Extending Problem Frames Projections to Support Subproblems as Services

Charles B. Haley, Robin C. Laney, Bashar Nuseibeh
Department of Computing
The Open University
Walton Hall, Milton Keynes, MK7 6AA, UK
{C.B.Haley, R.C.Laney, B.A.Nuseibeh}@open.ac.uk

Abstract

Subproblems in a problem frames decomposition frequently make use of projections of the complete problem context. One specific use of projections occurs when one subproblem wishes to interact with the machine in a projection that represents another subproblem. Representing the projection, called a service in this paper, as a special connection domain within the using subproblem provides a significant benefit: an interface that defines the shared phenomena and specifies how the subproblems are to be composed. An extension to projections is proposed that realizes this benefit. The usefulness of the extension is validated using a case study.

1. Introduction

Problem frames [5, 6] are used to decompose a larger problem into a set of smaller ones. The process continues until the smaller problems each fit into an understood problem category, or problem frame. The resulting individual subproblems are analyzed, and then the results are recomposed into a solution for the original problem.

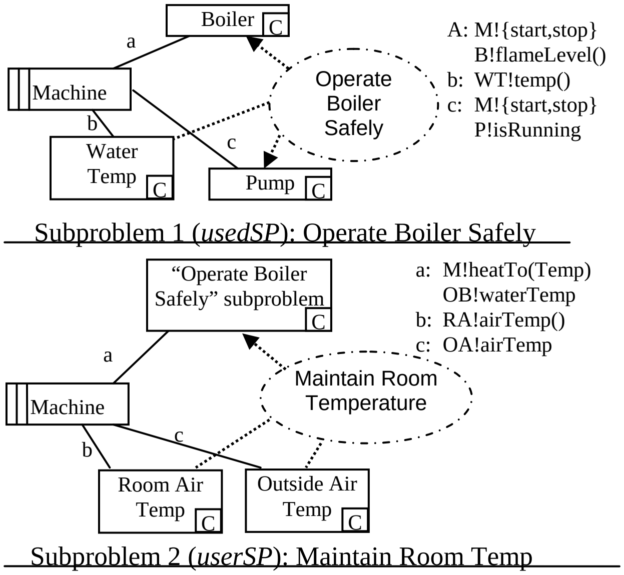
It is normally the case that solutions to subproblems do not use all of the domains in the problem context. It is also possible that a subproblem's solution does not need all of the phenomena used/controlled by the domains. Keeping subproblems clear and focused requires a mechanism for limiting the context of the subproblem to the domains and phenomena necessary to solve the problem. *Context projections* are used for this purpose.

Context projections in problem frames (discussed at length in [6] and briefly but more formally in [4]) are very similar to projections in relational databases [1]. A projection of a relational database table is a new table containing a (potentially improper) subset of columns, and a projection of a problem context is a new context containing a subset of the domains in the problem. The context of a subproblem is a projection of the context of the problem, limiting the domains and/or phenomena in the subproblem to those needed to solve the subproblem.

Some problems push the analyst to consider using the solution of some subproblem *usedSP* as a causal domain (a projection) in the solution of some other subproblem *userSP*. The fact that usedSP is a causal domain in userSP is important; it means that phenomena on usedSP's interfaces affect the behavior of userSP, usedSP, or both.

Figure 1 presents an example, a heating control system that measures air & water temperatures to anticipate the correct water temperature required to maintain the room at the desired temperature. Maintain Room Temp, the userSP, uses a projection to represent Operate Boiler Safely, the usedSP, to supply water at the needed temperature. Maintain Room Temp does not care how the furnace is controlled. It wants heated water, and sends the heatTo(temp) phenomenon to Operate Boiler Safely to accomplish that goal.

The interactions in the example illustrate a specific



Subproblem 1 (*usedSP*): Operate Boiler Safely

Subproblem 2 (*userSP*): Maintain Room Temp

Figure 1. Heat control system as subproblems

form of decomposition. Instead of controlling one of the causal domains in Operate Boiler Safely, Maintain Room Temp wishes to control the subproblem's *machine*. When such a machine-to-machine interface occurs, we say that userSP is using usedSP as a *service*. In the example, Maintain Room Temp is using Operate Boiler Safely as a service to supply heated water.

There are two difficulties with the use of projections as services as shown in Figure 1. The first difficulty is that no domain in the projection of Operate Boiler Safely receives the phenomenon heatTo(). The only choice for a domain to be controlled by the phenomenon is the machine in Operate Boiler Safely, but there is no interface in the subproblem with which the machine can share the phenomenon. The end result is that the two subproblems are incompletely specified and will not compose.

The second difficulty is control of visibility. Nothing indicates which phenomena in Operate Boiler Safely *should* be visible to Maintain Room Temp. The phenomenon waterTemp() is passed to Maintain Room Temp, but is it the only one? Maintain Room Temp can in theory control any phenomena used in Operate Boiler Safely, which implies that Maintain Room Temp could directly (and probably incorrectly) control the boiler.

The extension to projections proposed in this paper resolves both of these difficulties. The extension is validated using a case study: the decomposition of a lighting control system. The case study is an expanded version of the example in [3][1]. The case study investigates the use of context projections as services by exploring certain difficulties such as concurrency and security. It is not intended to provide a finished analysis.

The remainder of this paper is structured as follows. Section 2 describes the proposed extension to problems frames notation. Section 3 begins the case study, presenting a requirements statement for a lighting control system. Section 4 presents the context diagram for the problem. Section 5 explores decomposing the problem using projection domains. Section 6 examines recomposition, section 7 looks at specific concerns raised by the decomposition, and section 8 concludes.

2. Projections as Connection Domains

As noted above, an analyst may wish to solve subproblem *userSP* by using subproblem *usedSP* as a service. The two difficulties described above must be resolved: which domains in usedSP receive phenomena controlled in userSP, and which phenomena controlled by domains in usedSP are visible outside its projection.

