Composing Requirements Using Problem Frames

Robin Laney, Leonor Barroca, Michael Jackson, Bashar Nuseibeh
Dept. of Computing,
The Open University,
Walton Hall, Milton Keynes, MK7 6AA, UK
{R.C.Laney, L.Barroca, M.Jackson, B.A.Nuseibeh}@open.ac.uk

Abstract

Problem Frames are a systematic approach to the decomposition of problems that allows us to relate requirements, domain properties, and machine specifications. Having decomposed a problem, one approach to solving it is through a process of composing solutions to sub-problems. In this paper, we contribute to supporting such a process by providing a way to compose multiple Problem Frames. We develop a systematic approach to composing inconsistent requirements. We introduce Composition Frames, a requirements construct that models relevant aspects of composition and thus deals with unwanted effects, such as interference of overlapping reactions to events. Throughout the paper we use a simple case study to illustrate and validate our ideas.

1. Introduction

Problem Frames are an approach to the decomposition of problems [17]. Given a good decomposition of a problem into sub-problems, a range of benefits would arise if we were able to provide a solution by solving the sub-problems in isolation and then composing the partial solutions to give a complete system [25, 26, 5]. The benefits include: scalability (due to working at the level of simpler sub-problems), traceability (since sub-problems map directly to their solutions), and easier system evolution (changes to sub-problems can be addressed by modifying corresponding solutions or compositions).

The composition problem is important, since solving it supports a better separation of concerns between requirements analysis and the design phase. The ability to compose solutions allows us to take a set of decomposed requirements, provide individual solutions to discrete requirements and then address the overall system requirements by recomposing solutions. This is consistent with an iterative approach to development [21,14].

The composition problem involves a number of difficult areas: Are the requirements to be composed consistent? Do the specifications to be composed share assumptions about their environment? Do they embody consistent models? How do we deal with interference between the effects of machines on the problem domain? We focus on the first and last of these questions, but in doing so address the others to varying degrees.

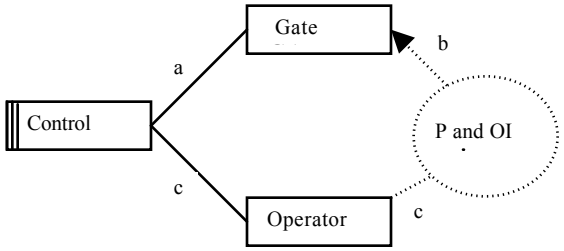
The contribution of this paper is to show how to address the composition problem for inconsistent requirements. We propose ways to weaken the conjoined requirements, resolving any inconsistencies, in a manner that allows us to satisfy them. We introduce *Composition Frames,* to express the *requirements* of composition, allowing us to provide arguments showing their satisfaction. Composition Frames model composition in order to deal with unwanted effects: that is interference of overlapping reactions to events. Composition Frames include the specification of a Composition Controller. In solution space terms these correspond to the notion of an architectural connector [1]. This allows us to move backwards and forwards between architectural and requirements perspectives using the Composition Frame as a reasoning tool. Thus trade-offs can be made between design and analysis issues.

The paper is organized as follow. In section 2 we present a simple problem decomposition whilst giving a brief introduction to Problem Frames. In section 3 we present our approach to the composition problem. We begin by considering the semantics of requirements composition, then introduce Composition Frames as a way of reasoning about the relationship between composed requirements and composition specifications. We give specifications for some example compositions. In section 4 we compare our work with other approaches. Section 5 is a discussion drawing some lessons about the composition of requirements, the composition of solutions, and their

relationship. We conclude in section 6 and present future work.

2. Introductory Example

Throughout this paper we will use an example that involves specifying the requirements for the control of a sluice gate. The example is based on material in [17] which explores a number of further aspects of the problem that do not concern us here. The sluice gate has a motor that can be controlled by pulse events (Clockw, Anti, On, Off) to move it up or down, and a sensor that indicates when it is fully open or shut by generating events (Top, Bottom). The problem is to control the sluice gate by moving it up and down subject both to pre-programmed control (P) that ensures it is open for 10 minutes in each three-hour period whilst also allowing for intervention by a human operator (OI). The human operator may wish to raise, lower, or stop the gate. The problem is represented in the problem diagram in figure 1.



a: C!{Clockw,Anti, On,Off} b:G{Open,Shut}
G!{Top,Bottom}, c:O!{Raise,Lower,Stop}

Figure 1. Problem Diagram

The diagram shows us the relationship between a *machine domain* (Control) that implements a solution, *problem domains,* that is entities in the world that the machine must interact with (Gate and Operator), and the *problem requirements* (the dotted oval). The requirements are to be met by providing a suitable machine. The machine domain (indicated by double lines on its left side) is a domain that is to be designed and for which we will provide a specification. The lines between domains labeled a, b, c represent interfaces between those domains i.e. shared phenomena by which they interact. In the example, the phenomena are given as sets of events. The prefixes, such as C!, show which domain controls a set of events (in this case the machine Control). Ultimately, we wish to show that given a particular machine specification the requirements are met; this requires us to have appropriate descriptions of the domains. In the above diagram we show that requirements P and OI are to constrain the operation of the gate (the dotted arrow), they reference the behaviour of the operator (the dotted line). We will work with the following specification of P and OI.

