Developer Requirements in the PF Approach

T. T. Tun
Open University
Centre for Research in Computing
Milton Keynes
MK7 6AA
t.t.tun@open.ac.uk

J. G. Hall
Open University
Centre for Research in Computing
Milton Keynes
MK7 6AA
j.g.hall@open.ac.uk

ABSTRACT
This paper considers the interplay between the requirements of two key stakeholder groups, users and developers, in an extended Problem Frames approach. It is already known that user requirements usually constrain the behaviour of a solution, whilst the solution structure is largely determined by the requirements of, among others, developers who subsequently implement the system. Since Jackson's Problem Frames are restricted to the problem domain, in this extended Problem Frames framework, we demonstrate how considerations for developer requirements in the solution domain may influence the problem structure in terms of division of responsibilities between physical domains and selection of a subproblem recomposition operator.

Categories and Subject Descriptors
D.2.1 [**Requirements/Specifications**].

General Terms
Design.

Keywords
Problem Frame, Requirements Engineering, Subproblem Recomposition, Feature, Feature-based Systems.

1. INTRODUCTION
Requirements for modern software applications are often expressed in terms of "features", which are "coherent and identifiable bundle[s] of system functionality that help [characterise] the system from the user perspective." Feature-driven development is an approach in which features become "an organizational mechanism that can structure important relationships across life-cycle artefacts and life-cycle activities" [1]. Feature-based systems are often based on flexible and extensible software architectures, so that components for new features can be added, and components for existing features can be upgraded or removed with limited knock-on effects.

Development of any software system begins with some form of analysis of the requirements for the system. We recognise that requirements can be divided into several categories on the basis of stakeholders. The most important requirements are those of users, which generally relate to the behavioural properties of the system. Developers also tend to have important requirements relating to the system development process and system structure. In this scheme of things, there are often competing solutions that satisfy a set of user requirements, and developers are often presented with albeit narrow opportunity for choosing a design that best satisfies their requirements. For example, given a set of user requirements, there may be two solutions that satisfy the requirements; one solution based on a reuse architecture, another based on an extensible architecture. Developers in the component-based development setting may opt for the first design, whilst those in the feature-driven development setting may prefer the second design. In a realistic situation, a number of solutions based on various architectural properties may exist, and tradeoffs are often made between architectural properties in choosing a particular design [2].

IWAAPF'06, May 23, 2006, Shanghai, China.

2. PROBLEM FRAMES
The Problem Frame approach (PF) provides a rigorous intellectual framework for analysing requirements and their contexts, and producing specifications for the system that satisfies stated requirements. Decomposing complex problems using PF involves identification of physical domains and shared phenomena between them. Physical domains in Problem Frame diagrams are generally categorised into three groups: machine domains, designed domains and given domains. Designed and given domains can be further characterised as causal, lexical and biddable [3, 4]. Properties and interface of given domains are known and cannot be modified, whilst properties and interface of designed domains are not known and therefore they can be constructed with much freedom.

Because of their emphasis on the problem, the choices for development in the original Problem Frame framework are limited to those that apply in the problem domain. For instance, although problem decomposition through projection exposes sub-problems that can be individually solved, there is no explicit recognition that the choice of solution structure can influence problem development, as has been acknowledged elsewhere, [5, 6]. This has recently been recognised in Problem Frames, with the precise nature of how this could be done being the subject of lively debate, [7, 8, 9, 10]. Without access to the solution domain it will be difficult for us to represent the choices that exist there, and so we use Hall et al's representations of architectures in this paper [9].

3. USER REQUIREMENT I

Consider a user requirement for a feature, which states that a system is needed to accept commands from a user and cause appropriate intended effects in a designed lexical domain.

This requirement fits the basic frame Workpiece [3], and let us call this requirement Correct Effects. In Figure 1, the shared phenomena c, among other things, has a degree of abstraction which allows us to be rather unspecific about the precise nature of the shared phenomena.