2.1. Solution Using Projections of the Machine

We first look at how one might approach solving the problem using two projections of the machine in the userSP subproblem. One projection is userSP's machine; it represents the machine and phenomena necessary to solve the userSP subproblem. The other projection represents the machine in usedSP along with the subset of phenomena exchanged between the userSP and the usedSP subproblem. Figure 2 presents the heat control example redone in this style. Subproblem two, Maintain Room Temp, now contains a causal domain that is a projection representing the Operate Boiler Safely machine. It also contains the Water Temp causal domain. The phenomenon heatTo() is controlled by the Maintain Room Temp machine. The phenomenon temp() is controlled by the Water Temp domain.

This solution resolves our first difficulty. The interfaces are completely specified in both subproblems. This being said, the solution is not ideal. Nothing in the solution of Operate Boiler Safely indicates that it might be controlled externally. Someone looking at the subproblem would not know why the subproblem exists. In addition, the Maintain Room Temp subproblem assumes that the phenomena on the interface between the two machines are valid, but no information on either subproblem diagram permits the assumption to be verified. The second difficulty (visibility) is not resolved.

The next section presents an alternate solution that better resolves the difficulties.

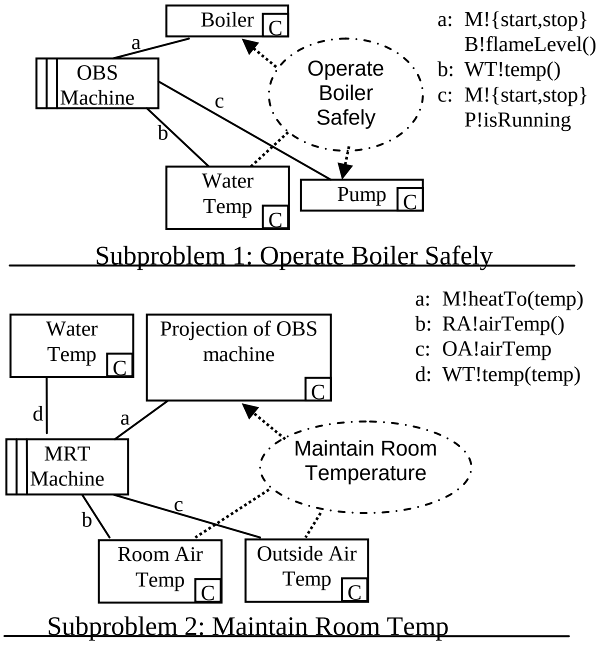


Figure 2. Heat system w/machine projections

[1] Although developed independently, the scenario resembles one found in [8]. The major differences are multiple control interfaces, incorporation of security requirements, and dynamic definition of 'rooms' for control purposes.

2.2. Solution Using Explicit Connection Domains

The two difficulties from Section 1 can be better resolved by inserting a *connection pseudo-domain* into both projections, making the connections between userSP and usedSP explicit and symmetric. The inserted domain is a *pseudo-domain* because it is fictitious, not representing something physical in the problem. It is a *connection domain* because it represents the point through which the world connects to the domains in the projection. The inserted pseudo-domain *represents* the projection in subproblems that use the projection. The pseudo-domains are referred to as *projection domains*.

To better support validation, we propose a strict definition/reference relationship between the one subproblem that *defines* the service and subproblem(s) that *reference* the service. A *defining occurrence* is a projection domain in the subproblem that *provides* the service (Operate Boiler Safely in Figure 1), and acts as a causal domain within the subproblem. The phenomena on its interfaces are the phenomena available to the service, specifying the phenomena controlled by the service and available to subproblems that use the service, and the phenomena the service is willing to respond to.

When a subproblem *uses* the service, the subproblem will contain a *referencing occurrence* projection domain. The referencing occurrence acts as a causal domain within the using subproblem. Completeness must be preserved: all phenomena appearing on an interface of the referencing occurrence must appear on an interface of the defining occurrence (or be declared somehow as *optional*, a possibility not further discussed in this paper), and vice versa. Directionality must be preserved: all phenomena controlled by the referencing occurrence must be controlled by a domain on one of the defining occurrence's interfaces, and all phenomena used by the referencing occurrence must be controlled by the defining occurrence.

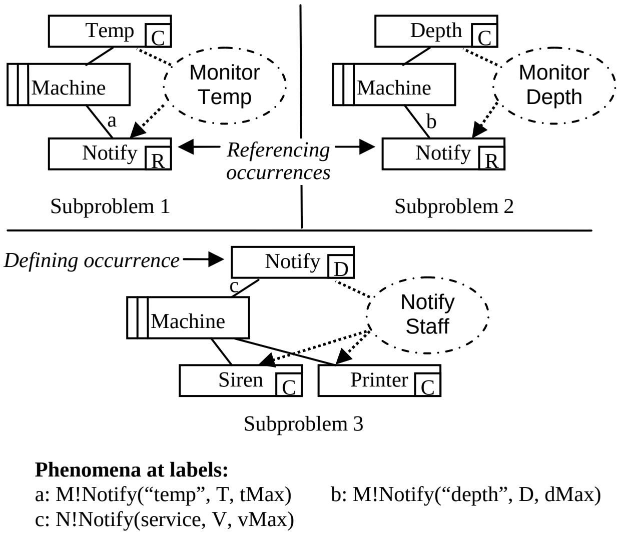


Figure 3. Example of projection domains

Figure 3 illustrates the use of projection domains. The example contains a common 'consumer' subproblem used as a service by two 'producer' subproblems. Producer subproblem one monitors the temperature of some liquid. Producer subproblem two monitors the depth of the liquid. If the values go beyond a limit, the 'consumer' subproblem (subproblem three – the service) is used to notify the staff by sounding a siren and printing something. Subproblems one and two contain referencing occurrences. Subproblem three contains the defining occurrence. The phenomenon Notify is used by the referencing occurrences and controlled by the defining occurrence, preserving completeness and directionality. A defining occurrence is indicated on a problem frame diagram by a projection domain with type **D** (**D**efining); the name must be unique across the set of subproblems. A referencing occurrence is indicated by a projection domain with type **R** (**R**eferencing).

