Implementing Characteristics nugget

Created by John Reilly

When using and implementing the CharacteristicSpecification/CharacteristicValue pattern a number of questions often arise, including

1. When should the attributes (characteristics) be modeled explicitly versus using the pattern?

2. If characteristics are used, is there anything else that needs to be considered from a modeling perspective?

3. Are there any performance issues that should be considered?

This Golden Nugget addresses these questions.

Explicit modeling should take the following considerations into account

- Adding/removing/modifying attributes changes the model
- Attributes are visible
- Attributes should be stable and well known
- Logic associated with attribute
- Can start with this technique to identify characteristics

Characteristics should take the following considerations into account

- No changes required to the model when adding/modifying/removing attributes
- Hides attributes
- Dynamic attributes- ones that are not known at the time of constructing the model
- Informational attributes – ones that are not used in logic

It's about choices. There is nothing wrong with explicitly modeling new types of entities (specifications and entities). The considerations above should help make the choice. Also, note that it is often convenient to start out using explicit modeling as this is a way to document the attributes and their properties. Information modelers often refer to this type of modeling as the construction of a business object model and keep a historical copy of it to be used when populating instances of characteristic specification entities. Note that the explicitly modeled entities would not be present in the information model if characteristics are chosen to support the entity and its attributes.

There are other considerations that need to be taken into account. The Information Framework does not contain entities that support the dynamic design of user interfaces, such as web pages. To more completely support the CharacteristicSpecification/CharacteristicValue pattern the framework should be extended to provide this support. No user wants characteristics to be randomly placed on a user interface! To help model these requirements, think of the properties that are specified when designing a user interface, such as position, label, prompt, length, and so forth.

Another consideration is modeling behavior-related entities so that code associated with a characteristic can easily be added to an application.

From a performance perspective caching the CharacteristicSpecification entities in memory should be considered. This avoids having to navigate a database when dynamically constructing a user interface or using the CharacteristicSpecification entities to support any other functionality. And, to support user queries and to ensure proper performance some key attributes, such as name and unit of measure, should be considered for de-normalization.

Denormalization is the process of attempting to optimize the read performance of a database by adding redundant data. In the example above, name and unit of measure attributes would be added to the implementation of CharacteristicValue in a database. More information on denormalization can be found at http://en.wikipedia.org/wiki/Denormalization or by searching for the term "denormalization" on the web.

Another view is that the relationship between CharacteristicSpecification and CharacteristicValue is being implemented "by reference", as opposed to "by value". In this case the reference (foreign key) is the name. This view may consider the "name" attribute the key of CharacteristicSpecification, but a foreign key can be created for any attribute.

There may be other considerations that are specific to a given use of the pattern, but these are ones that are typical to any use of the pattern.