

With these interpretations, we may, for a problem diagram consisting of CSP-described domains, $C_0, ..., C_n$, requirement $R$ with a topology that allows the machine to control $c$ and observe $o$, write the solution machine as any in the set
\[
\{ S : \text{Specification} \} \\
S \text{ controls } \iota_M(c) \\
\wedge S \text{ observes observed} \\
\wedge C_n |\ldots| C_0 |S \text{ sat } R
\]

Applying the quotient operator, we see that
\[
\{ S : \text{Specification} \} \\
S \text{ controls } \iota_M(c) \\
\wedge S \text{ observes observed} \\
\wedge S \text{ sat } (C_n |\ldots| C_0) \setminus R
\]

which is a much easier challenge to discharge: we have factored out all intervening domains $C_i$ so that the transformed requirement $(C_1 |\ldots| C_n) \setminus R$ is directed at the machine.

In the next section we present an example of the combination of PF with CSP, to show a) how the two work together, and b) the deeper semantic relationship between the two at the level of the correctness argument.

### 3.1 The problem statement

The problem is to build a machine to interact with a one-place buffer system so that their composition is a two-place buffer$^5$. A problem diagram for this problem is given in Figure 3.

![Figure 3: The initial problem diagram](image)

### 3.2 Domain descriptions

Informally, a one-place buffer $\text{Buff}1$ repeatedly inputs and faithfully outputs values of some fixed type, ensuring that output lags at most one value behind input. It has two external channels, $in$ for input and $md$ for output, both of that type, so that $\alpha \text{Buff}1 = in, md$. Formally, as a Spec, it can be described as [5]:

\[
\text{Buff}1(tr, ref) \triangleq (md \preceq_1 in) \\
\wedge (in \not\in ref \Rightarrow in = md \wedge md \not\in ref)
\]

The environment $\text{Environment}$ can both offer to input a value to $\text{Buff}1$ 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 $\alpha in, out$ with:

\[
\text{Environment}(tr, ref) \text{ sat true}
\]

The requirement $\text{Buff}2$ can be expressed as a process representing the behaviour of a two-place buffer which repeatedly inputs and faithfully outputs values of some fixed type, ensuring that output lags at most two values behind input. $\text{Buff}2$ can be expressed as a Spec predicate with two external channels, $in$ for input and $out$ for output, so that $\alpha \text{Buff}2 = in, out$. Its description is:

\[
\text{Buff}2(tr, ref) \triangleq (out \preceq_2 in) \\
\wedge (in \not\in ref \Rightarrow in \not\in out \wedge ref)
\]

It is important to note that the environment in which the composition of the solution and buffer will operate is given explicit representation. Such an environment is essential to the analysis of the problem from a requirements engineering viewpoint, for which requirements are in the context, not applied to the machine. In particular, requirements express properties the context should have once a solution is provided, rather than the properties of the machine itself. This is the essential distinction between requirement and specification. As a consequence, the environment with which the machine shares phenomena has to be made explicit.

$^5$This is a simple problem, but not unrepresentative of the co-design problem mentioned in [10].
In terms of the semantics of [1] the problem diagram is a challenge to find an element of
\[
c, a : [K, R] = \{out\}, \{md\} : [Buff1 \parallel Environment, Buff2]
\]
In what remains of this section, we show how this challenge can simply be met through the application of the quotient operator, with particular emphasis on the form of the correctness argument.

3.3 Problem transformation

Before applying the quotient operator, we want to transform the problem so that it fits a required behaviour frame (see Section 2). The benefit of this transformation is that we wish to take advantage of the template to build the correctness argument for the solution to our problem. To transform the problem, we collate the description of Buff1 and Environment. We do this through the parallel composition of their description under CSP. More precisely, process Buff1 and Environment can be merged into one process Buff1 \parallel Environment, whose Spec description is:

\[
Buff1 \parallel Environment(tr, ref) \triangleq \\
\quad (md \leq in) \\
\quad \land (in \not\in ref < in = md \lor md \not\in ref)
\]

The transformed problem diagram is given in Figure 4, where the combined new domain is denoted by Buff1&Environment.

Figure 4: Problem diagram after transformation

3.4 Solving the problem

Applying Lai’s quotient provides the following as satisfying solution specification: the problem:

\[
Machine \triangleq (Buff1 \& Environment) \setminus Buff2
\]

which, expanding the definition, is the process:

\[
Machine(tr, ref) \triangleq \\
\quad (out \leq^1 md) \\
\quad \land (md \not\in ref < md = out \lor out \not\in ref)
\]

3.5 Correctness argument

We have transformed the problem for that it fits the Required Behaviour frame. As such the solution specification should be provided with a correctness argument that follows the templated correctness argument attached to this frame (as shown in Figure 2). Here we develop such a correctness argument.

The reader will recall that there were various parts to the requirement. In particular we need to argue that:

a. it is always the case that out \leq 2 in;

b. when out = in then in is possible;

c. when out < 1 in then both in and out are possible;

d. when out < 2 in then out is possible.

Here is the instantiation of the argument tailored to our problem.

1. We have built the machine to behave like this:
   a. it is always the case that out \leq 1 md (i.e., the number of outputs from the machine is at most one less than the number of inputs);
   b. when out = md (i.e., there have been an equal number of inputs and outputs to the machine), it can share an md with Buff1 & Environment;
   c. when out < 1 md (i.e., there has been one more input than outputs), it can share an out with Buff1 & Environment;

2. ... so that, knowing that the Buff1 & Environment behaves like this:
   a. it is always the case that md \leq 1 in (i.e., the number of md is at most one less than the number of in);
   b. when md = in (i.e., there have been an equal number of those phenomena), it can perform an in;
   c. when md < 1 in (i.e., there has been one more in than md), it can share an md with Machine

3. ... we will be sure that the requirement Buff2 is satisfied:
   a. as it is always the case that out \leq 1 md and md \leq 1 in, then it is always the case that out \leq in;
   b. when out = in then out = md and md = in, from the description of Buff1 & Environment we know that in is possible; hence the requirement is satisfied in this case;
   c. when out < 1 in then there are two possible alternatives:
      c1. if out < 1 md and md = in, from the description of Buff1 & Environment we know that in is possible, and from that of Machine we then know that out is possible; hence the requirement is satisfied in this case;
      c2. if out = md and md < 1 in, from the descriptions of Buff1 & Environment and Machine, we know that md is possible, and the occurrence of md takes us to case c1.; hence the requirement is satisfied in this case;
   d. when out < 2 in then out < 1 md and md < 1 in, from the description of Machine we know that out is possible; hence the requirement is satisfied in this case.

4. DISCUSSION AND CONCLUSIONS

In this paper, we have shown that the combination of PF and the quotient operator \setminus of [5] 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 particular, we have shown how PF 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 [14] on the use of conjunction as composition.

However, having CSP descriptions is a severe restriction, especially for requirements engineering, in which such descriptions are rarely encountered. Nonetheless, inspired by the combination of Lai’s quotient and PF, we are actively investigating parallels of the quotient operator for, what might be called, more traditional PF descriptions, such as natural language.

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 Environment 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.

5. ACKNOWLEDGMENTS

We are pleased to acknowledge the financial support of IBM, under the Eclipse Innovation Grants, and the EPSRC, Grant number EP/C007719/1. Thanks also go to our colleagues in the Centre for Research in Computing in The Open University, particularly Michael Jackson, Bashar Nu-seibeh and John Brier.

6. REFERENCES