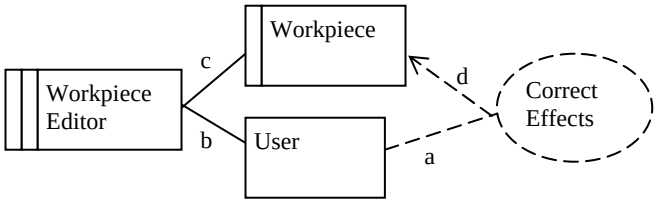

Figure 1. Problem diagram for Correct Effects.

4. DEVELOPER REQUIREMENT I

When discharging the frame concern to construct the correctness argument, the level of abstraction has to be reduced, and the domain properties and the interface, or domain responsibilities, of the designed domain have to be made known. If the domain responsibilities are known and unchangeable, all there is to do is think about what the machine must do to achieve the desired effects in the designed domain. Alternatively, if the machine specification is known and unchangeable, we can also design the domain in such a way that the intended effects are also achieved. In this example, neither is the case. This problem is rather similar to the codesign problem [11] and responsibility sharing of classes in Object-Oriented design [12, 13].

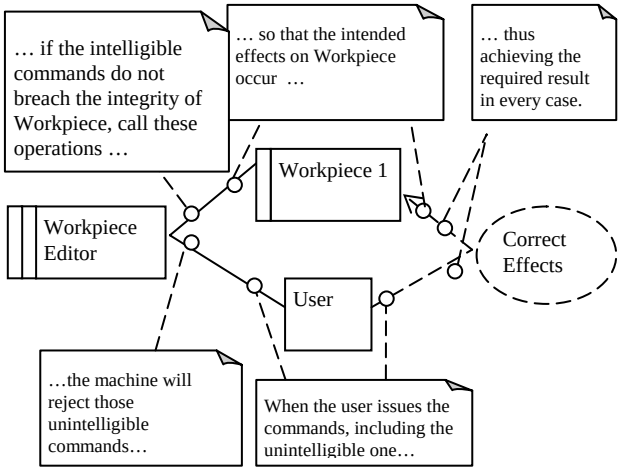

Figure 2. 'Smart' machine with 'dumb' Workpiece

Since the designed domain does not yet exist, we can make various legitimate assumptions about the nature of the lexical domain here. At one end of the spectrum of distributing responsibilities, we can treat the Workpiece domain as essentially 'dumb', and specify the machine with the assumption that it is the responsibility of the machine to maintain the integrity of the Workpiece domain, as shown in Figure 2. At the other end of the spectrum, we may discharge the concern in such a way that the designed domain is regarded as 'smart,' i.e., the domain can carry out certain operations itself, whilst the role of the machine is reduced to a simple connector, Figure 3.

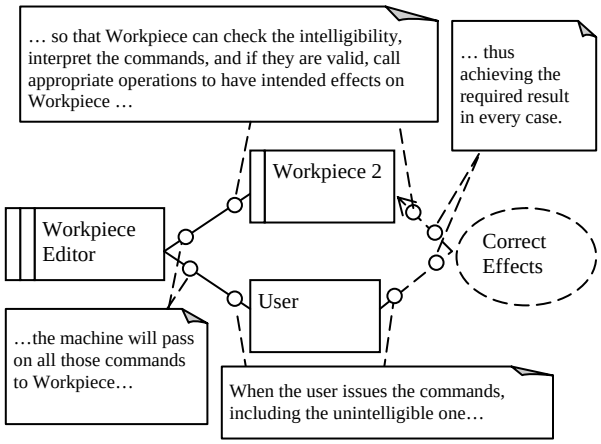

Figure 3. 'Dumb' machine with 'smart' Workpiece

The key difference between the way the frame concern is discharged in Figures 2 and 3 is that in the first case, much of the responsibility is given to the machine, whilst in the second case, it is allocated to the designed lexical domain. There are, of course, other options between the two extremes. It is, however, important to note that the same requirement is satisfied in each case. The question is: Which specifications should developers write?

In this situation, we may consider developer requirements, which provide some constraints on the choices. For example, development with reuse, rather than development from scratch, may want a specification of fairly dumb machine, which will allow reuse of a Workpiece that is not dumb.