P – The gate should be opened for the first ten minutes of each three-hour period

OI – The gate should respond to Raise, Lower and Stop commands issued by the operator.

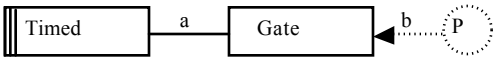
It is clear that there is some potential inconsistency between the requirements P and OI. For example, what if the Operator wants to raise the gate when the Timed machine is lowering it? This inconsistency is an example of "divergence" [19]. In section 3.1 we propose ways to weaken the conjoined requirements, resolving any inconsistencies, in a manner that allows us to satisfy them. Our two domains are described as follows:

Gate: The Gate, when stopped, will react to a <Clockw, On> event sequence by moving upwards, unless already fully open. When stopped, it will react to an <Anti, On> event sequence by moving downwards, unless already fully closed. When it receives an Off event it will turn its motor off, stopping any motion. When it detects that it is fully open it will generate a Top event. When it detects that it is fully closed it will generate a Bottom event.

Operator: The operator will generate Raise commands to request the gate moves upwards, Lower commands to request the gate moves downwards, and Stop commands to request the gate stops.

The domain descriptions given above, along with the interface topology, gives us a framework for checking whether a given machine specification meets our requirements. One immediate difficulty is how we arrive at that specification. The Problem Frames approach involves decomposing a Problem Frame by identifying sub-problems (projections of the original problem) that match well-known diagram forms, known as basic Problem Frames. These are problems in a form that are both relatively simple and well understood to the extent that we can expect to address them directly. In our example, we can decompose the problem into the pre-programmed and operator intervention sub-problems as shown in figures 2 and 3.

The Problem Frame for requirement P is of a well-known form called a *Required Behaviour Frame*, drawn from a catalogue of five basic Frames identified in [17].



a: T!{Clockw,Anti, On,Off} b:G{Open,Shut}
G!{Top,Bottom},

Figure 2. Problem Frame for Req. P

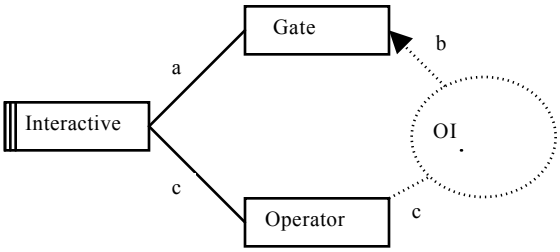
We postulate that the following specification for the machine called Timed, along with the above description of the Gate, is sufficient to establish that the requirement P is satisfied. The obligation to demonstrate that such a predicate holds is known as the *frame concern*, and the case for it holding needs to be made either formally or informally depending on context.

S_{Timed}:

At (time mod 180) = 0, start to open the gate (issue a <Clockw, On> event sequence).

At (time mod 180) = 10, start to close the gate (issue an <Anti, On> event sequence).

On receiving Top or Bottom events from the gate and motor, issue an Off event.



a: I!{Clockw,Anti, On,Off} b:G{Open,Shut}
G!{Top,Bottom}, c:O!{Raise,Lower,Stop}

Figure 3. Problem Diagram for Req. OI

The Problem Frame for requirement OI is similarly of a well-known form and is called a *Commanded Behaviour Frame*. Again, we postulate that given the above descriptions of the domains Gate and Operator, the following specification for the machine Interactive will ensure that the requirement OI is met.

$S_{Interactive}$:

On receiving an operator Raise command, issue a <Clockw, On> event sequence.

On receiving an operator Lower command, issue an < Anti, On> event sequence.

On receiving an operator Stop command, issue an Off event.

At this stage, we have taken our problem and decomposed it into two sub-problems. In itself this process is useful as it gives us a better understanding of the problem domain [17]. We have also provided solution specifications for the sub-problems. But how do we compose the two machines specified for Interactive and Timed, in order to produce a specification for the machine named Control in figure 1 that meets the frame concern of that diagram? To address this question for more than the simplest of disjoint requirements, we need to deal with the inconsistencies between requirements. We assume that the two machine specifications are likely to be composed in such a way that their implementations can be run on a single physical machine.

3. Composing Requirements

In this section we propose a way to deal with the issue of composing inconsistent requirements. In section 3.1 we present some systematic options for resolving inconsistencies. In section 3.2 we present a modeling construct – Composition Frames - allowing us to relate composed requirements to composed solutions. In sections 3.3 we illustrate how to specify a composition consistent with a Composition Frame.

