Technical Report No: 2003/09

Problem Frames for Socio-technical Systems

Jon G. Hall
Lucia Rapanotti

15th October 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

Problem Frames for Socio-technical Systems

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

1 Introduction

By socio-technical system we mean a collection of interacting components in which some of the components are people and some are technological. In this chapter, we focus on the requirements analysis of socio-technical systems in which some of the technological subsystems are computer-based, these systems forming the largest part of modern software design problems.

More precisely, there are two (not necessarily disjoint) sub-classes of socio-technical systems that we will treat in this chapter. The first sub-class contains those systems in which existing components or sub-systems (i.e., domains) are to be allowed, through software, to interact. An example from this first class might be the problem of designing software for the operator of heavy machinery. The larger, second class, contains those systems for which software, a user interface, and user instruction is to be designed to enable a new process or service. An example of this second class might be the development of a new customer call centre.

The use of Problem Frames (PFs) underpins our requirements analysis process. As described in [8], PFs are a concretisation of the ideas of Michael Jackson and others in the separation of machine and its environment's descriptions. This separation is generally accepted as being a useful principle for requirements analysis[1]. The usual representation of the separation of machine and environment descriptions is as the 'two ellipse' model, illustrated in Figure 1. In that figure world knowledge W is a description of the relevant environment; R is the statement of requirements; S is the specification that mediates between environment and machine; M is the description of the machine; and P is the program that, on machine

[1] We will have cause, later in the chapter, in dealing with a more general class of socio-technical problems to further detail this separation, but nothing we do compromises its fundamental status.

M, implements the specification S. The role of W is to bridge the gap between specification S and requirements R. More formally[14, 4, 5]:

$$W, S \vdash R$$

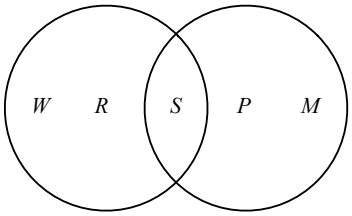


Figure 1: The requirements analysis model

One of the aims of the PF framework is to identify basic classes of problems that recur throughout software development. Each such class should be captured by a *problem frame* that provides a characterisation for the problem class. Socio-technical systems are an important class of problems, and so should be representable within the PF framework, possibly with their own (collection of) problem frame(s).

In a fundamental sense, of course, the PF framework already deals with socio-technical systems: problem frames are an attempt to allow customer and developer to come together to match real-world problem and technological solution. There are many examples in [10] as to how this relationship can be facilitated using problem frames.

We observe, however, that the application of problem frames to particular socio-technical problems remains under-explored. Currently, some discussion of HCI appears in [10], and some analysis appears in [8], but otherwise, there is little in-depth coverage of how to apply problem frames in this context. In this chapter we show, in some detail, how problem frames can be applied to socio-technical systems.

Our development is threefold. We first show how the problem of representing interaction with (and not just control of) technology can be represented within the PF framework. To do this we introduce two new basic problem frames, the *User Interaction Frame* and the *User Commanded Behaviour Frame*, each dealing with the class of user-interaction problems.

Secondly, we show how architectural artefacts can be used to guide the analysis of socio-technical problems. To do this we discuss the notion of an AFrame, a new PF artefact that can be used to guide problem decomposition in the light

of particular solution expertise as might, for instance, exist in a software development company. As an exemplar of AFrames and their use, we define and apply an AFrame corresponding the MVC architectural style [1].

Lastly, we adapt the PF framework to meet the needs of representing the problems of more complex socio-technical systems, including those in which, as well as user-machine interaction, user training needs to be addressed. This causes us to consider a reification of the two ellipse model into three ellipses, to represent machine, environment, and user descriptions. Consequently, by interpreting this third ellipse in the PF framework, we discover a new type of PF domain – the knowledge domain – to represent a user's knowledge and their 'instruction' needs, and argue that this leads to a more general PF framework for socio-technical systems.

1.1 Chapter Overview

In the major part of this chapter, we will use the well-known chemical reactor problem [2, 11], as an example for illustration of techniques. Later we briefly describe the design of a 'cold-calling' system. The chemical reactor is a socio-technical system, and is representative of the class of operator controlled safety-(and mission-)critical systems. A schematic for the chemical reactor hardware appears in Figure 2.

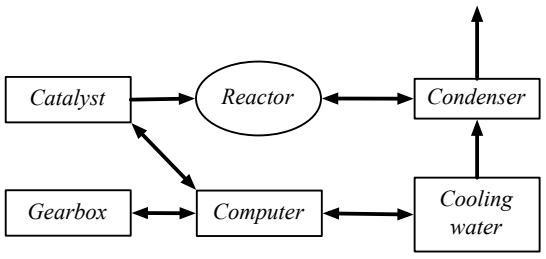


Figure 2: The chemical reactor schematic (adapted from [2])

A statement of the problem is as follows.

> A computer system is required to control the safe and efficient operation of the catalyst unit and cooling system of a chemical reactor. The system should allow an operator to issue commands for activating or deactivating the catalyst unit, and to monitor outputs. Based on the operator's commands, the system should instruct the unit accordingly and regulate the flow of cooling water. Attached to the system

is a gearbox: whenever the oil level in the gearbox is low, the system should alert the operator and halt execution.