5. REQUIREMENT DECOMPOSITION

If the requirement Correct Effects is examined closely, we will find it to be an aggregate requirement, which may be decomposed as follows.

The system should: a) examine the commands issued by the user, and reject those commands that are unintelligible, b) translate the intelligible user commands to the commands (e.g. names of operation to be called) understood by the lexical domain, and c) examine the resulting domain commands and reject those commands that will render the lexical domain into an undesirable or unknown state, and issue the valid commands to the lexical domain (or call operations of the lexical domain) so that its properties are modified accordingly.

Each of these can be considered a subproblem of a larger problem, and they can be projected as shown in Figure 4.

6. SUBPROBLEM RECOMPOSITION

After decomposing the problem, we can now consider recomposition options in the solution domain.

One implication of the assumption in Figure 2 is that when we recompose the submachines in Figure 4, they have to be kept together somehow in a single unit. For example, we may create a monolithic machine subsuming the submachines, which may not be desirable for a number of reasons, including limited reuse opportunity. Instead, using a controller operator, a new controller machine can be introduced here [14]. Requirement for the controller is to co-ordinate and constrain interactions between the three submachines. This centralised approach may often starve other domains of some of their responsibilities, and concentrates them in the controller component.

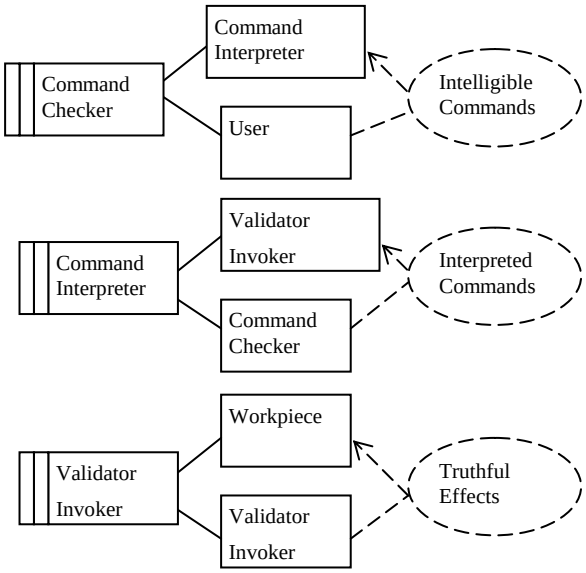

Figure 4. A projection of subproblems of Correct Effects

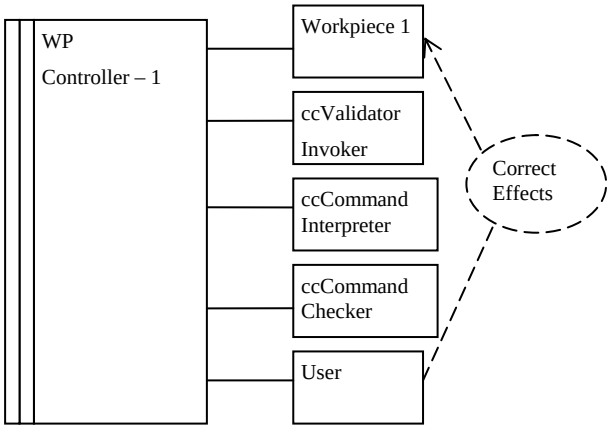

Figure 5. Pluggable architecture with a central controller.

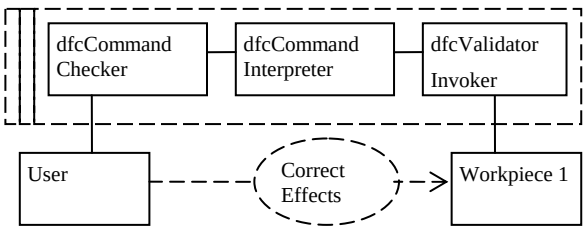

Figure 6. Distributed linear architecture for Correct Effects

Instead of such a highly centralised composition of submachines, we may consider using a decentralised recomposition operator. The recomposition in Figure 3, describes a distributed linear recomposition style, loosely based on pipe and filer and the DFC architecture [15], the latter also emphasises feature-oriented decomposition.