3.1 Requirement Composition Options

In composing requirements, we need some way to address inconsistencies between them. There is a time dimension to this problem, in that there may be times when requirements are inconsistent and times when they are consistent [19]. From a Problem Frames perspective, inconsistencies will manifest themselves in terms of machine specifications producing interfering effects on problem domains. For example, the timed machine may try to raise the gate whilst the interactive machine is trying to lower it. However, in order to separate concerns between problem and solution spaces, we want to deal with requirements inconsistencies purely in requirements, rather than implementation terms. In this section we show how this can be done. We need to reason about interactions between our two machines and the problem domain in order to see if conflicts of control can arise. Given a machine M and a domain D, let us take the phrase, "D is currently reacting to M" to mean that M has generated an event that D is in the process of reacting to, i.e. the event has caused some activity in the domain which has yet to terminate. Note that we do not require the reaction to generate an event shared with either the original or any other machine. In the sluice gate system, for example, the sluice gate and its

motor will be currently reacting to a machine that has caused a clockwise or anticlockwise pulse, followed by an On pulse. This condition will cease once an Off pulse is received by the motor.

In general, we will need to analyse individual Problem Frames in order to establish what observable phenomena allow us to determine when a domain becomes under active machine control and when this ceases. We would expect this analysis to be amenable to at least partial automation.

In the absence of any knowledge about the internal structure of D, we must assume that any machine that generates a further event shared with D may interfere in some manner with the original reaction. In the sluice gate example we have a number of options:

1. Take no steps to prevent interference between operator and timed commands.
2. Only allow a machine to generate events shared with the Gate domain if that domain is not currently reacting to the other machine.
3. Establish priorities between the two machines, so that one of them is allowed to generate events shared with the Gate domain even though it is currently reacting to the other machine, but not vice versa.
4. As in 3, but attach priorities to the events themselves, i.e. work with smaller units of granularity.

In order to clearly state our composition requirements, we need to describe these options independently of solution-space concerns and in more general terms. To this end we propose the following ways of combining two general requirements R1 and R2. For the example above, R1 and R2 are the requirements for the timed gate and operator controlled gate respectively.

Option 1: No Control

Let R1 $\hat{}_{\{any\}}$ R2 be the requirement that R1 and R2 should each be met at times when they are not in conflict, but there is no requirement that any conflicts should be resolved, and if there are times when conflicts occur any emergent behaviour is acceptable.

Option 2: Exclusion

Let R1 $\hat{}_{\{active\}}$ R2 be the requirement that both R1 AND R2 should hold at all times except that when R1 leads to some activity in the world, R2 may be suspended and vice versa.

Option 3: Exclusion with Pre-emption

Let R1 $\hat{}_{\{R1\}}$ R2 be the requirement that both R1 and R2 should hold except that when R1 leads to some activity in the world, R2 may be suspended.

Option 4: Exclusion & Fine Grain Pre-emption

Let R1 $\hat{}_{\{important,\ R1\}}$ R2 be the requirement that R1 $\hat{}_{\{R1\}}$ R2 holds, except that any sub-requirement associated with the phenomenon *important* should be given top priority. For the sluice gate example, *important* might be instantiated as the operator Stop event.

Option 1 is unacceptable in the context of the sluice gate problem. For example, the domain properties given above do not tell us the effect of overlapping a raise command from the operator with the lowering of the gate by the timed machine. In general, however, there might be cases where interference between the behaviours specified by two requirements is allowed or even desirable. In this paper we will investigate Options 2 and 3, leaving Option 4 for further work, but noting that many of the principles developed here will also apply.

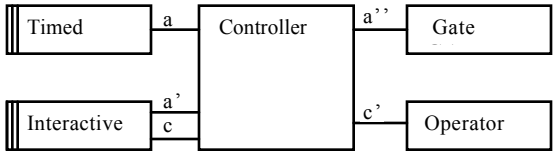
3.2 Composition Frames

We need to address the problem of composing two machines, each specified separately using Problem Frames, in order to address combined requirements. The solution we present below is to introduce a new form of Problem Frame: a Composition Frame. These frames include all of the domains from the two Problem Frames being combined, including the machine domains. Additionally we need a mechanism to resolve conflicts between requirements, which we develop below in the context of our running example. We have developed a single Composition Frame that can address either Option 2 or Option 3 above. We show below how the choice of a machine specification allows us to select between one of these two options.

We want our Composition Frame to clearly express the *requirements* of composition, allowing us to provide arguments showing their satisfaction. This requires us to overcome the tendency to address composition in design or implementation terms. Once again we start by making a strategic retreat: looking at some solution space concerns and then reverse engineering the requirements. In this we follow [14, 21] in recognizing that it is not always desirable or possible to ensure a complete separation of concern between problem and solution spaces.

Now consider the sluice gate example in terms of Option 2 and Option 3 from section 3.1. These options share some implementation concerns. We need to model whether the gate domain is reacting to a particular machine, and impose control over which machine(s) is allowed to interact with the gate on the basis of this model. These two concerns cannot be seen as belonging solely to the solution space. Both issues are tightly bound to our problem domain descriptions.