The chapter is organized as follows. Section 2 develops the problem frame representation of the chemical reactor problem, and uses this to recall the basis of problem representation in the PF framework. Section 3 provides a small taxonomy of problem classes including those of relevance to socio-technical systems. Section 4 addresses problem decomposition, both in the classical PF framework, and through AFrames. Section 5 details our separation into three of the various domain descriptions, and uses this to motivate a new type of PF domain, the knowledge domain. Section 6 concludes the chapter.

2 Problem representation

We first consider the PF representation of the chemical reactor problem. Within the PF framework, problems are represented through *problem diagrams*. A problem diagram defines the shape of a problem: it records the characteristic descriptions and interconnections of the parts (or *domains*) of the world the problem affects; it places the requirements in proper relationship to the problem components; it allows a record of concerns and difficulties that may arise in finding its solution.

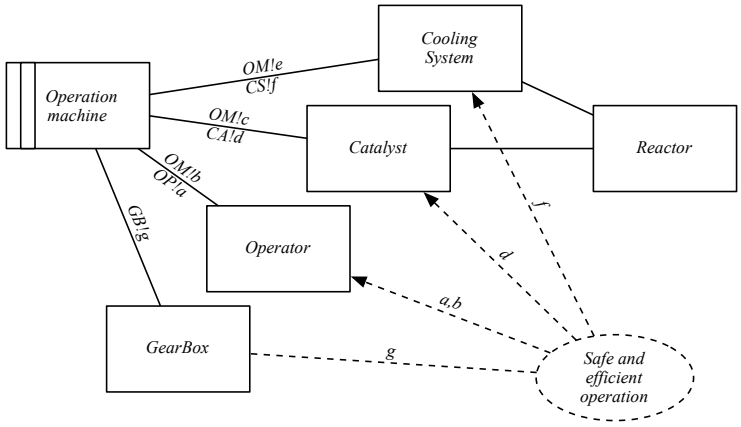
For the chemical reactor, there are a number of domains, including those that appear in the schematic of Figure 3. Also the operator will play an important role, in issuing commands to control the catalyst and cooling systems. Placing all of these domains in their correct relationship one to another leads to the problem diagram shown in Figure 3.

The components are:

- *Operation machine*: the *machine domain*, i.e., the software system and its underlying hardware. The focus of the problem is to build the Operation machine.

- Other boxes (*Cooling System*, *Catalyst*, *etc.*): *given domains* representing parts of the world that are relevant to the problem.

- *Shared phenomena*: the ways that domains communicate. These can include events, entities, operations and state information. In Figure 3, for example, the connection between the *Operation machine* and the *Cooling System* is annotated by a set *e*, containing the events *increase_water* and *decrease_water*, and a set *f*, containing the phenomenon *water_level*. Phenomena in *e* are controlled by the *Operation machine*; this is indicated by an abbreviation of



$a : \{open\_catalyst, close\_catalyst\}$ $e : \{increase\_water, decrease\_water\}$
$b : \{catalyst\_status, water\_level\}$ $f : \{water\_level\}$

$c : \{open\_catalyst, close\_catalyst\}$ $g : \{oil\_level\}$
$d : \{is\_open, is\_closed\}$

Figure 3: The chemical reactor problem diagram

the domain name followed by *!*, i.e., *OM!*. Similarly, the phenomenon in f is controlled by the *Cooling System*, indicated by the *CS!*.

- The dotted oval *Safe and efficient operation*: the *requirement*, i.e., what has to be true of the world for the (operation) machine to be a solution to the problem.

- In the connections between the requirement and the domains, a dotted line indicates that the 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). In Figure 3, for instance, the oil level in the gear box is referenced, while the cooling system's water level is required.

- Phenomena at the requirement interface (e.g., those of sets f or d) can be, and usually are, distinct from those at the machine domain interface (e.g., those of sets c and e). The former are called *requirement phenomena*; the latter, *specification phenomena*. The intuition behind this distinction is that the requirement is expressed in terms of elements of the problem, while the

specification (that is what describes a machine domain) is expressed in terms of elements of the solution.

Other artifacts which are not represented on the problem diagram, but are related to it are *domain and requirement descriptions*. Such descriptions are essential to the analysis of a problem, and address relevant characteristics and behaviours of all given domains, the machine domain and the requirement.

An important distinction in the PF framework is that of separating two types of descriptions: *indicative* and *optative*. Indicative descriptions are those which describe how things are; optative descriptions describe how things should be. In this sense, in a problem diagram, given domain descriptions are indicative, while requirement and machine descriptions are optative. In other words, things that have to do with the problem domain are given, while things that have to do with the solution domain can be chosen. For instance, in the chemical reaction problem indicative descriptions of the *Catalyst*, *Cooling System* and *Gear Box* should include characteristics of the domains which are of interest in the specification of the machine, say, the mechanics of opening the catalyst, or of changing the water level in the cooling system, or the oil level in the gear box. On the other hand, the requirement should express some constraints on the status and operations of those domains which, when satisfied, result in their safe and efficient operation. Indeed, it is our intent to build a machine so that they are satisfied at all time. Finally, the machine specification should describe how we would like the control system to behave and interact with the given domains so that the requirements are met.