Figure 4 presents the heating control example again, this time using projection domains. A defining occurrence is added to subproblem one, Operate Boiler Safely. The defining occurrence, *Operate Boiler*, controls the heatTo phenomenon on the interface between it and the machine. The waterTemp phenomenon is on the interface between the defining occurrence and the Water Temp domain. Subproblem two, Maintain Room Temp, contains a referencing occurrence standing for the boiler operation service. The referencing occurrence is named *Operate Boiler* to connect it by name with the defining occurrence. The referencing occurrence has the same phenomena on its interface as the defining occurrence, preserving complete-

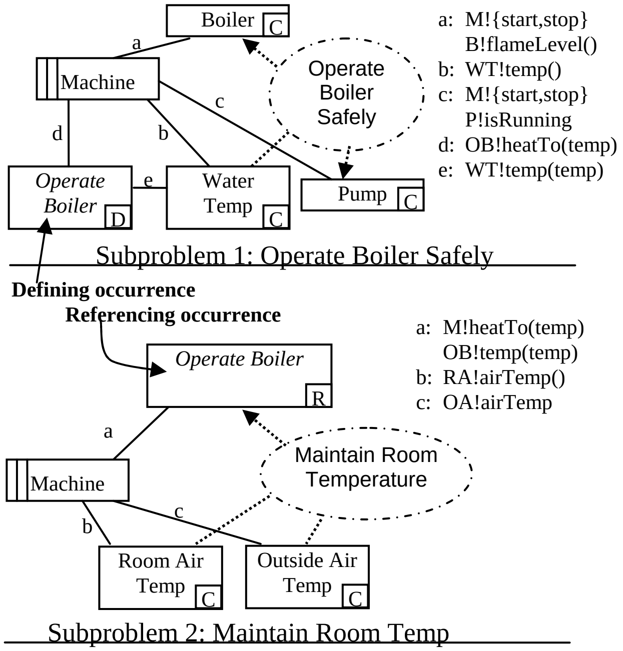


Figure 4. Heat system with projection domains

ness. The referencing occurrence controls the waterTemp phenomenon and uses the heatTo phenomenon, thus preserving directionality.

The use of projection domains satisfactorily resolves both the difficulties listed in Section 1. As in the first solution, all interfaces are completely specified. However, projection domains do not exhibit the problems found in the first solution. They exist in both the using and used subproblems, assisting an analyst with understanding the solution, and they permit a basic level of correctness verification. The second difficulty is resolved because *all* the phenomena that a referencing occurrence can use must be found on an interface on the defining occurrence.

In closing, note that the defining occurrence is similar to a *façade* [2]. It also plays a role similar to an interface in Java, exposing some functionality of the projection while hiding the bits that are private to the projection. Using the referencing occurrence is equivalent to using the façade or interface. As do façades and interfaces, defining and referencing occurrences provide the possibility of automatically verifying at some level that the projection is being used properly.

3. The Lighting Control System Case Study

The lighting control system to be built must conform to the following problem statement, provided by the firm constructing the building. The problem statement is:

> The architect wishes to have a lighting control system for a building. From the user's perspective, the system consists of switches and lighting units (lights) associated with a room. When a switch is actuated, the associated light or lights in the room must be turned on or off.
>
> The architect requires the use of up/down momentary contact switches. A momentary contact switch must cause its lighting units to be in the state indicated by the switch's motion: up turns the lights on if they are not already on and down turns the lights off if they are not already off.
>
> The system must include a master control panel that indicates the state of the lighting units in each room. If the lights are on in a room, the indicator on the panel shows green. If the lights are off, the indicator does not glow. The state of the lights in any room can be changed using the panel. Only certain people are allowed to use the control panel; they must identify themselves using a proximity badge (see below) that confirms their identity.
>
> Lighting units contain a unique identifier. The system must keep track of where each lighting unit is (which room it is associated with) and how long any given lighting unit has been illuminated.
>
> Certain lights are in secure rooms and are to be actuated only by people with an appropriate level of authorization. Users carry an identity card (a proximity badge) that is read by a proximity reader either embedded in or installed next to a switch. Lack of a card means the person has the lowest level of authorization possible. The level of security necessary for a room is established using the master control panel. The system must record who operated the lights in a secured room. A person who lacks authorization may not change the state of the lights.
>
> The owner of the building requires the system to be able to trace all light on or light off actions, printing the trace in real time on a printer in the control room. If this printer is not working correctly, an alarm of some kind must be given.
>
> The system must monitor the lighting units. If a lighting unit is not in the correct state (e.g. off when it should be on, or not responding at all), the system must try to correct it. If the correction fails, the system must indicate this fact by changing the indicator on the master control panel of the room containing the failing lighting unit to show red and logging the printer discussed above. The detection of a lighting unit not associated with any switch is to be logged on the printer and indicated by illuminating a red indicator on the master control panel reserved for this eventuality.

4. The Light Control Context Diagram

The problem statement and requirements leads us to propose Figure 5 as the context diagram for the system.

This diagram mentions all the components of the system listed in the problem statement and relates them to the machine. Unfortunately, the diagram leaves out several important parts of the problem. For example, the relationship between people and badges cannot be determined. The badge identifies the person to the system, and establishes the person's privileges. The privileges determine whether the switch actuation is to be honored.