One possible diagram for specifying this form of composition is given in figure 4. Note that this is not a problem diagram: there is no requirement shown (and also there are two machines included). In this diagram, a Composition Controller is interposed between the machine and the world in order to control what events each side sees. In architectural terms, this corresponds to imposing a connector [1] between the machines and the interfaces to the Gate and Operator. The Controller will monitor events to build up a global view of what is happening and provide appropriate control. However this view is based on a premature jump into the solution space.

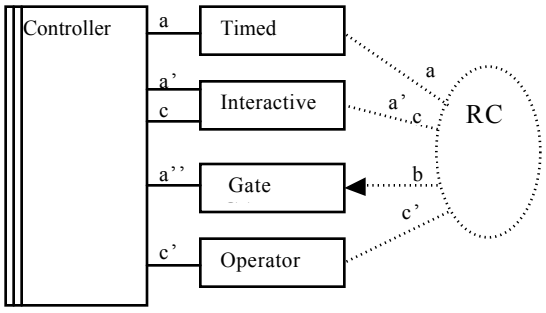


a: T!{Clockw,Anti, On,Off}
C!{Top,Bottom},
a'': C!{Clockw,Anti, On,Off}
G!{Top,Bottom},

a': I!{Clockw,Anti, On,Off}
C!{Top,Bottom},
c:C!{Raise,Lower,Stop}
c':O!{Raise,Lower,Stop}

Figure 4. Composition Diagram

In figure 5 we move back to a requirements perspective. Whilst the layout of this diagram is less suggestive of the idea of interposing the Controller (an implementation concern), it is more illustrative of the need to establish a frame concern (requirement) by arguing from a machine specification (Controller) via domain properties. Our original machines now become given problem domains. The diagram, which we will call a Composition Frame, has a requirement attached to it, denoted RC. The idea is that we can address either of the requirements options 2 and 3 of section 3.1 using machines that fulfill requirements R1 and R2 and a Controller meeting a suitable RC.



a: T!{Clockw,Anti, On,Off}
C!{Top,Bottom},
a'': C!{Clockw,Anti, On,Off}
G!{Top,Bottom},
b:G{Open,Shut}

a': I!{Clockw,Anti, On,Off}
C!{Top,Bottom},
c:C!{Raise,Lower,Stop}
c':O!{Raise,Lower,Stop}

Figure 5. Composition Problem Frame

We will parameterise the requirement RC with a protocol Pr_n, such that the frame can address either of Option 2 or Option 3 above. The protocol Pr_2 for Option 2 is that the original machines (Timed and Interactive) retain control of the gate once they gain it, until they voluntarily relinquish it. The protocol Pr_3 for Option 3 is that the Timed machine retains control of the gate but can forcibly grab control from the Interactive machine. Support for Option 4 can be given within a similar scheme, but to do so would complicate the presentation of our main ideas.

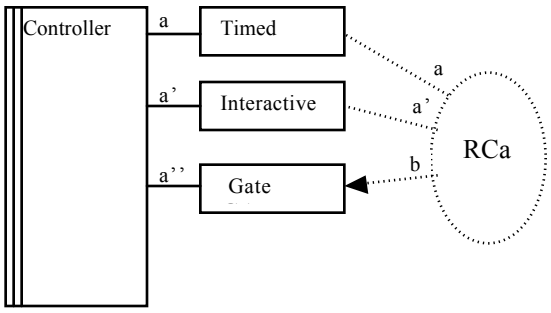
Requirement RC:

The requirement RC can be expressed as two sub-requirements:

> RCa: The Timed and Interactive Machines shall gain mutually exclusive control to the gate. This is done subject to a protocol Pr_n.
>
> RCb: Operator commands received whilst the gate is under timed control shall be rejected. Otherwise commands shall be passed to the Interactive machine.

Notice that the extent to which the requirement RC can be said to be expressed independently of phenomena at the machine interface is debatable. Nonetheless the requirement is expressed precisely. Furthermore, it will become clear that we can address the requirement RC using a decomposition that addresses RCa and RCb independently.

In figure 6 we see that requirement RCa is a constraint on the behaviour of the gate (it must only respond to a machine that has gained control according to Protocol Pr_n). The requirement involves references to the phenomena sets a, a' and b.

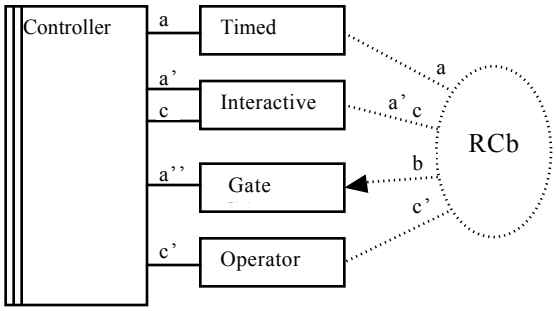


a: T!{Clockw,Anti, On,Off}
C!{Top,Bottom},
a'': C!{Clockw,Anti, On,Off}
G!{Top,Bottom},

a': I!{Clockw,Anti, On,Off}
C!{Top,Bottom},
b:G{Open,Shut}

Figure 6. Frame for Rca

In figure 7, the requirement RCb involves knowing whether the gate is actively under the control of the Timed machine. So we include all the phenomena references from figure 6. Furthermore, the requirement references the phenomena set c, representing the operator issuing commands, and constrains the behaviour of the gate according to whether they are accepted or ignored. Notice that the diagram has helped us to refine requirement RCb from its original expression in terms of which events get passed to the Interactive machine to its expression in terms of the Gate behaviour.