Indeed not all domains share the same characteristics. There is a clear difference between a cooling system and an operator. In a cooling system there exist some predictable causal relationships among its phenomena. For instance, it could be described as a state machine, with a set of clearly identified states and predicable transitions between them. A domain with such characteristics is known as a *causal* domain. On the other hand, an operator's phenomena lack such predictable causal relationships. We can describe the actions an operator should be allowed to do, but can not guarantee that they will be executed in any particular order, or not at all, or that some other (unpredicted) actions will not be executed instead. A domain with such characteristics is known as *biddable*.

The distinction between causal and biddable domains is an important one, as it has ramifications for the type of descriptions we can provide, and the assumptions we can make in discharging proof obligations during the analysis of a problem, as we will see in the following sections.

Of course, there exist other types of domain, each with its own characteristics. An exhaustive domain classification is beyond the scope of this chapter but can be found in, e.g., [10, 8, 6].

3 Problem Classification

One of the intentions of the PF framework is to classify problems. An initial problem classification is given in [10]. Therein are identified five *basic* problem frames. They are basic because they represent relatively simple, recurrent problems in software development. In describing the basis of problem frames, it is not the intention of that work to be exhaustive in its classification of problems. Indeed, there are other frames by Jackson [8] which do not make it into that classification.

In this section, we present a simple taxonomic development, beginning with the simplest of all problem frames, and adding domain types (modulo topology). As each type has different characteristics, the resulting problem frames represent different classes of problems. In doing so, we introduce a new basic problem frame, the User Interaction Frame, which is novel in the problem class it represents within the PF framework (but not, of course, within software engineering). The taxonomy is summarised in Figure 4.

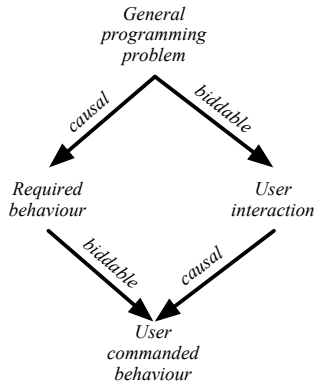


Figure 4: Simple problem frame taxonomy: adding domain types

3.1 Programs

The simplest form of problem representable in problem frames is that of producing a program from a given specification. This is illustrated in Figure 5. Although this is a sub-problem of all software engineering problems, it is not a very interesting problem class to be analysed using PFs: nothing exists outside the machine. Other techniques, such as JSD [7] or design patterns [3], are probably more appropriate to tackle this problem class. Indeed, the PF framework does not include a problem

frame for this class of problems.

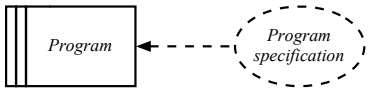


Figure 5: Writing programs

3.2 Embedded controllers

For a problem class to be purposefully analysed in PFs, some given domain(s) of interest must exist outside the machine. An interesting problem class is identified in [10] by introducing just one single causal domain. The problem frame is known as the *Required Behaviour Frame*, and its characterising problem is that of building a machine that controls the behavior of some part of the physical world, so that it satisfies certain conditions. Some software engineers may find it easy to identify this problem with that of building an embedded controller (although it does apply also to more general software problems).

The Required Behaviour Frame is illustrated in Figure 6. The frame has a topology, which is captured by a *frame diagram* (that of the illustration).

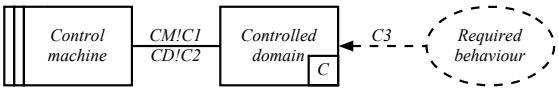


Figure 6: Required Behaviour Frame

The frame diagram resembles a problem diagram, but it also includes some further annotation. This provides an indication of the characteristics of the domains and phenomena involved in problems of the class. For the Required Behaviour Frame:

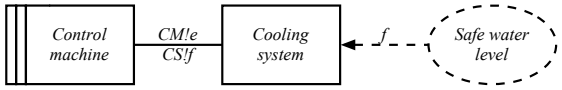
- The *Controlled domain* is causal (the C annotation in the figure). Its phenomena are also causal (indicated by C on the arcs) - they are directly caused or controlled by a domain and may cause other phenomena in turn.
- The *Control machine* has access to causal phenomena of the *Controlled domain* (in $C2$) and controls another set of phenomena which are also shared with the *Controlled domain* (in $C1$). Intuitively, phenomena in $C1$ are used

by the machine to control the domain, while phenomena in $C2$ to obtain information and feedback on the functioning and state of the domain.

- The requirement, *Required behaviour*, is expressed in terms of a set ($C3$) of causal phenomena of the *Controlled domain*.

When a problem of a particular class is identified, it can be analysed though the instantiation of the corresponding frame diagram. The instantiation is a process of matching problem's and frame's domains and their types, as well as problem's and frame's phenomena types. The result of the instantiation is a problem diagram, which has the same topology of the frame diagram, but with domains and phenomena grounded in the particular problem.

Let us return to the chemical reactor problem and consider the problem of regulating the water level in isolation. This can be regarded as a required behaviour problem (see Figure 7) with some safety requirement on the water level.



