Considering Cardinality in a Medical Device PnP System

William Spees
US FDA
william.spees@fda.hhs.gov

Abstract

In designing systems of hardware and software, we get more leverage from decisions about cardinality than almost any other choices. When we study an existing system, the fundamental insight we gain is an understanding of the cardinalities of the system. From the cardinalities we can choose (or infer) the basic operation and prioritization of the system, without being distracted by details. This paper will briefly explore some of the cardinality-sensitive attributes of Medical Device Plug and Play systems.

1. Introduction

A medical device Plug and Play (PnP) system is an ad hoc network of medical devices that is assembled before it is used in a clinical setting. Today, any medical device that can be connected to the PnP system would also be capable of being used as a medical device without being connected to the system. The advantages of connecting a medical device to a PnP system would be centralization of control, temporal coordination of actions, record-keeping, optimization of logistics, and so forth.

PnP systems are characteristically reconfigurable during their operation. Connecting a new device or disconnecting an inactive device should not require that the system be powered down or rebooted. Work already in process in the PnP system should not be interrupted by configuration changes. Configuration changes must not involve the disconnection of any device that is allocated to the system’s current work.

PnP systems can always continue working when the user adds a new device, even if there is a problem such as lacking scripts that know how to do work with the device, inability to recognize the new device and command it to test itself, or device communication failure or inability to pass its self-test.

2. What is cardinality?

At its core, cardinality is a degenerate form of counting, usually expressed in terms of zero, one, and many. Occasionally there is need for more complexity, but even such a complication is straightforward, a variable with minimum and maximum values [1]. Some common cardinalities of an object are “optional” meaning zero or one, “none” meaning zero, “one and only one” meaning one, “zero or more”, and “one or more”.

Cardinality provides a natural way to describe the relationships among components in different kinds of systems. Most systems exhibit familiar cardinalities, such as rows of relational databases, containers and their contents, clients and servers, computers and users. For example, normal cars have a one to four relationship to their wheels; the wheels have a one-to-one relationship with their tires, an egg carton has a one to zero through twelve relationship with its eggs.

Exactly one patient can swallow a particular capsule; a refill of a prescription may contain sixty doses; a care-giver can have many patients; a patient can have many care-givers; children have at most two natural parents.

We use cardinality constraints to provide the quantitative glue that maintains the shapes of systems within expected bounds. Large changes of scale often invert cardinalities. Some people have seen the historic cardinality inversion from users per computer to computers per user.

3. Cardinality and Object-Oriented System

Object Oriented (OO) systems usually allow for objects to be created and prepared for use whenever they are needed. The new object is created and usually attached to a name that the programmer had already prepared to receive it.
When there is only one such name, then associating it with a new object gets rid (in the best case) of any old object with which the name was previously associated.

If the programmer wanted to allow a known fixed number of similar objects, an array would be needed.

If the number should be allowed to grow without (practical) bounds, then another data structure would be needed, such as a vector, a linked list, or a stack. Such structures, called containers, allow storage of an infinite number of objects (for small values of infinity, on the order of a billion or less). Containers allow for long and elastic storage and provide different methods of storing and retrieving objects. If the system establishes a correspondence between real things and system objects, it would be surprising to need larger numbers of objects than could be accommodated by a container.

To relate OO programs to cardinality, zero-or-one (also called optional) appears as a name (sometimes called a reference) which can be empty (and have a testable value called null) or have an object as its value. One-and-only one appears as a name for which an object is created on its first appearance. Zero to many can appear as either an array (for a many that is just a few) or a container (for many that could be very many). “One to many” appears similar to “zero to many,” but the program assures that there is always at least one value.

4. Entities and Relationships

Cardinalities promote clear analysis of relationships. Components of a system with cardinality of one and only one are the central features of the system. A one and only one component may be associated with other one and only one components, but in a certain sense, they are really all part of the same single super-component. Other components will associate with the super-component, making that relationship a one-to-many relationship; these are typically components that the super-component creates, or that it services in response to the environment or communication connections. Outside the super-component, any of the many components may have one-to-one, optional, or one-to-many associated components.

The build-up of the software and the hardware of a system will always recapitulate the cardinalities of the system.

If you are designing the system, this is a fact that you cannot fight, and if you are trying to analyze a system, this will suggest what was important to the designer, and it will be fairly easy to analyze the cardinality from the nature of the collections, arrays and scalar (single) objects in the design.

5. Reading the Cardinality Trail

If the design shows that there is a super-component that has many peer user interfaces, it suggests that the system is built around a mainframe computer, a hospital information system, or similar fixed system. The user’s interface to the system is susceptible to the threats that can take any centralized system down, threats against the user’s connection to the system, and the threat of the user’s being disconnected for some time without being aware of it.

If the design shows a single user interface, then if the user of the system will always be present when the system is operating, there is an excellent chance of discovering a system failure as soon as it occurs.

If the design is a monitoring system for several patients, say, intensive care patients, or medical evacuees on an aircraft, that will be clearly shown by the cardinality.

If the design shows a central user interface fanning out into a cluster of other user interfaces, that suggests a system where an overseer is able to keep an eye on multiple care-givers (or perhaps that multiple patients are able to control their own therapy, for example, with a patient-controlled analgesia (PCA) pump. Looking at the other associations, you could distinguish between a central overseer and an instructor whose actions could be observed by multiple students. How would these last two similar designs likely differ?

As designer, your cardinality choices allow you to channel the expansion of the PnP system during its operation, as components are added. You can choose cardinalities that drive down (or up) the clinician’s workload, that maximize the number of patients connected, or that favor the utilization of devices.

As an investigator, you can easily infer cardinality from the data structures and whether the objects are allocated statically at compile time or dynamically at execution time. You can identify the central super-component and see what components can grow quickly. Examination of cardinality can indicate some potential points of vulnerability to common cause failures.

6. Reference