a: T!{Clockw,Anti, On,Off}
C!{Top,Bottom},
a'': C!{Clockw,Anti, On,Off}
G!{Top,Bottom},
b:G{Open,Shut}

a': I!{Clockw,Anti, On,Off}
C!{Top,Bottom},
c:C!{Raise,Lower,Stop}
c':O!{Raise,Lower,Stop}

Figure 7: Frame for RCb

3.3 Controller Specification and Frame Concerns

In sections 3.1 and 3.2 we showed how to specify composition requirements and relate them to machine specifications addressing sub-problems. We now need to show how to provide specifications for the new machine introduced by a Composition Frame. Furthermore, in order to provide a framework for verifying composition correctness we need to state a frame concern [17]: the shape of an argument allowing us to show the composition specification meets the composition requirements. Below, we show how to address these problems using our running example. We first deal with Option 2, and then show that Option 3 can be dealt with by extending the specification for the machine in Option 2.

Now consider the frames introduced in section 3.2, in respect of Option 2. We need to clarify the frame concerns for figures 6 and 7 in relation to the requirements (P $\hat{}_{\{active\}}$ OI). The frame concern for figure 6 is that the Timed or Interactive machines do not generate events that cause a reaction in the gate if the gate is currently reacting to the other machine. The frame concern for figure 7 is that operator commands be rejected when the gate is currently reacting to the Timed machine. Since both of these concerns require a machine that models the status of the gate it is sensible to provide one specification for the Controller that meets both frame concerns. As a consequence of doing so, we avoid introducing a second composition problem!

The controller can be viewed as a state machine that must be able to deal with three states: neither machine considered active; the Timed machine considered active; and the Interactive machine considered active. In the first case we are concerned with monitoring the Timed and Interactive machine to detect that they wish to be considered active. In the latter two cases we need to pass events from one of the machines to the gate and ignore the other machine. We also need to detect when the machine considered active can be treated as passive. When the interactive machine is active we need to pass events from the operator to the gate. An operational specification based on such a state machine can be provided systematically.

```
timedMActive = False
interactiveMActive = False
LOOP
  WHILE NOT (timedMActive OR interactiveMActive)
    wait and accept an event
    CASE event OF
      T:event:          pass event to Gate;
                        timedMActive = True;
      O:event:          pass event to Interactive;
                        interactiveMActive = True;
    ENDC
  ENDLOOP
  WHILE timedMActive
    wait and accept an event
    CASE event OF
      T:Off:            pass event to Gate;
                        timedMActive = False;
      T:event:          pass event to Gate;
      O:event:          reject event;
    ENDC
  ENDLOOP
  WHILE interactiveMActive
    wait and accept an event
    CASE event OF
      T:event:          reject event;
      I:Off:            pass Off to Gate;
                        interactiveMActive = False;
      I:event:          pass event to Gate;
      O:event:          pass event to Interactive;
    ENDC
  ENDLOOP
ENDLOOP
```

Figure 8: Specification for machine to meet RCa and RCb for P $\hat{}_{\{active\}}$ OI

On this basis a specification to meet the frame concerns of figures 6 and 7 is given in figure 8. The specification is operational in form. It encodes the state machine discussed above, in a pseudo code including a pattern matching facility. Note that in the *case* statements we have chosen to prefix events by the

initial of the domain that generates them. Thus *T:event* represents the binding of an event generated by the Timed machine to a variable named *event*. More specific cases are given by pattern matching with specific event names, e.g. *T:Off*. In this example this is unambiguous and aids comprehensibility. The specification makes a number of important assumptions. Firstly we assume the frame is not super-imposed on machines that are already actively in operation. Removing this assumption leads to an initialization problem (of the world and the machine!) which is almost certainly worthy of a problem diagram in itself. We also assume that the Timed and Interactive machine are correctly specified. For example in the first inner loop we do not explicitly deal with the situation where the Interactive machine generates an event: we assume it doesn't do so as it has not yet received any operator commands. The possibility that in another scenario we are required to deal with a number of such anomalies demonstrates the desirability of our systematic approach to the composition problem.

A scenario in which the Timed machine generates an event whilst the Interactive machine is controlling the gate is dealt with in the third inner loop. The Controller will reject the event from the Timed machine. The timed machine might generate all the correct events (and none other) to meet its specification in isolation from the composed frame. However, the overall system will be meeting the requirements of the composed frame that allows for a weakening of the behaviour specified for the timed machine.