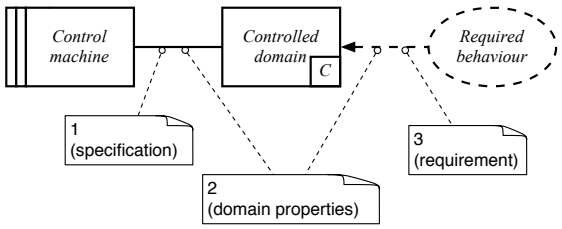
$e : \{increase\_water, decrease\_water\}$
$f : \{water\_level\}$

Figure 7: Regulating the water level in the cooling system as a required behaviour problem

For a problem to be fully analysed, the instantiation of a problem frame is only the first step of the process. Suitable domain and requirement descriptions (see Section2) need to be provided and the frame concern needs to be addressed. The frame concern is an overall correctness argument, common to all the problems of the class. It is the argument that must convince you, and your customer, that the specified machine will produce the required behaviour once combined with the properties of the given domains. Each problem frame comes with a particular concern, whose structure depends on the nature of the class problem. For the Required Behaviour Frame, the argument is outlined in Figure 8.

3.3 User Interaction

Another interesting class of problems can be obtained by including a single biddable domain outside the machine, which represents the user of the system. We



1 We will build a machine that behaves like this, so that...

2 knowing that the controlled domain works like this...

3 we can be sure that its behaviour will be this.

Figure 8: Frame concern for the Required Behaviour Frame

call the resulting problem frame the *User Interaction Frame*, and its characterising problem is that of building a machine, which enforces some rule-based interaction with the user. The frame diagram for the User Interaction Frame is given in Figure 9.

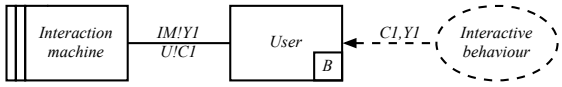


Figure 9: User Interaction Frame

The *Interaction machine* is the machine to be built. The *User* is a biddable domain representing the user who wants to interact with the machine. The requirement gives the rules which establish legal user/machine interactions.

The manifestation of the user/machine interaction is though exchanges of causal phenomena (in $C1$) controlled by the user and symbolic phenomena (in $Y1$) controlled by the machine. Intuitively, the user issues commands in $C1$ and the machine provides feedback through $Y1$. The interaction rules specify the legal correspondence of user and machine phenomena.

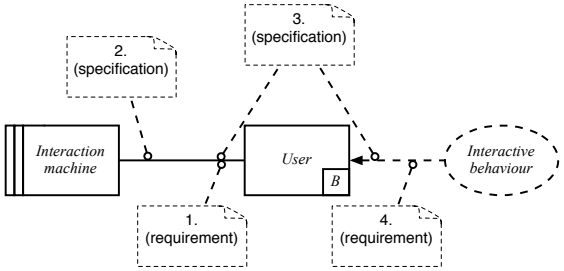
In the chemical reactor problem, we can isolate the operator/machine interaction as a user interaction problem as illustrated in Figure 10, where the requirement establishes some rules on the relationship between operator's commands and system feedback, say, that a *open_catalyst* command cannot be issued when the *water_level* value is below a set threshold. Note that this would be the perspective of a UI designer, whose main concern is the user interacting with a black box system. Indeed, in the wider problem analysis of the chemical reactor problem,

machine responses to user commands depend on a faithful representation of the internal state of, say, the catalyst or the cooling system.

$$a : \{open\_catalyst, close\_catalyst\}$$
$$b : \{catalyst\_status, water\_level\}$$

Figure 10: Operator/system interaction as an instance of the User Interaction Frame

Figure 11 illustrates the frame concern for the User Interaction Frame.



1 Given this set of machine phenomena, when the user causes this phenomena (it may or may not be sensible or viable)...

2 if sensible or viable the machine will accept it...

3 resulting in this set of machine phenomena...

4 thus achieving the required interaction in every case.

Figure 11: Frame concern for the User Interaction Frame

3.4 User Commanded Behaviour

The Required Behaviour and the User Interaction Frames are representative of relatively simple problems, albeit oft-recurring in software development. It is possible, and indeed likely, that other interesting problem classes could be identified by considering single given domains of some other type (see discussion at the end of Section 2). However, we do not pursue this any further here. We look, instead, at what happens when there are two given domains outside the machine.

As for the single domain case, other interesting classes of problems emerge. In fact, the remaining four basic problem frames introduced in [10] are all of this form.

If we add a biddable domain to the Requirement Behaviour Frame we obtain a User Commanded Behaviour Frame, which is illustrated in Figure 12. Its characterising problem is that of building a machine that will accept the user's commands, impose control on some part of the physical world accordingly, and provide suitable feedback to the user[2].

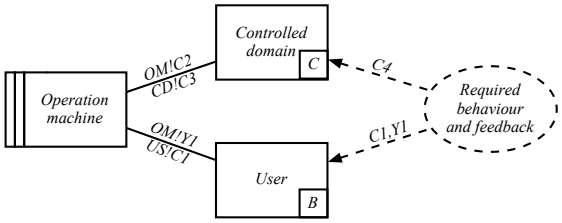


Figure 12: User Commanded Behaviour Frame

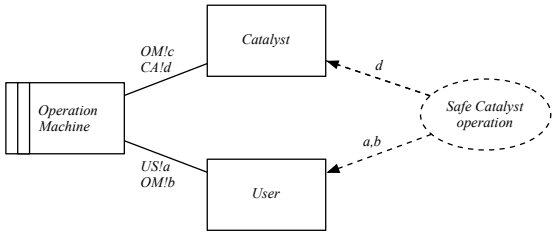
In the chemical reactor problem, we can apply the User Commanded Behaviour Frame to analyse how the catalyst is controlled by the operator. The corresponding problem diagram is given in Figure 13.