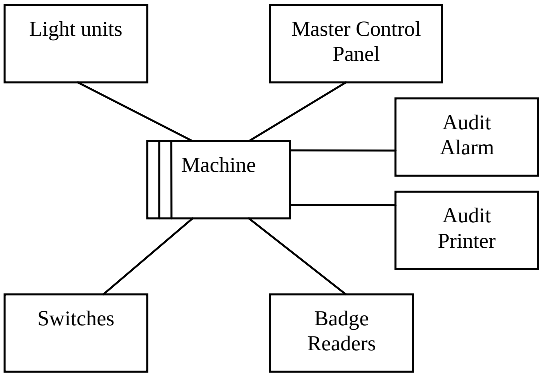


Figure 5. An incomplete context diagram

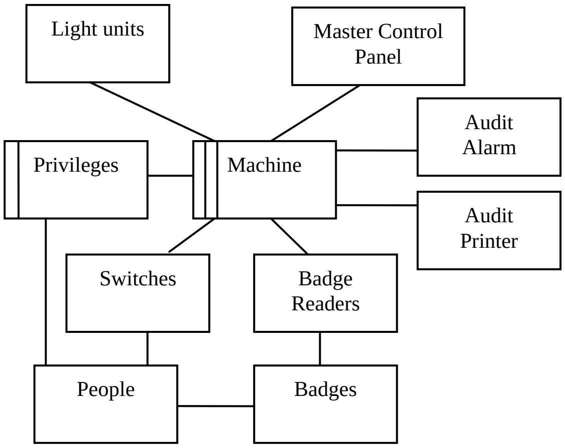


Figure 6. The completed context diagram

Therefore, the person, the badge, and the privileges are important parts of the problem and should be included in the context diagram. After doing so, we have the diagram shown in Figure 6.

5. Subproblem Diagrams

5.1. Initial Thoughts

There is nothing physical that relates a switch to the lights it controls or to the logical room that contains the lights. Equally, there is nothing physical that relates a badge reader to a switch or to a room, or relates a badge to a person. It seems that the notion of *room* is a unifying concept fundamental to the problem, and perhaps the problem could be decomposed along that dimension.

Actuating a switch is a request that the state of the lights in a room be changed. From the user's point of view (and the switch's as well), the lights in a room are treated as a unit. It makes sense, therefore, to incorporate the notion of *room* into the switch phenomena along with the *up* and *down* phenomena. A method to map switches and lights to rooms is required. Following this line of reasoning further, it becomes clear that the badge and privilege determination are separate from the switch actuation. A badge is associated with a person and privilege is associated with a person/room pair, meaning we need another map. We thus end up with the lexical domains *People → Privileges, Switches → Rooms*, and *Rooms → Lights*.

One of the fundamental problems, controlling the lights, seems to be a *commanded behavior* problem. People are commanding the lights using the switches and the master panel. However, it would seem that the master panel presents enough differences from use of the 'normal' switches to justify separating the two into distinct subproblems, Switches & Lights and Master Control Panel. We must next consider the Audit problem, which responds to the parts of the problem statement requiring verification that the lights are in the state that they should be. The last problem is the maintenance of the lexical domains.

Please note: to keep the diagrams simpler, phenomena are not shown in the subproblem diagrams. As will be noted later, they should be.

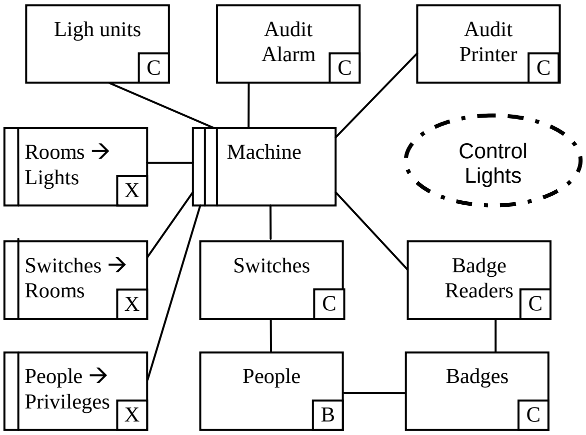


Figure 7. Overly complex subproblem diagram

5.2. Switches & Lights Problem – Attempt One

Accepting this first analysis, a first-try problem frame diagram for the switches subproblem is shown in Figure 7. Unfortunately, this problem frame diagram is far too complex to be of use. It hides multiple requirements under the name Control Lights, and it doesn't fit any basic problem frame. For these reasons and others, it is not worth trying to complete the requirements arrows or frame concern. We need to subdivide the problem further.

We start by connecting the switches to the lamps in the rooms that they control. This is a commanded behavior problem. The requirement, derived from the system requirements and roughly stated, is *if the user actuates a switch, then the lights in the room(s) associated with the toggle shall turn on, turn off, or remain as they are, depending on the actuation.* The problem diagram would look something like the one shown in Figure 8.

We now turn our attention to the security aspects of the problem. The requirement is, again roughly stated, *if a room is secured, then only people with the appropriate permission can cause a state change in the lights*. People are identified by badges. This is a required behavior problem; the diagram looks something like Figure 9.

The subproblem in Figure 9 suffers from the same flaw as the one shown in Figure 7; it is overly complex. For example, according to the information supplied, a badge reader notes when a person enters and exits its detection area. How the badges interact with the switches is not clear. This behavior is not made explicit in the diagram, and it is difficult to do so without adding secondary requirements. We need to throw away the solutions in

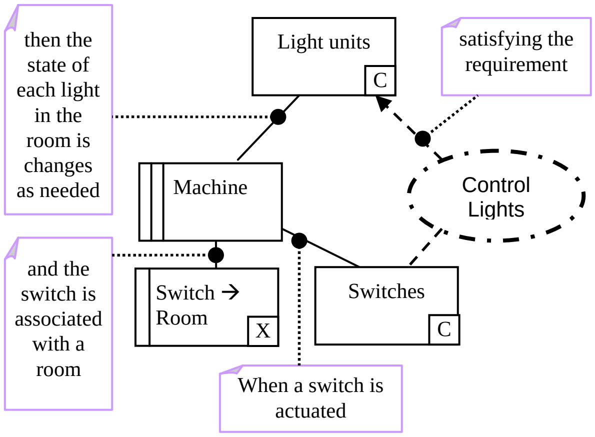

then the state of each light in the room is changes as needed
Light units
C
satisfying the requirement
Machine
Control Lights
and the switch is associated with a room
Switch → Room
X
Switches
C
When a switch is actuated

Figure 8. Basic lights control subproblem

Figure 8 and Figure 9, and further subdivide the problem.