The difference between the semantics of composition for Option 3 compared to Option 2 is that the timed machine should be able to forcibly gain control when the Interactive machine is currently controlling the gate. Clearly the Controller Specification is a variant on that for Option 2, and requires us to make changes only to the third inner loop of that specification. That loop is shown in figure 9 with changes in italics.

```
WHILE interactiveMActive
    wait and accept an event
    CASE event OF
      T:event:      issue Off event to Gate
                    interactiveMActive = False
                    timedMActive = True
                    issue event to Gate;
      I:Off:        pass Off to Gate;
                    interactiveMActive = False;
      I:event:      pass event to Gate;
      O:event:      pass event to Interactive;
    ENDC
  ENDLOOP
```

Figure 9: Changes to Controller to meet RCa and RCb for P ^ {P} OI

4. Related Work

There are very few other general approaches to decomposing problems rather than solutions. Notable exceptions are the goal-based approaches of KAOS [20] and the NFR framework [6]. However these two approaches are not immediately suited to our composition approach, as they do not focus on domain properties.

Composition of software artifacts has been addressed by a range of aspect-oriented techniques [7]. However with the notable exceptions of [15], [23], and [3], aspect-based approaches, whilst capable of addressing design and implementation issues, do not address requirements and their decomposition. The approaches of [15] and [23] are mainly concerned with reconciling inconsistencies between a range of non-functional requirements and do not fully address decomposition of functional requirements. While [3] does consider requirements and their relationship to design and implementation, it does not focus on the issues of composition.

A number of formal approaches exist where emergent behaviours due to composition can be identified and controlled [1,8]. Our approach differs from these in that we identify how requirements interact and remove non-deterministic behaviour by imposing priorities over the requirements set.

The whole area of inconsistency management [11,12,13] offers a variety of contributions to dealing with inconsistencies in specifications. Robinson, in particular [24], reviews a variety of techniques for requirements interaction management, and Nuseibeh et al [22] discusses a range of ways of acting in the presence of inconsistency. None of these approaches address the decomposition and recomposition of requirements to facilitate problem solving.

In [10], a run-time technique for monitoring requirements satisfaction is presented. This approach is taken further in [9], where requirements are monitored for violations and system behavior dynamically adapted, whilst making acceptable changes to the requirements to meet higher-level goals. This requires that alternative system designs be represented at run-time. One view of our approach is that it involves the monitoring of when a requirement leads to system activity (and the taking of appropriate action). Our approach differs further, in that it is more lightweight: we do not need to maintain alternative system designs at run-time.

Our work is related to the feature interaction problem, identified in the field of telecommunications [25]. While less ambitious about the extent to which requirements can be composed, our work is less domain-specific. In [26] work is presented on the conjunction of specifications as composition in a way that addresses multiple specification languages, but the

emphasis is less on the relationship between requirements and specifications.

The need to compose frames is identified in [17] which provides a frame called a Composite frame to this end. However Composite Frames omit any detail of how to reconcile inconsistencies between the requirements of the machines being composed, or how to deal with interference between phenomena.

In [18] we sketched some options in composing a sluice gate control machine with a safety machine in order to address safety concerns. That was in the context of a more philosophical discussion of composition and decomposition. The work presented in this paper differs, in that we embody the composition as an extra machine (in Problem Frame terms). This gives us the potential to deal with a wider range of compositions.

In [16] other concerns of a timed sluice gate are dealt with, in a more formal setting. An example of composition is given, but the conjoining of specifications takes place in a context where the outputs of the two specifications are distinct. As a result, the difficulties dealt with in this paper do not arise.

Our approach is strongly related to the mutual exclusion problem [2] of concurrent resource usage, but with an explicit emphasis on requirements satisfaction.

5. Discussion

We have shown how to systematically analyse the requirements for composing two specific Problem Frames and arrive at the specification for a machine to control the composition. We have given an instance of a new form of Problem Frame encapsulating these ideas. The composition is done non-intrusively in the sense that we have made no changes to the specifications of the machines being composed. In this section we consider how our work can be generalized, alternative composition semantics, and the significance of our work.

It is well understood that in producing a machine to solve a real-world problem there is usually a need to implement an (analogic [17]) model of at least part of the problem domain. Of course arriving at a conceptual model that can subsequently be implemented is often very problematic in itself. In the case of our Timed and Interactive machines not only are the models very simple but they are also rather implicit. This is partly because the original machine specifications assume that no other entity interacts with the sluice gate. For example, the Timed machine assumes that the sluice gate is in the last position it left it in. So we just have a simple cycle of *open-wait-close* at fixed times. What happens if we start to open the gate from a position other than fully closed? Luckily for us this is not problematic as the gate indicates when it has reached the fully open position by generating a Top event. If, for example, the domain description of the sluice gate had said that the sensor that produces the Top event was unreliable, and that failing to stop the gate when it reaches the fully open position could burn out the motor, we would be in more problematic territory. In such a situation we would probably have calculated how long it takes to raise the gate from fully closed to fully open and timed the point at which we needed to generate an Off event. The Timed machine would then be more robust (assuming a reliable timer and that the motor raises the gate at the right speed). However the composed system may still result in a damaged gate if the operator leaves the gate in positions other than fully closed.