A possible description of the interaction rules, could be as follows. The machine shall allow the user to control the catalyst under the following constraints:

1. *catalyst_status* is a faithful representation of the state of the catalyst
2. the initial state of the catalyst is *catalyst_closed*
3. possible user commands are *open_catalyst* or *close_catalyst*
4. state transitions are represented in Figure 14.

The frame concern for the User Commanded Behaviour Frame is given in Figure 15. From the figure you will notice that the argument has two parts: satisfying the required behaviour of the domain (from 1 to 4); and providing suitable feedback to the user (5 and 6).

[2][10] introduces a subclass of this frame, the Commanded Behaviour Frame, which does not require the user to receive any feedback.



$a : \{open\_catalyst, close\_catalyst\}$
$b : \{catalyst\_status\}$
$c : \{open\_catalyst, close\_catalyst\}$
$d : \{is\_open, is\_closed\}$

Figure 13: Controlling the catalyst as an instance of a user commanded behaviour problem

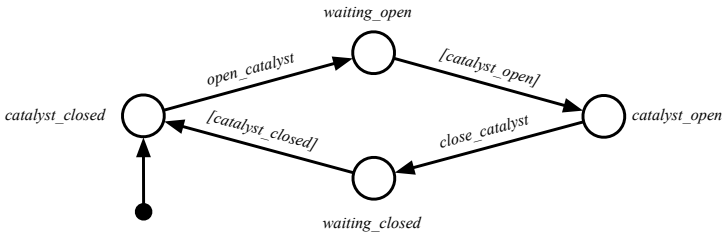
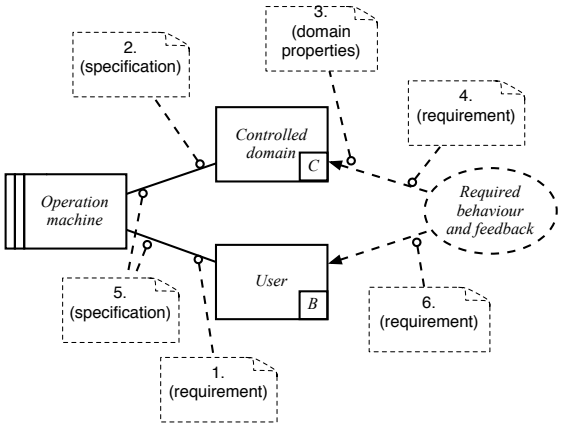


Figure 14: State machine model for the catalyst

4 Problem Decomposition

Most real problems are too complex to fit basic problem frames. They require, rather, the structuring of the problem as a collection of (interacting) sub-problems. In this section, we discuss two ways of decomposing problems within the PF framework. The first, classical decomposition, proceeds through sub-problem identification and problem frame instantiation. The second, our novel approach, combines sub-problem identification with guided architectural decomposition using AFrames.



1 Given a choice of commands in the current state, when the user issues this command (it may or may not be sensible)..

2 if sensible or viable the machine will cause these events...

3 resulting in this state or behaviour...

4 which satisfies the requirement...

5 and which the machine will relate to the user...

6 thus satisfying the requirement in every case.

Figure 15: The frame concern for the User Commanded Behaviour Frame

4.1 Classical decomposition

In classical PF decomposition, a problem is decomposed in simpler constituent sub-problems which can then be analysed separately. If necessary, each sub-problem can be decomposed further, and so forth, until only very simple sub-problems remain. Decomposition proceeds through the identification of sub-problems that fit a recognised problem class, and the instantiation of the corresponding problem frame. We illustrate the process on the chemical reactor problem.

There are three sub-problems:

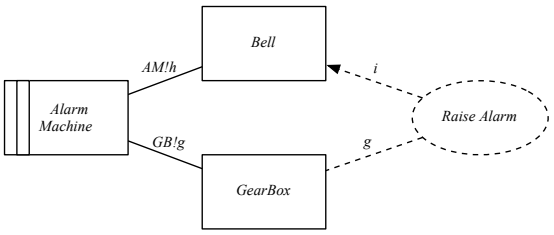
1. a user commanded behaviour problem, for the operator to control the catalyst;

2. a required behaviour problem, for regulating the water flow in the cooling system; and

3. a sub-problem to issue a warning (and halt the system) when there is an oil leak in the gearbox.

Addressing sub-problems 1. and 2. means instantiating the corresponding problem frames to derive problem diagrams for each subproblem. These are depicted in Figures 13 and 7, respectively.

The third sub-problem has no standard problem frame to represent it. The closest fit would be the Information Display Frame ([10]), but this requires a decision on how the alarm will be raised. Here, we have made an arbitrary choice of introducing a *Bell* domain, and assume that it will ring when the oil level in the gearbox is below a certain threshold. The resulting sub-problem diagram is shown in Figure 16.