It is stressed again that the composite behaviour of the domains in Figure 5 and Figure 6 are identical, although the static structures of the machine domains are rather different.

7. USER REQUIREMENT II

Let us now assume that the user of this initial system wishes to have another feature in this system. The requirement for this new feature states that the user wants to have the information about the length of time the user has been working on the workpiece, inserted into the domain at a timely interval, Figure [7].

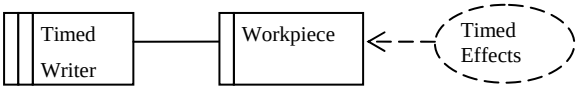

Figure 7. Requirement for a new feature.

8. FURTHER RECOMPOSITION

In this case, commands for Workpiece are generated by a causal submachine of TW, named CG (Commands Generator), and the commands it generates are intelligible and interpreted commands that Workpiece understands. The TW machine still needs to ensure that it does not breach the integrity of Workpiece; that is to say, TW has its own VI component. The two submachines of the TW can be recomposed using a controller, which in turn can be further recomposed with WC1, as shown in Figure 8.

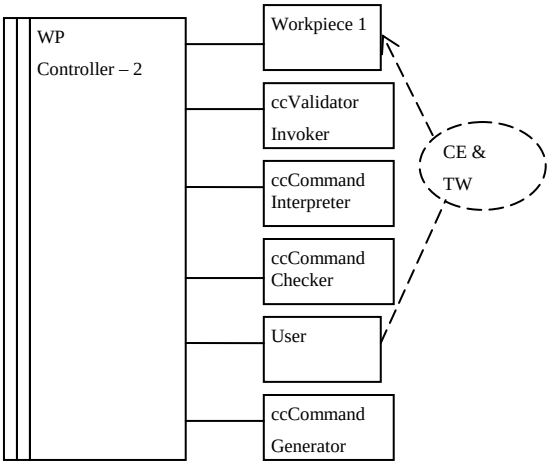

Figure 8. Pluggable architecture with new component.

The centralised architecture allows the reuse of VI component by TW, and composition constraints such as non-interference and synchronisation have to be satisfied by the controller. Therefore, WC2 needs to be built, probably based on WC1.

If we add the CG component to the decentralised architecture suggested in Figure 6, the VI component is also reused. However, being the single port of communication with Workpiece, it needs to be redesigned so that the composition constraints are also

satisfied, Figure 9. For example, mfcVI could be made atomic and singleton to enforce exclusion [16].

Again, the behavioural requirements of the system, i.e. the requirements of the user, are satisfied in both cases in Figure 7 and Figure 8. The structural properties of the domains in the two diagrams, however, are different.

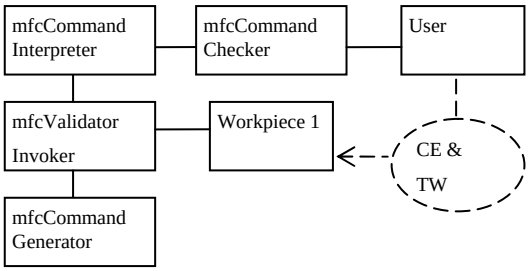

Figure 9. Distributed linear architecture with new component.

9. DEVELOPER REQUIREMENT II

When presented with two solutions that produce identical behaviour, but with different structures, developer requirements may be taken into account in choosing a specific solution. In fact, in some cases, these requirements may compel the developers to choose a certain option. Here tradeoffs between architectural properties are inevitable.

In this case, we can reason that, an advantage of the centralised architecture in Figure 8 is that of reuse; all basic components are readily reusable when components for new features are added. A possible downside of the architecture is that of scalability; when components for new features are added, the complexity of the controller is substantially increased. Therefore, this architecture may be suitable for component-based development, in which a predictable number of components need to be assembled.

