Using problem frames with distributed architectures: a case for cardinality on interfaces

Conference or Workshop Item

Using Problem Frames with Distributed Architectures: A Case for Cardinality on Interfaces

Charles B. Haley
The American University of Paris, Paris, France
The Open University, Milton Keynes, UK
charles [at] the-haleys.com

Abstract

Certain classes of problems amenable to description using Problem Frames, in particular ones intended to be implemented using a distributed architecture, can benefit by the addition of a cardinality specification on the domain interfaces. This paper presents an example of such a problem, demonstrates the need for relationship cardinality, and proposes a notation to represent cardinality on domain interfaces.

1. Introduction

In a Problem Frames analysis [3, 4], *domains* share *phenomena* at their *interfaces*. One of the domains in the analysis is the *machine domain*, which represents the software to be constructed by the developer. Phenomena are the externally visible characteristics of the domains. The phenomena visible at the machine domain's interfaces drive much of the analysis process.

The existence of certain phenomena can be predetermined by purchased products to be used in the system [1] or by considering architectural implications early in the requirements cycle [6, 7]. Hall et al [2] argued for extending Problem Frames to take architectural considerations *within* the machine domain into account, thus incorporating domain knowledge into the analysis. This paper takes the argument one step further, arguing that there are architectural considerations that affect the propagation of phenomena *between* domains, and that it is helpful to explicitly note these considerations in the diagrams.

In a 'standard' Problem Frames analysis, phenomena are considered shared and instantaneous. All domains that participate in a given interface share the phenomena; participation is a relationship. The question of cardinality of the relationship does not arise, because the phenomena are always shared by all. However, a class of problems exists wherein it is convenient to define more precisely how phenomena are shared over an interface. The case comes up when the implementation of a system is to contain redundancy or be partitioned into semi-autonomous units, such as what occurs when using a distributed architecture. The originating domains may need to know about how phenomena are propagated, either for correctness or for efficiency. Using explicit *connection domains* can resolve the problem, but they introduce complexity. The author argues that by noting *cardinality* on the interfaces, appropriate information can be included in the analysis without a significant increase in complexity.

Section 2 of this paper describe a small lighting control system using Problem Frames. Section 3 presents one possible implementation, showing a case where the current shared phenomena notions do not expose certain difficulties. Section 4 proposes an extension to Problem Frames notation to correct the problem, and Section 5 presents conclusions.

2. The Lighting System

2.1. The Problem Statement

A lighting control system is to be built that conforms to the following problem statement, provided by the firm constructing the building.

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 user actuates a switch, the associated light or lights in the room are 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 change to the state indicated by the switch's motion, if needed: up turns the lights on if they are not already on and down turns the lights off if they are not already off.

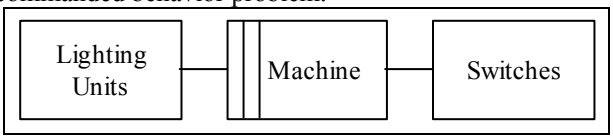
The system is to be built using networked components and to include redundancy where appropriate.

Discussions with the architect and the vendors of the lighting equipment establish the following facts:

1. Switches and lighting units are connected by a network. They are not able to converse directly with each other.
2. A *room* is a logical concept, covering from part of a 'real room' to multiple floors of a building.

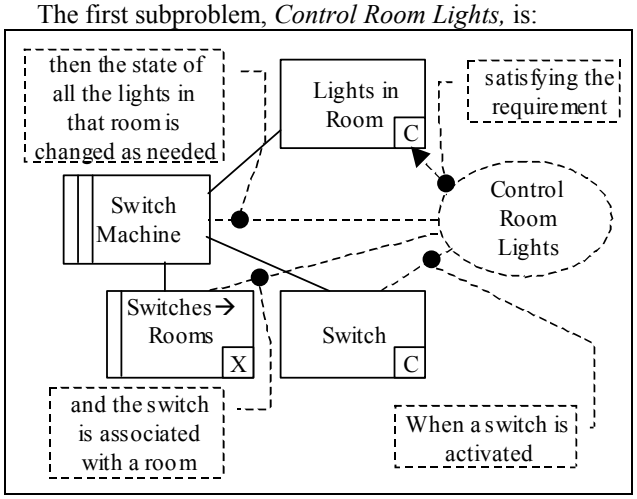
2.2. The Problem Diagrams

The following is the context diagram for the environment. It appears to describe a straightforward commanded behavior problem.