To fully resolve the problems in the scenario above, we would need to explicitly model the position of the gate. Composing machines containing such models is more complex, unless they have been originally designed to model changes to domains affected by entities other than themselves. The problem is that the model in one machine may become inconsistent with the world, due to the world being changed by another machine.

This paper has demonstrated that composing machine specifications in order to meet a conjoined requirement requires careful analysis, involving both implementation and requirements issues. However we have also shown one way in which composition comes for free. This was demonstrated in section 3.3 where requirements RCa and RCb were addressed using one shared specification. In a realistic system development, we would expect to have to address many composition problems, so being able to deal with some cases in a lightweight manner is important.

It is not difficult to see how the Composition Frame can be generalized to two machines A and B and a domain C under their control. In the specification we used the notion of a machine becoming active when the gate is currently reacting to it. We modelled the state of the system with respect to either or no machine being active, and had to identify and deal with phenomena involving a change in that state. Of course, for other examples, examining how some domain in the world comes to be 'actively reacting to a machine', and how that condition changes might be much more complex.

Our Composition Frame deals with two specific Problem Frames, a Required Behaviour and a Commanded Behaviour Frame. It is fairly easy to see that it would generalize to composing two Required Behaviour Frames, or similarly two Commanded behaviour Frames. We can deal with other basic Problem Frames in a similar fashion. The basic set of Problem Frames in [17] also includes frames to display information, edit data, and transform data. The interaction issues that need to be dealt with in such

cases by the Controller go beyond those illustrated in this paper to include issues such as sequencing and scheduling [17].

An alternative semantics for composition is a situation where we know that at any given time a specific requirement should apply; e.g., the sluice gate should be subject to operator control during weekdays but timed behaviour at night and during weekends. At first sight this seems to simplify the design of the Controller, as we do not need to dynamically deal with inconsistencies in requirements. We suggest that this may not be the case, since in general we presumably still need to effect a smooth transition between the Timed and Interactive machine [18]. We would go further and say that systematic analysis of the issues in switching between two machines is an important issue in its own right.

Whilst much work has been done on protocols for controlling mutual access to resources in program code, less attention seems to have been paid to the problem of systematically gaining control over domains in the real world [14]. This oversight seems all the more serious when consideration is given to the potential complexities of determining both when a domain begins to react to a machine and when this stops, taking into account in the latter case issues associated with momentum, acceleration and deceleration etc.

6. Conclusions and Future Work

We have shown how to combine two inconsistent requirements in terms of the operator given in section 3.1. We have given a simple example of developing a system through a process of decomposition and re-composition, with solutions to sub-requirements being developed in isolation from other concerns. Our Composition Frame allowed us to reason about the relationship between sub-solutions and sub-requirements. It allows us to arrive at the specification of a Composition Controller (a connector in architectural terms) in order to meet our overall (sub)system requirement. We were able to specify composition at a requirements level (RC) rather than solely in design or implementation terms.

In principle our approach could be extended to deal with n requirements. However practical limits exist due to the fact that, in principle, global consistency cannot be proved through local consistency checking [22]. In this respect it may be helpful to consider issues of scoping.

As noted above, our approach is strongly related to the typical mutual exclusion problem of concurrent resource usage. Clearly one area for future consideration is a mechanism for queuing rather than dropping events that we choose to not immediately respond to.

We are currently looking at combining the Problem Frames approach with the architectural approach of coordination contracts [4], giving us the ability to exploit a more developed architectural approach underpinned by a solid mathematical semantics [8].

Future work is planned in order to tighten up the relationship between our requirements composition operators, the problem frames for sub-problems, the composition requirements (RC) and the composition sub-requirements (RCa and RCb). In this context we will also investigate re-use of composition requirements.

We also need to address a wider range of compositions, both in terms of the options in section 3.1 and across a larger set of basic Problem Frames. The set of basic frames in [17] includes 5 such frames, i.e. 10 possible combinations. The latter generalization may not be as difficult as it sounds since we can pattern match on shared domains. On the other hand, particularly in a large Problem Frames development, sub-parts of domains and amalgamations of domains can appear in different frames.

We would like to be able to automate some of the work of deriving a Composition Frame and its corresponding Composition Controller specification. Related to this is a need to explore more complex examples. We would expect to address this by providing tool support for investigating scenarios implied by our "domain D is currently reacting to machine M" property. It might be possible to develop patterns for particular domain areas.

Finally we would like to explore the relationship between the work presented here, that has quite a pragmatic flavour, and more theoretical work both on inconsistency in requirements and composition.

7. Acknowledgements

The authors are grateful for the support of their colleagues in the Department of Computing at The Open University. In particular we have had many interesting conversations on the topic of Problem Frames with Charles Haley, Jon Hall and Lucia Rapanotti. Thanks are also due to the anonymous reviewers for many helpful suggestions.