5.3. Switches and Lights Problem – Attempt Two

We can reduce the complexity of the security problem by introducing a model that uses badge reader events to maintain a database of who is in a room. This model will be the interface between the lights control problem and the badge reader problem. The *enter* and *exit* events generated by the badge reader give us the information we need to build the model. A person is considered able to control a room between enter and exit events. The model is used by a second subproblem that verifies permissions and enforces security. Following this route, we find we have two problems, one to build the Person → Room model and one to use it. Figure 10 presents the first subproblem – constructing the model.

Using the model would seem to be straightforward. The required behavior problem would be similar to the switches commanded behavior problem in Figure 8. However, the resulting diagram would again suffer from being overly complex, due to the interactions between the associating the switches with rooms and then checking security for those rooms. The too-complex problem diagram is not shown.

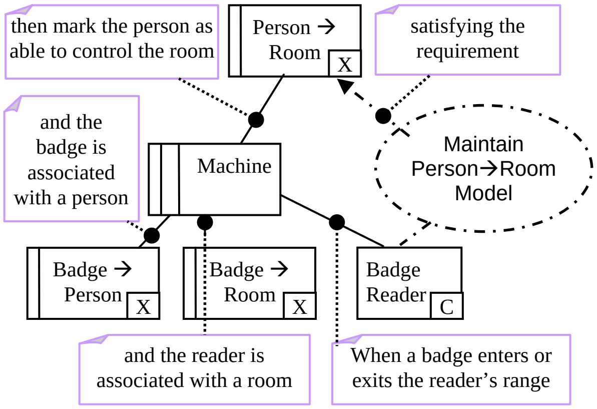

then mark the person as able to control the room
Person → Room
X
satisfying the requirement
and the badge is associated with a person
Machine
Maintain Person→Room Model
Badge → Person
X
Badge → Room
X
Badge Reader
C
and the reader is associated with a room
When a badge enters or exits the reader's range

Figure 10. Building the person → room model

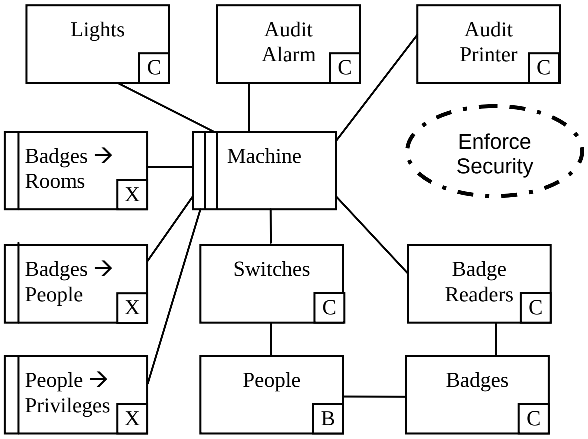


Figure 9. A complicated security solution

To remove the complexity, the problem is broken into subproblems where one subproblem uses the other as a service. A projection domain is used to form the link.

The first subproblem, named Honor Switches and shown in Figure 11, is the same as Figure 8 except that the controlled domain is now a service, indicated by reference to a projection domain named *Control Lights*, defined in Figure 12. The phenomena passed to *Control Lights* are shown on the diagrams; they are **on(room)** and **off(room)**. Note: Figure 11 uses a notational convenience: names of projection domains are shown in *italics* as well as by their definition-type letter (D or R).

Figure 12 shows the Enforce Security required behavior problem that Honor Switches uses as a service. Enforce Security accepts the **on** and **off** phenomena produced by Honor Switches, then checks to see if the room is secure. If the room is secured (in the *Person → Room* model) then verify that at least one person near a panel for the room is permitted to control the lights for that room. If permitted or if the room is not secured, then pass the events along through a reference to a service Control units in room (described below) through the projection domain

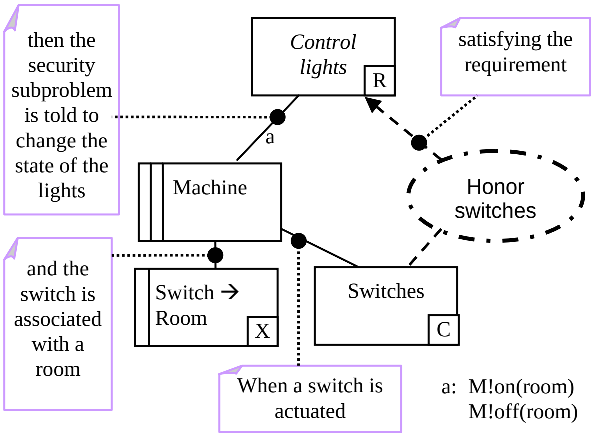

then the security subproblem is told to change the state of the lights
Control lights
R
satisfying the requirement
a
Machine
Honor switches
and the switch is associated with a room
Switch → Room
X
Switches
C
When a switch is actuated
a: M!on(room)
M!off(room)