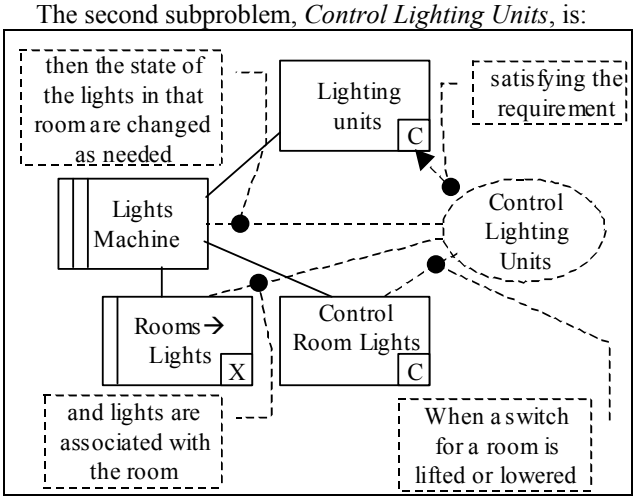


The problem decomposes into two commanded behavior subproblems[1]. The first maps *switch events* to the rooms that they control, using a lexical domain as a *Switches* → *Rooms* map. The second maps *room events* to the lighting units in that room, using a *Rooms* → *Lights* map.

The first subproblem, *Control Room Lights,* is:



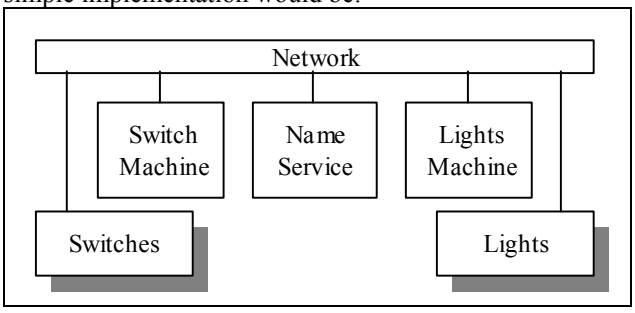
The second subproblem, *Control Lighting Units*, is:



Looking at the diagrams, we see that lifting or lowering a switch causes an event that is a phenomenon shared with the Switch Machine. The machine determines which logical room is to have its lights changed, and is the source of a phenomenon shared with the Lights Machine, as shown in the second diagram. The second machine determines which lighting units are involved, and then is the source of phenomena shared with the appropriate lighting units.

3. A Possible Implementation

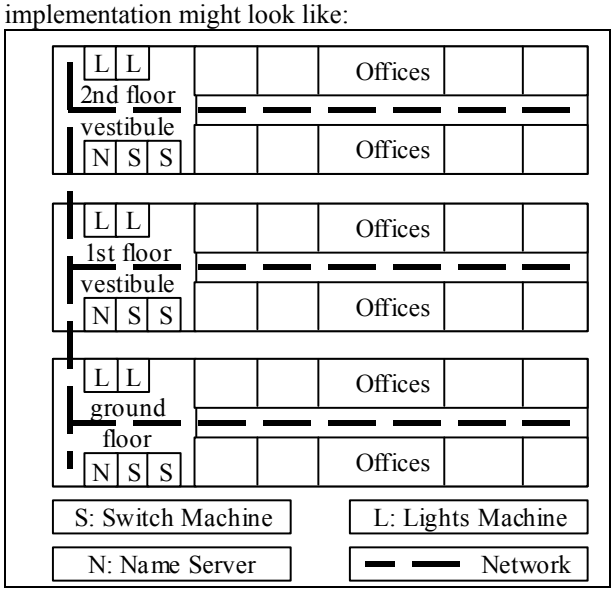
One can imagine constructing this system using a Jini-like distributed architecture [5]. In a Jini-based system, when a switch is actuated it uses a *name service* to find an appropriate service to process the event. Maintaining correspondence with the problem diagrams, the switch will next find the *switch machine*. The switch machine will use its map to determine which rooms need to know about the switch actuation, and then use the name service to find the *lights machine* to contact. A diagram of a simple implementation would be:



If we consider the name service to be part of the network, then the above implementation corresponds very closely to the subproblem diagrams.

[1] The simple workpiece problems needed to maintain the lexical domains are not discussed in this paper.

However, one might choose a different implementation for a larger building. If the building has multiple floors, then for performance we might put switches and lights machines on each floor. To improve reliability, we might put multiple machines of the same type on each floor, where any instance of a machine type can substitute for any other (i.e. introduce redundancy). Such an implementation might look like:



To complicate things a bit more, assume the existence of a logical room consisting of lights in all three of the vestibules.

Assume that the architect specifies the following two rules:

1. A switch on a given floor can select either of the switch machines on its floor, choosing at random. If that machine does not answer, another machine is tried.
2. Either of the lights servers on a floor can control the lights on that floor. The server to use is chosen at random. If that machine does not answer, another server is tried.

Therefore, when a user lifts a *vestibule switch* on the ground floor, the switch chooses either of the *switch servers* on the ground floor. That switch server subsequently must contact either one of the two light servers on each floor, requesting that the lights be turned on.

The problem diagrams shown in Section 2.2 do not express the added complexity of the multiple servers, and thus it is difficult to reason about the system's behavior under certain conditions. For example, analyzing the effects of particular concerns such as initialization, fault recovery, and component maintenance pose problems. Adding explicit connection domain subproblems to the problem can show the missing behavior, but the domains also add significant additional complexity.

4. Extension of Problem Frames Notation

The deficiency in Problem Frames notation exposed by the above example is the inability to accurately specify a *limited many* relationship on an interface. In the example, from the point of view of the switch there are many candidates for the switch machine, but only one of them is to be used. From the point of view of the switch machine, there are many candidate lights machines, where potentially many of them are to be used. These relationships have a form of *cardinality*.

Relationships on an interface are directed. All phenomena have a source domain and some number of destination domains. From the point of view of a source or a destination, there can be from one to N domains on the other side of the relation. Thus, the cardinality of a relationship can be described as follows:

> N(b) → M(c): there are *N* sources of phenomena on an interface where *b* sources are to be considered interchangeable, and *M* destinations for the phenomena where *c* destinations participate.

For convenience, if the parenthesized portion is omitted, it is assumed to be identical to the number that would be in front of it. Thus 1 → N is the same as 1(1) → N(N).

Referring to the more complicated example above, the cardinality of the *switch* to *switch machine* interface is N(1) → 2(1). The left side is N(1) because only one of the N switches participates in a given switch actuation. However, the example specifies that there are two interchangeable switch machines available to the switch, and the switch must choose which one to use. Thus, the cardinality of the switch machine is 2(1).