7. References

[1] R. Allen and D. Garlan, "A Formal Basis for Architectural Connectors", ACM Transactions On Software Engineering and Methodology, 6(3), 1997, pp. 213-249.

[2] J. Bacon, Concurrent Systems, Addison Wesley, 2003.

[3] E. Baniassad and S. Clarke, "Theme: An Approach for Aspect-Oriented Analysis and Design", In Proceedings of

the International Conference on Software Engineering, 2004.

[4] L. Barroca, Fiadeiro J.L., M. Jackson, R. Laney and B. Nuseibeh, "Problem Frames: a Case for Coordination", Sixth International Conference on Coordination Models and Languages, 2004.

[5] D. Bjørner, "Towards Design Calculi for Requirements Engineering and Software Design", In From Object-orientation to Formal Methods: Dedicated to the Memory of Ole-Johan Dahl, volume 2635 of Lecture Notes in Computer Science, page 21. Springer-Verlag, August 2003. Editors: O.Owe, S.Krogdahl, and T.Lyche.

[6] L. Chung, B.A. Nixon, E.Yu, J. Mylopoulos. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, 2000.

[7] T Elrad, R. Filman and A. Bader (Guest editors). Special Issue on Aspect Oriented Programming. Communications of the ACM, 44(10), October 2001.

[8] J. L. Fiadeiro, A. Lopes, M. Wermelinger, "A Mathematical Semantics for Architectural Connectors", in Generic Programming, R.Backhouse and J.Gibbons (eds), LNCS 2793, Springer-Verlag, 2003, pp. 190-234.

[9] M.S. Feather, S. Fickas, A. van Lamsweerde, C. Ponsard, "Reconciling System Requirements and Runtime Behaviour", Proceedings of the 9th International Workshop on Software Specification and Design, 98.

[10] S. Fickas and M. Feather. "Requirements Monitoring in Dynamic Environments". In Proceedings of the Second IEEE International Symposium on Requirements Engineering, York, England, Computer Society Press, Mar. 1995.

[11] Finkelstein and I. Somerville (eds), special issue of the BCS/IEE Software Engineering Journal on "multiple perspectives", Vol II(1), Jan 96.

[12] C. Ghezzi, and B. Nuseibeh (eds) special issues on inconsistency management in IEEE Transactions on Software Engineering, Vol 24(11), Nov 98.

[13] C. Ghezzi, B. Nuseibeh (eds) special section on inconsistency management in IEEE Trans. on Software Engineering, Vol 25(6), Nov 99.

[14] J.G. Hall, M. Jackson, R.C. Laney, B. Nuseibeh, L. Rapanotti, "Relating Software Requirements and Architectures using Problem Frames", IEEE Proceedings of RE 2002, 2002.

[15] J. Grundy, "Aspect-Oriented Requirements Engineering for Component-based software systems". In Fourth IEEE International Symposium on Requirements Engineering (RE'99). Limerick, Ireland: IEEE computer Society Press, 7-11 Jun 1999.

[16] Ian J Hayes, Michael A Jackson, Cliff B Jones, "Determining the specification of a control system from that of its environment", in Keijiro Araki, Stefani Gnesi and Dino Mandrioli eds, Formal Methods: Proceedings of FME2003, Springer Verlag, Lecture Notes in Computer Science 2805, 2003, p.p. 154-169.

[17] M. Jackson, Problem Frames, ACM Press Books, Addison Wesley, 2001.

[18] M.Jackson, "Why Software Writing is Difficult and Will Remain So", in J.Fiadeiro, J.Madey and A.Tarlecki (eds), Information Processing Letters Special Issue in Honour of Wlad Turski 88(1-2), 2003.

[19] A. van Lamsweerde, R. Darimont, E. Letier, Managing Conflicts in Goal-Driven Requirements Engineering, IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development, November 1998.

[20] A. van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", Proceedings of the 5th International Symposium on Requirements Engineering (RE'01), pp249-261, IEEE CS Press, 2001.

[21] B.A. Nuseibeh, "Weaving Together Requirements and Architecture", IEEE Computer 34 (3), March 2001, pp. 115-117.

[22] B. Nuseibeh, S. Easterbrook and A. Russo, Making Inconsistency Respectable in Software Development, Journal of Systems and Software, 58(2), September 2001, Elsevier Science Publishers, pp. 171-180.

[23] A. Rashid, A.M.D. Moreira, and J Araujo. "Modularisation and Composition of Aspectual Requirements", Aspect Oriented Software Development Conference, 2003.

[24] William N. Robinson, Suzanne D. Pawlowski, Vecheslav Volkov, Requirements interaction management. ACM Compututing Survey, 35(2) (2003), pp. 132-190.

[25] P. Zave, "Feature Interactions and Formal Specifications in Telecommunications", IEEE Computer XXVI (8), 1993, pp. 20-30.

[26] P. Zave, M. Jackson, "Conjunction as Composition", ACM Transactions On Software Engineering and Methodology 2(4), 1993, pp. 371-411.