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].

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 subproblems 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.

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].

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 operations 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.

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].

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.
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.

An advantage of the decentralised architecture in Figure 9 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.

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