An advantage of the decentralised architecture in Figure 9 is that of extensibility; this architecture allows addition of new components to the system, the effect of which can be localised. A weakness of this architecture is that of reuse: high level of component coupling may make the components less reusable than those in the centralised architecture. This architecture is perhaps better suited for feature-driven development.

10. CONCLUSION

In this paper we have argued that the needs of the developers are a force worthy of consideration in the Problem Frames framework. We have shown that such need stem from a) the tools available to the developers – e.g., the existence of re-usable artefacts, such as smart Workpieces – and b) the processes into which the developers' work must fit – e.g., the need for the feature-driven development of a product whose market lifetime will be measured in years, not months. The former we will call *point* needs, the latter we will call *longitudinal* needs. In an extended problem frame framework, we have observed that point needs lead to decisions of how to assign responsibilities to domains, whereas longitudinal needs lead to questions of the most appropriate solution architecture.

If we see developers' needs as non-functional, or quality, requirements, then our observations may be seen as the beginning of the reflection of the work of Bass et al in [2] into the problem frames setting. We consider such a combination to be valuable to the extended Problem Frames.

11. ACKNOWLEDGMENT

We wish to thank our colleagues at The Open University, in particular Michael Jackson, Robin Laney, Bashar Nuseibeh and Lucia Rapanotti for valuable comments.

12. REFERENCES

[1] C. Reid Turner, A. Fuggetta, L. Lavazza and A. L. Wolf, "A conceptual basis for feature engineering", *Journal of Systems and Software*, vol. 49, no. 1, pp. 3-15, 1999.

[2] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.

[3] M. Jackson, *Problem Frames*, ACM Press Books & Addison Wesley, 2001.

[4] M. Jackson, *A Tutorial on Problem Frames Day 1*, Milton Keynes: Open University, December 2005.

[5] W. Swartout, and R. Balzer, "On the inevitable intertwining of specification and implementation", *Communications of the ACM*, 7(25), pp. 438-440, 1982.

[6] B. Nuseibeh, "Weaving together requirements and architectures", *Computer*, 3(34), pp. 115-119, 2001.

[7] C. Choppy, and G. A. Reggio, "A UML-Based Method for the Commanded Behaviour Frame", in *Proc. IWAAPF04*, Karl Cox and Jon G. Hall and Lucia Rapanotti Eds., IEE , pp. 27-34, 2004.

[8] L. Lavazza, and V. A. Del Bianco, "UML-based Approach for Representing Problem Frames", in *Proc. IWAAPF04*, Karl Cox and Jon G. Hall and Lucia Rapanotti Eds., IEE , pp. 39-48, 2004.

[9] J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh and L. Rapanotti, "Relating Software Requirements and Architectures Using Problem Frames", in *Proc.10th Anniversary IEEE Joint International Conference on Requirements Engineering (RE 2002)*, pp. 137-144.

[10] L. Rapanotti, J. G. Hall, M. Jackson and B. Nuseibeh, "Architecture-driven Problem Decomposition", in *Proc. 12th IEEE International Conference on Requirements Engineering (RE 2004)*, pp. 80-89, 2004.

[11] D. W. Franke and M.. K. Purvis, "Hardware/Software CoDesign: A Perspective", in *Proc. 13th International Conference on Software Engineering*, pp. 344–352, 1991.

[12] R. Wirfs-Brock and B. Wilkerson, "Object-Oriented Design: A Responsibility-Driven Approach", *SIGPLAN Notices*, vol. 24, no. 10, pp.71-75, 1989.

[13] D. Bellin and S. S. Simone, *The CRC Card Book*, Addison-Wesley Object Technology Series, 1997.

[14] R. Laney, L. Barroca, M. Jackson and B. Nuseibeh, "Composing Requirements Using Problem Frames", *Proc. 12th IEEE International Requirements Engineering Conference (RE'04)*, pp.122-131, 2004.

[15] P. Zave and M. Jackson, A Call Abstraction for Component Coordination, in Proc. ICALP2002, 2002.

[16] Gamma et al, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.