$h : \{ring\_bell\} \quad i : \{bell\_ringing\} \quad g : \{oil\_level\}$

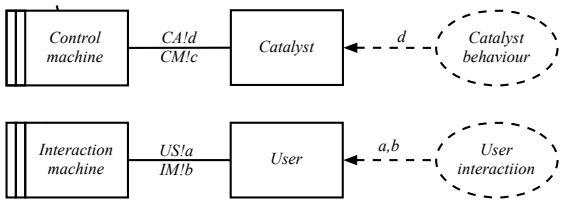
Figure 16: Raising the alarm as an instance of the information display problem

We already know from the simple taxonomy in Section 3 that the sub-problem 1. can be decomposed further resulting in a required behaviour and a user interaction sub-problem. These are shown in Figure 17.

4.2 AFrames

AFrame decomposition complements classical decomposition in providing guidance and decomposition rules. The rational behind AFrames is the recognition that solution structures can be usefully employed to inform problem analysis.

AFrames characterise the combination of a problem class and an architectural class. An AFrame should be regarded as a problem frame for which a 'standard' sub-problem decomposition (that implied by an architecture or architectural style) exists. AFrames are a practical tool for sub-problem decomposition that allow the PF practitioner to separate and address, in a systematic fashion, the concerns



$c : \{open\_catalyst, close\_catalyst\}$ $a : \{open\_catalyst, close\_catalyst\}$
$d : \{is\_open, is\_closed\}$ $b : \{catalyst\_status\}$

Figure 17: Further decomposition of the user Commanded Behaviour sub-problem

arising from the intertwining of problems and solutions, as has been observed to take place in industrial software development [12]. Further motivation for, and other examples of, AFrames can be found in [13].

Here we introduce the MVC AFrame as applied to the User Commanded Behaviour Frame. This represents the class of user commanded behaviour problems for which an MVC solution [1] is to be provided.

The intention of using the MVC in the solution space is recorded through an annotation of the machine as illustrated in Figure 18. Guidance on decomposition is in the form of decomposition templates, which are applied to obtain sub-problem diagrams. The decomposition templates for the MVC AFrame are given in Figure 19. It can be seen from the figure that the original problem is decomposable into two sub-problems, whose machine domains are the View and Controller machines (in the MVC sense). Also, a Model domain is introduced which represents an abstraction of the real-world domain to be controlled. This is a designed domain [10], i.e., one that we have the freedom to design, as it will reside inside the solution machine. The resulting sub-problems are then: that of building a View machine to display the Model's representation of the state of the controlled domain; and that of building a Controller machine that acts on the Model, which will pass on the commands to the controlled domain. In PF terms, the Model acts as a *connection domain* between the real-world domain and presentation and control subsystems.

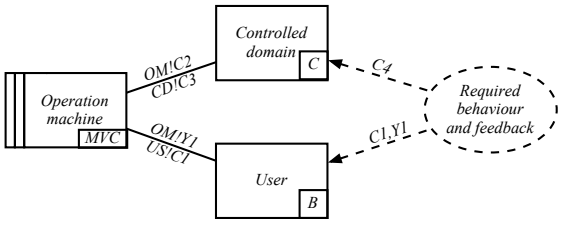


Figure 18: MVC annotation of the User Commanded Behaviour Frame

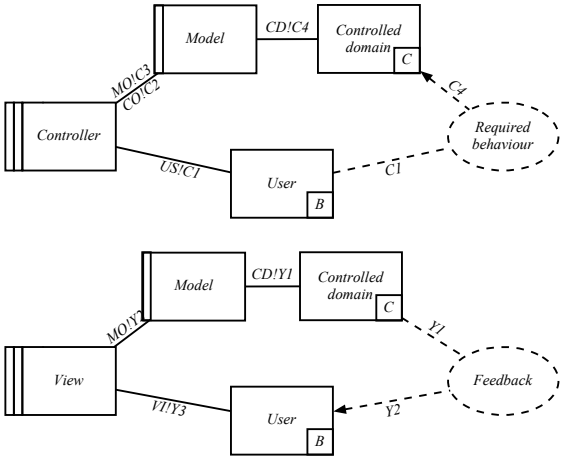
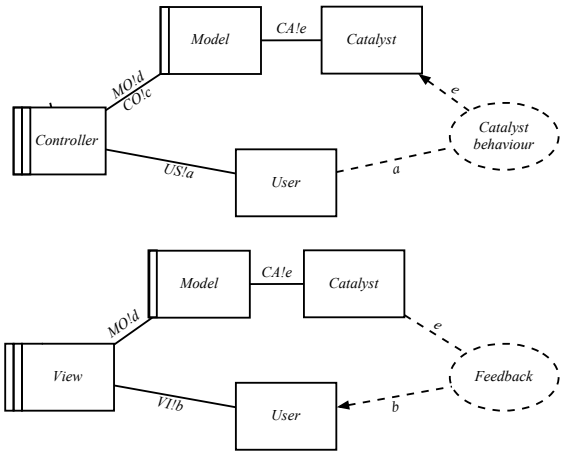


Figure 19: Decomposition templates for the MVC AFrame

The application of the MVC AFrame to the sub-problem of controlling the catalyst (see Figure 13) results in the decomposition of Figure 20.

We see at least two strengths of AFrames. The first is that they suggest how a problem would need to be restructured for a particular solution form. For instance, in the MVC case, that an abstract model of the catalyst needs to be produced (or, for that matter, a connection domain between Operator and Gearbox– a *Bell* – would be needed).

The second is that they help the recomposition of sub-problem solutions into the original problem. Recomposition is facilitated by the fact that AFrame decomposition is regularized through the application of the AFrame templates. For the MVC, this is through the identification of the links among its architectural elements. The recomposition diagram for the MVC AFrame is illustrated in Figure 21, and its frame concern in Figure 22.