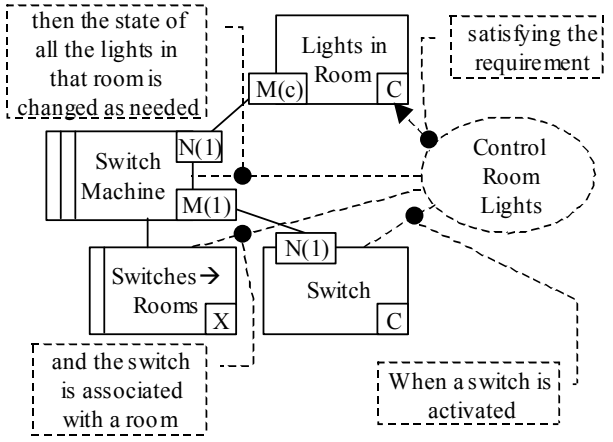
Still referring to the example, the cardinality of the *switch machine* to *lights machine* interface is 6(1) → 6(3). There are six switch machines on three floors, but only one of them can be the source of a phenomenon on the interface. There are three groups of two identical light machines, thus three of them participate as destinations of a phenomenon.

Finishing the example, we see that the cardinality of the *lights machine* to *lighting units* interface is 2(1) → M(M) (or 2(1) → M). Two lights machines can share phenomena with any given lighting unit, but only one at a time. Each lighting unit is an individual, meaning that all M lighting units must share phenomena with the given lights machine.

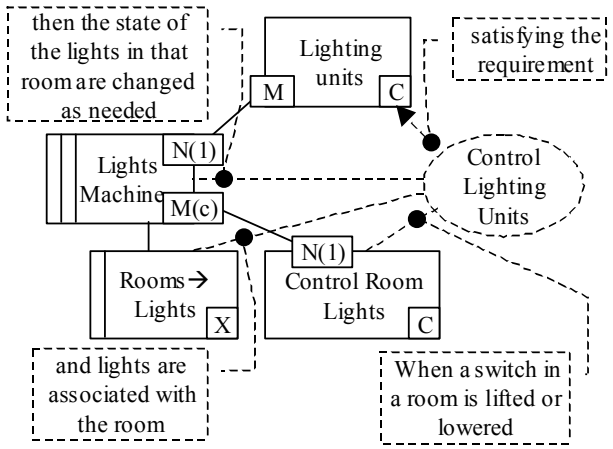
Clearly one would not use such specific notations on a problem diagram unless the numbers are fixed in the problem statement, which is not the case in this example. The *switches to switch machine* cardinality is better

written as N(1) → M(1). The *switch machine to lights machine* cardinality is N(1) → M(c s.t. c≤M) and the lights machine to lighting units is N(1) → M.

Applying these cardinality notes to the subproblem diagrams, we arrive at:



and



5. Conclusions

Adding cardinality notations to Problem Frames diagrams conveys information about how phenomena are to propagate. The engineers responsible for implementing the system would use this information to ensure that the system behaves as desired and to verify correctness in the face of errors, such as partial loss of power and machine failure. Using cardinality avoids the complexity of adding connection domains to provide equivalent information.

References

1. B. Boehm, "Requirements That Handle IKIWISI, COTS, and Rapid Change," *IEEE Computer*, vol. 33, no. 7, Jul, pp. 99-102, 2000.

2. J. G. Hall, M. Jackson, R. Laney, B. Nuseibeh, and L. Rapanotti, "Relating Software Requirements and Architectures Using Problem Frames," in *Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02)*. Essen, Germany, 9-13 Sep 2002.

3. M. Jackson, *Software Requirements & Specifications*. Addison Wesley, 1995.

4. M. Jackson, *Problem Frames*. Addison Wesley, 2001.

5. "Jini Network Technology," http://wwws.sun.com/software/jini/. Sun Microsystems, 1999-2002. (31/01/2003).

6. B. Nuseibeh, "Weaving the Software Development Process Between Requirements and Architecture," in *From Software Requirements to Architectures (STRAW '01)*. 23rd International Conference on Software Engineering, ICSE 2001. Toronto, Ontario, Canada, 12-19 May, 2001.

7. B. Nuseibeh, "Weaving Together Requirements and Architectures," *IEEE Computer*, vol. 24, no. 3, March, pp. 115-119, 2001.