Figure 11. Lights control with security

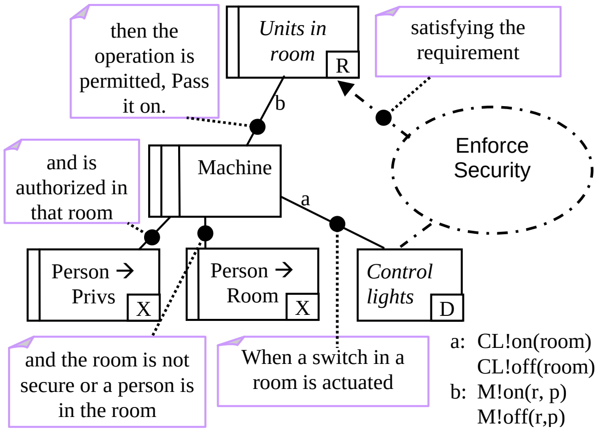

then the operation is permitted, Pass it on.
Units in room
R
satisfying the requirement
b
and is authorized in that room
Machine
Enforce Security
a
Person → Privs
X
Person → Room
X
Control lights
D
and the room is not secure or a person is in the room
When a switch in a room is actuated
a: CL!on(room)
CL!off(room)
b: M!on(r, p)
M!off(r,p)

Figure 12. Enforce security

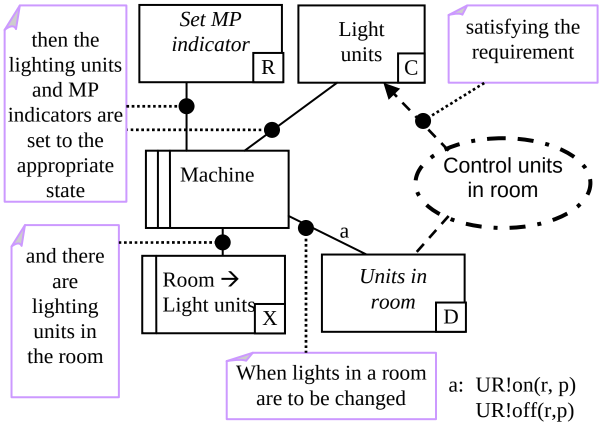

then the lighting units and MP indicators are set to the appropriate state
Set MP indicator
R
Light units
C
satisfying the requirement
Machine
Control units in room
and there are lighting units in the room
Room → Light units
X
a
Units in room
D
When lights in a room are to be changed
a: UR!on(r, p)
UR!off(r,p)

Figure 13. Control lights in room

Units in room, defined in Figure 13. The phenomena passed along are of the form **on(room, person)** and **off(room, person).**

We end with the diagram in Figure 13, Control units in room where the *Units in room* projection domain is defined. It is a commanded behavior problem, looking up which lights are associated with the room and controlling them appropriately. It informs the Maintain MP Indicators subproblem (discussed in the next section) what it did using the service defined by the *Set MP indicator* projection domain.

5.4. The Master Control Panel

The Master Control Panel problem is decomposed into three subproblems. The first, shown in Figure 14, is an information display problem in which the indicators are set appropriately and the audit trail is maintained. It defines the projection domain *Set MP indicator* through which it accepts **on** and **off** phenomena from the Control units in room subproblem.

The second subproblem concerns controlling the lights from the master panel. Shown in Figure 15, it is a commanded behavior problem where pushing a button associated with a room inverts the state of the lights in that room. It uses the service represented by the projection domain *Units in room* and defined in Figure 13 to actually control the lights.

The third subproblem is concerned with master panel security, and is a required behavior problem. As this subproblem is almost identical to the Enforce Security problem presented in Figure 11, the subproblem will not be further discussed here.

5.5. The Audit Subproblems

The Audit problem is decomposed into two information display subproblems and one commanded behavior subproblem. The first information display subproblem, Audit lights unit shown in Figure 16, scans the lights in each room to determine if they are in the proper state. The fault indicator on the MP is lit via the projection domain *MP fault indicator* if some unit is not in the correct state.

The information display subproblem defining the *MP fault indicator* projection domain is very similar to Figure

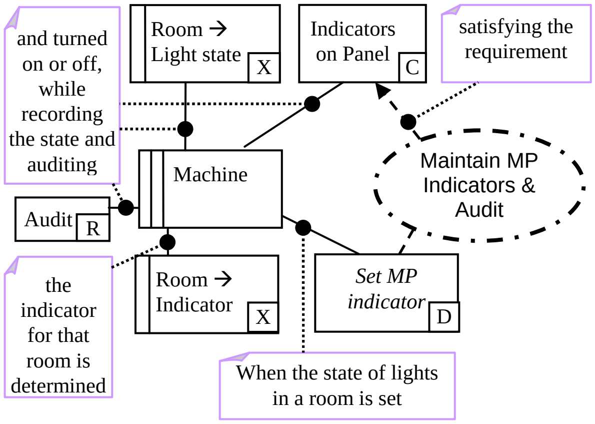

and turned on or off, while recording the state and auditing
Room → Light state
X
Indicators on Panel
C
satisfying the requirement
Machine
Maintain MP Indicators & Audit
Audit
R
the indicator for that room is determined
Room → Indicator
X
Set MP indicator
D
When the state of lights in a room is set

Figure 14. Master control panel

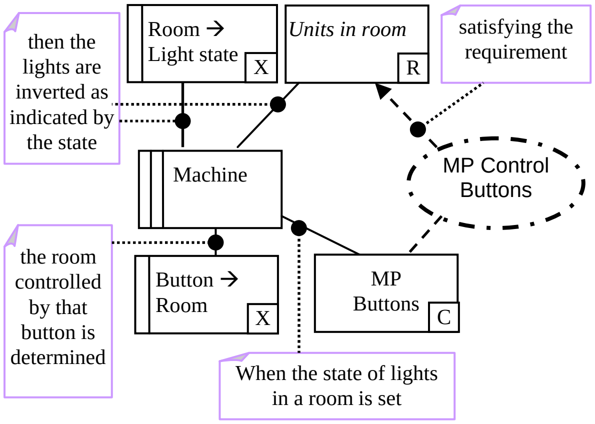

then the lights are inverted as indicated by the state
Room → Light state
X
Units in room
R
satisfying the requirement
Machine
MP Control Buttons
the room controlled by that button is determined
Button → Room
X
MP Buttons
C
When the state of lights in a room is set