$b : \{catalyst\_status\}$ $a : \{open\_catalyst, close\_catalyst\}$
$d : \{is\_open, is\_closed\}$ $c : \{open\_catalyst, close\_catalyst\}$
$e : \{open, closed\}$

Figure 20: MVC decomposition of the user commanded behaviour sub-problem

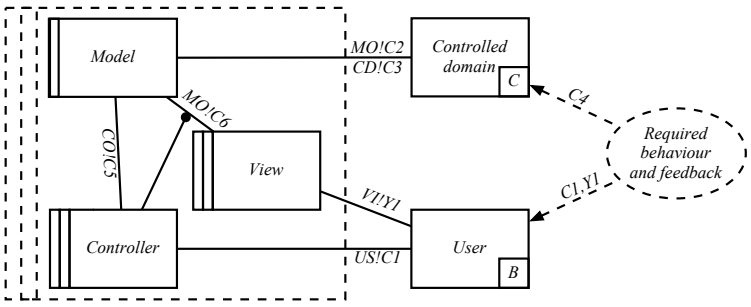
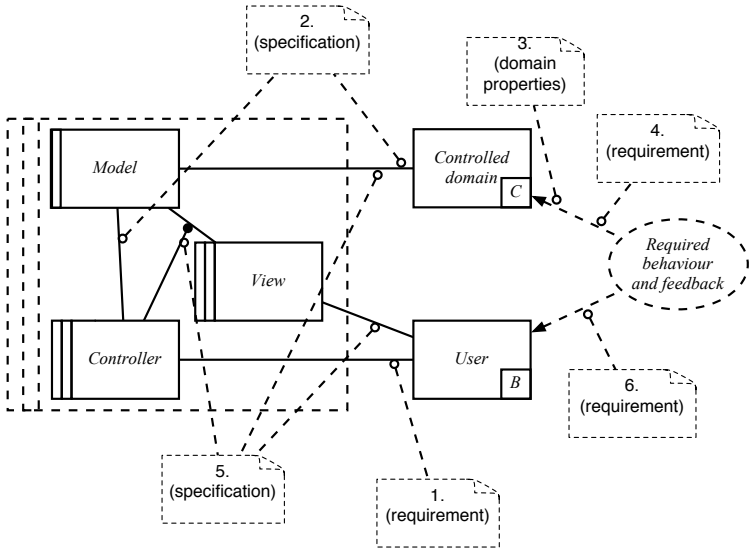


Figure 21: MVC recomposition



1 Given a choice of commands in the current state, when the user issues this command (it may or may not be sensible)..

2 if sensible or viable the machine will cause these events...

3 resulting in this state or behaviour...

4 which satisfies the requirement...

5 and which the machine will relate to the user...

6 thus satisfying the requirement in every case.

Figure 22: Discharging the correctness argument in MVC recomposition

5 A Requirements Analysis Model for Socio-Technical Systems

The consideration of more sophisticated human-machine relationships is our next concern. To be specific, we now wish to look at users' behaviour as being the subject of requirements statements, admitting that users are the source of much of the flexibility in socio-technical systems, and can compensate for some of the inability of machines to adapt. In short, we wish to allow the design of human instruction to be the subject of the requirements engineering process addressed through problem frames alongside that of the program.

Paraphrasing this, we might say that the human, as well as the machine, is to be the subject of optative descriptions, i.e., that their flexibility will lead to the discharging of the requirements. Foundationally, this means the separation of the description of the world from that of the human that is the subject of the design. This leads to the reification of the original ellipse model shown in Figure 23. In it we have three ellipses – that for the Human H with knowledge K, that for the Machine M with program P, and that for the remaining Environment W with requirements R. Of course, just as machines outside the design process have indicative descriptions in W, so do humans.

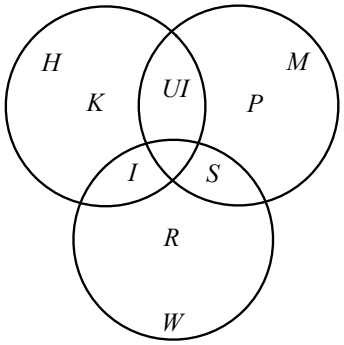


Figure 23: The extended requirements analysis model

With the introduction of the human H, we identify and separate two new areas of interest, which now form explicit foci for design:

- the specification UI, anonymous in the S region in the original model, which determines the Human-Machine interface; and
- the specification I, missing from the original model, which determines the knowledge and behaviour that is expected of the human as a component of the socio-technical system.

As in the original model the description W has the role of bridging the gaps between the requirement R and the specification S, in our extension W has the role of bridging the gaps between the requirement R and the instruction I, human-machine interface UI and specification S together. More precisely we assert that S, I, UI and W must be sufficient to guarantee that the requirements of the socio-technical system are satisfied. More formally:

$$W, S, I, UI \vdash R$$

5.1 A Problem Frame interpretation

In the PFs framework, the machine domain represents the machine for which the specification *S* must be designed. By analogy, a new domain type will be required to represent the human for which the instruction *I* has to be designed. To this end, we introduce into the PF Framework the notion of a *knowledge domain* to represent that domain. In a problem diagram, a knowledge domain should be represented as a domain box with double bar on the right-hand side (to distinguish it from the machine domain).