Figure 15. Master control panel buttons

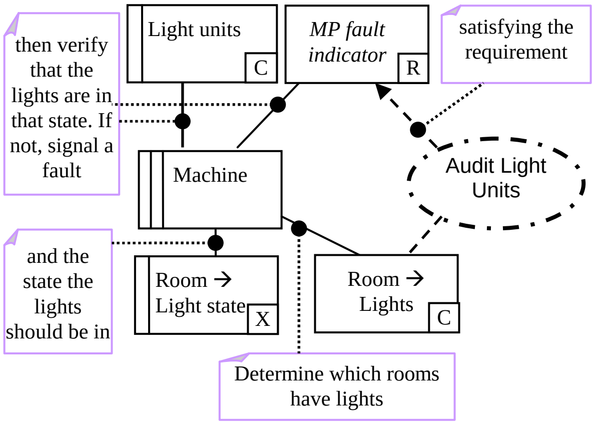

then verify that the lights are in that state. If not, signal a fault
Light units
C
MP fault indicator
R
satisfying the requirement
Machine
Audit Light Units
and the state the lights should be in
Room → Light state
X
Room → Lights
C
Determine which rooms have lights

Figure 16. Audit light units

14, as is the subproblem defining the projection domain *Audit*. These subproblems are not further discussed.

The job of the commanded behavior problem is to put the lights into the state they should be in. It is identical to the information display problem in Figure 16, except that it would use the service represented by projection domain would be *Units in room*, defined in Figure 13.

5.6. The Lexical Domains

Several lexical domains have been used in the above diagrams. The creation and maintenance of each of these is described by a simple workpieces problem frame. The subproblems are all very similar and have solutions well described in [6], so they won't be further discussed.

6. Recomposition

Recomposition of the subproblems into a solution to the original problem raises the following concerns.

6.1. Verify the Context

Each subproblem is an incomplete projection of the context. It is interesting to note that the context diagram itself appears to be a projection of something left unsaid, as some designed domains in the problem diagrams (e.g. the lexical domains) do not appear in the context diagram.

One could argue that any designed domain that appears in more than one problem diagram should also appear in the drawn context diagram. The rationale is that any domain that appears in only one problem diagram resolves some concern internal to that subproblem, introduces no composition concerns, and is therefore not part of the problem context. On the other hand, if a designed domain appears in more than one subproblem, it could easily introduce composition concerns. One is therefore tempted to assert that *designed domains that resolve internal concerns and appear in two or more projections should be included in the context diagram.*

6.2. Verify Phenomena across Projection Domains

Directionality must be preserved on phenomena on the interfaces of projection domains. This fact raises a naming problem; different phenomena with the same name cannot cross through the projection domain without creating confusion on the other side. It is sufficient to ensure that if the names of two phenomena entering a projection domain are the same, then the phenomena have the same meaning. Another solution would be to add a phenomenon + interface map to the projection domain, mapping phenomenon names from the controlled to controlling interfaces.

The verification process must extend throughout the virtual context (across all projection domains), ensuring that controlled/used relationships are correct and that the parameters of the phenomena are consistent.

6.3. Verify the Hidden Connections

The set of subproblems Audit Light Units, Maintain MP Indicators and Control Lights in Room illustrate a potential source of errors that seems hard to detect automatically. Audit Light Units depends on the existence of the lexical domain Room → Light state, which is maintained by Control Lights in Room and Maintain MP Indicators. However, there is no easy way to verify that these subproblems correctly use the lexical domain in this kind of hidden connection between subproblems.

6.4. Distribution

During recomposition, one must decide whether all the subproblem machines will compose to one machine, or if the system is distributed in some way. In theory, the machine in each subproblem could be a separate computer. In practice, this will not happen, and in some cases it cannot happen. For example, the existence of shared state could force merging. Does this mean that if two machines control the same symbolic phenomenon, they must be combined? A similar question must be asked about lexical domains to determine if they can be used in a distributed fashion (as a distributed database).

As phenomena are 'shared', one could argue that distribution is *never* allowed because it breaks the simultaneity assumptions of problem frames analysis. Ignored connection domains create similar difficulties. For example, in comments to Charles Haley [7], Michael Jackson says that "guards to be evaluated in one subproblem could be added to events in another subproblem. This solution method is particular to one kind of composition and to a special (undistributed) kind of problem environment.” This comment clearly states that certain solutions force the analyst to use a non-distributed implementation.

It would be very nice to have a better understanding of and a way to specify the cases that force merging of the machines. Indicating the simultaneity and concurrency assumptions at an interface would help enormously.

Projection domains assist with determining whether distribution is acceptable by specifying the interface between a defining occurrence and its using occurrences. Specifying the cardinality at these interfaces as described in [3] would provide more information, as cardinalities other than 1:1 imply at least some support for concurrency and distribution.

6.5. Concurrency

The notion of trying to detect potential concurrency problems during composition is intriguing.

Concurrency problems exist on at least two levels. The first is rather large, exemplified by lexical domains and models. There is an inherent concurrency problem between a machine that maintains the domain and a machine that uses it. The problem manifests itself as inconsistent or partial state. It would seem that this sort of problem is amenable to solution, at least at the phenomena level, by applying transaction semantics to the phenomena.

The second level can be illustrated by looking at the example presented in this paper. It is perfectly permissible to have multiple switches for the same room. The switches and lights in a room may not be controlled by the same computer, leading to potential race conditions as the switches are actuated. Clearly the nature of a concurrency problem depends on how the system is distributed.

7. The Specific Concerns

Many concerns arise because of problem recomposition or conditions outside the analysis. These are the *specific concerns* in [6]. Some are looked at here.