The most general form of a socio-technical problem, as a problem diagram, is shown in Figure 24. In the figure, both Knowledge and Machine domains are subjects of design, as are their shared user interface phenomena.

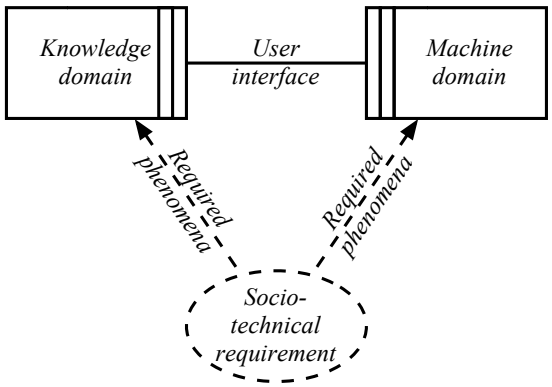


Figure 24: The general 'socio-technical problem' diagram

An example of how a real-world socio-technical problem could be treated in the new model is that of designing a 'cold-calling' system to allow an interviewee to be telephoned, and for their responses to certain questions, asked by an interviewer, to be recorded in a database for future analysis. The problem is to design both the technical subsystem (the machine domain) *and* the instructions that guide the interaction of the interviewer (the knowledge domain) with the interviewee. The interviewee sits firmly in the environment (as a biddable, indicative domain). The interaction with the database is a machine task.

The outcome of the design process, in addition to the specification for the technical subsystem, might be a script for the interviewer, and the human-machine interface as used by the interviewer. The problem diagram for this example is outlined in Figure 25.

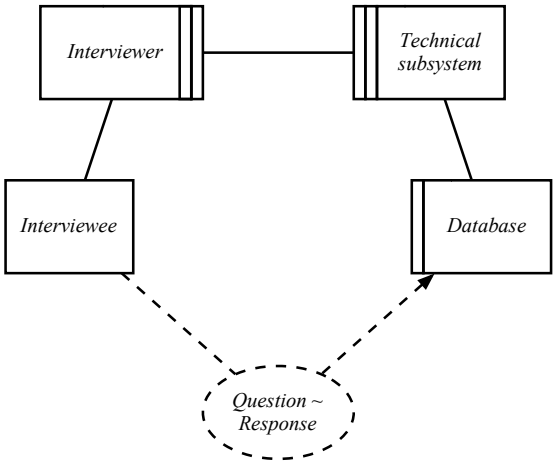


Figure 25: Outline problem diagram for the cold-calling system

6 Conclusions

In their classical form, problem frames happily represent interactions between a user and a machine, as might be characteristic of simple socio-technical systems. In this chapter, we have presented an enrichment of the PF framework to allow the representation and analysis of more complex socio-technical systems. To do this we have introduced two new problem frames, the User Interaction and User Commanded Behaviour Frames. Although not exhaustive in their treatment of socio-technological interaction problems, we hope that they will provide a sound basis for a richer taxonomy of user interaction within the PF framework.

One of the, as yet, under-developed areas within the PF framework is the treatment of problem decomposition, in particular, from the perspective of how to do it in practice. We are currently exploring the development and use of AFrames. An AFrame offers guidance in problem decomposition on the basis of solution space structures. In this chapter, we have shown how basic socio-technical interaction problems can be decomposed when the target architecture is to be the MVC style.

Although both these enrichments are new in the PF framework, they do not move outside of its original conceptual basis in the two ellipse model of requirements analysis. In contrast, we have seen in this chapter that the development of general socio-technical systems raises challenges for the PF framework. We have suggested solutions to these challenges in the reification of the two ellipse model to a three ellipse version, in which social sub-systems – individuals, groups, and organisations – can also be considered as the focus of the design process. With

the introduction of the knowledge domain, the manifestation of this extension in problem frames, we aim to bring the socio-technical system problems under the design remit of the PF framework.

7 Acknowledgements

We acknowledge the kind support of our colleagues, especially Michael Jackson and Bashar Nuseibeh in the Department of Computing, the Open University.

References

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison Wesley, 1998.

[2] O. Dieste and A. Silva. Requirements: Closing the gap between domain and computing knowledge. In *Proceedings of SCI2000, Vol. II (Information Systems Development)*, 2000.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1995.

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

[5] Jon G. Hall and Lucia Rapanotti. A Reference Model for Requirements *Engineering*. 2003/6, Department of Computing, The Open University, 2003.

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

[7] M. Jackson. *Principles of Program Design*. Academic Press, 1997.

[8] M. A. Jackson. Problem analysis using small problem frames. *South African Computer Journal 22; Special Issue on WOFACS98*, pages 47–60, 1998.

[9] M. A. Jackson. Some basic tenets of description. *Software and Systems Modeling*, 1(1):59, 2002.

[10] Michael Jackson. *Problem Frames*. Addison-Wesley, 2001.

[11] N. Leveson. Software safety: What, why and how. *ACM Computing Surveys*, 18(2), 1986.

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

[13] Lucia Rapanotti, Jon G. Hall, Michael Jackson, and Bashar Nuseibeh. Architecture-driven problem decomposition. Technical Report TR 2003/??, Computing Department, The Open University, 2003.

[14] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, January 1997.