7.1. Initialization

Some of the initialization concerns might be:

What happens after a power fail?

What is the system supposed to do when power is applied, either for the first time or after a power fail? Is the building to remain dark, or are the lights restored to their previous state? As an example of what might come out of a discussion of this form, we might discover that lighting units have a *safety switch* on them. If the switch is at *safety*, when power is applied to the lighting unit, the light is illuminated. This state is to be maintained until the unit is told otherwise. The existence of a safety switch and what it implies would certainly change several problem diagrams, in particular the audit subproblems.

What about partial power failures, where the controller loses power but the lights don’t?

There are several subquestions that might arise while discussing this point. Does a partial power failure trigger a safety concern? Can power be lost to parts of the control system, and if so what is to occur while power is lost and when power is restored? The problem is complicated by use of a distributed implementation, as different parts of the system could be ‘off’ at any given time.

The audit process cannot run until system is initialized.

This is an example of initialization sequencing. The audit system depends on having the various lexical domains correctly initialized and the lights in a known state. The point after which auditing can start must be determined, and then a required behavior frame added to express the requirement.

Lights added to a room may be in an incorrect state.

A maintenance engineer may repair or replace a lighting unit while the system is running. Doing so raises concurrency concerns (maintenance of the lexical domains), correctness concerns (the newly installed light is off when it should be on and vice versa), identities concerns (movement of units from another room), etc.

7.2. Identities

There are many identities concerns. Most of them are recognized by the inclusion of the lexical domains (the → maps). Some, however, cannot be satisfied with the domains. For example, a switch might be added to the system but not associated with any room. A lamp, set to *safety*, might be added to the system but not associated with a room. Badge readers present a similar problem.

Another identities concern that will cause changes to the problem diagrams comes from the assumption that switches are in rooms and badge readers are in rooms, therefore someone in the room is actuating a switch. This assertion is clearly incorrect if there are multiple badge reader/switch pairs associated with a room. We can confuse the identity of a person at the switch with a person at another switch for the same room. The solution is to map both badges and switches to a pair *(room, location)* instead of to *room*. The diagram in Figure 10 would be changed to build a Person at Location model. The diagram in Figure 10 would be changed to use the Person at Location model. Finally, the diagram in Figure 11 would be changed to use a *Switches → Rooms/Location* map.

7.3. Interference

The decomposition creates several interference or concurrency questions. For example, without care the Audit machine can busily undo the Honor Switches machine's actions. Interactions between the audit information display and audit setting the correct light state could make panel indicators flash. If two switches control the same room and one switch commands *off* while the other commands *on*, individual lights could be left in conflicting states. Inconsistent states while maintaining the lexical domains is another source of errors.

7.4. Reliability

The reliability concern touches several of the other concerns. For example, the safety question was discussed above. How the system degrades in the face of power or component failure is another.

8. Conclusions

The case study shows that projection domains help with modeling one subproblem using another as a service. Projection domains help keep the subproblems focused while specifying how the subproblems interact. They preserve completeness and directionality, providing a way to verify that all phenomena used and controlled by the defining subproblem are controlled and used by the referencing subproblem(s), and vice versa. They better encapsulate the service, as the phenomena visible at the projection's interface are defined by the defining occurrence and not by the subproblem using the service. They also provide a form of continuous composition by specifying the interface between a defining occurrence and its referencing occurrence(s).

Although projection domains resolve some composition problems, the case study showed that more remain. Future work will focus on ensuring consistent use of lexical domains by multiple subproblems, verifying the semantics of shared phenomena and their parameters, and describing and verifying the concurrency properties of domains and subproblems.

The case study brought several 'use of problems frames' issues to the surface that would be helped by tool support:

1. Recomposition of subproblems is non-trivial. Tool support would help by assisting tracking of domains through the various subproblems.
2. One should not take shortcuts with phenomena. One cannot easily reason about the *specific concerns* without the phenomena, which is why most of the specific concerns are left unresolved. Unfortunately, phenomena and how they are used changes rapidly during analysis, encouraging the analyst to 'wait until the end' to enter them into the diagrams, but the end never comes. The maintenance of their consistency across multiple problems is also difficult. Again, tool support would be very helpful.
3. It is not always clear if and when designed domains should be added to the context. As noted above, it seems that they should be in the context if they appear in more than one subproblem.
4. The *specific concerns* can point to changes needed in the subproblems. The analysis is not complete until they are all resolved, but there is no easy method to verify their global resolution. A tool might support some form of checklist, assisting the analyst in verifying that the concerns have at least been considered.

Acknowledgements

The financial support of the Leverhulme Trust is gratefully acknowledged. We also thank Michael Jackson for his highly pertinent comments, criticisms, and help.

References

[1] T. Connolly, C. Begg, and A. Strachan, *Database Systems: A Practical Approach to Design, Implementation, and Management*, Second ed. Addison-Wesley, 1998.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[3] C.B. Haley, "Using Problem Frames With Distributed Architectures: A Case for Cardinality on Interfaces," *Second International Software Requirements to Architectures Workshop (STRAW'03), International Conference on Software Engineering (ICSE '03)*. Portland OR USA, 9 May 2003.

[4] J.G. Hall and L. Rapanotti, *Towards a Semantics of Problem Frames*, Technical Report 2003/05, Department of Computing, The Open University, Milton Keynes UK, 2003.

[5] M. Jackson, *Software Requirements & Specifications*, Addison Wesley, 1995.

[6] M. Jackson, *Problem Frames*, Addison Wesley, 2001.

[7] M. Jackson, "Personal Communication to Charles Haley: Response to Questions about Problem Frames," 2003.

[8] S. Queins, G. Zimmermann, M. Becker, et al., "The Light Control Case Study: Problem Description," *Journal of Universal Computer Science*, vol. 6 no. 7, Jul 2000, pp. 586-596.