Occurrence - A Knowledge Layer Pattern

NIGEL DAVIS, Ciena corporation
CHRISTOPHER HARTLEY, Cisco Systems

Consider a model that represents things in terms of classes, their properties and their interrelationships. In a realization, there are various distinct instances of each class. In most realizations there are many repeated arrangements of the instances of classes.

When deriving a solution for a specific uses case from a class model, it is necessary and beneficial to represent these arrangements and hence to represent arrangements of cases of use of each relevant class. There may be several cases of use of any particular class within an arrangement. Each case of use of a class in an arrangement is neither a class nor an instance of a class. In this paper the case of use of a class is called an Occurrence of the class. This is specifically an occurrence of a thing in space as opposed to occurrence of an event in time.

Each arrangement can be represented as a model. The arrangement may result from application of a combination of constraints from the environment, from the usage, from policy, from architecture etc. An arrangement may be used many times and in many distinct solutions. Interpreting Fowler, it is apparent that this sort of model should be considered in the Knowledge Level where the Knowledge Level consists of types of classes.

The use of Occurrence in an arrangement, is the focus of this paper. This pattern has proved useful in the representation of the capability and application of various network structures and physical assemblies. Having started out to solve a particular problem, it now appears that this pattern potentially has a very broad application. The pattern points towards an approach to modeling using focusses as opposed to partitions and to the dissolving of the distinction between classes and instance leaving only Occurrences with degrees of specificity.

The paper sets out an example of a use of Occurrence to illustrate some of the rationale and then works through a more formal expression of the pattern. The paper explains Occurrence, shows how it can be applied and points out key implications of the pattern.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architectures—Patterns

General Terms: Knowledge, Pattern, Architecture, Information, Instance, Metamodel

Additional Key Words and Phrases: Occurrence, Specification

ACM Reference Format:

Davis, N. and Hartley, C. 2020. Occurrence - A Knowledge Layer Pattern. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 27 (October 2020), 14 pages.

1. PATTERN ANALYSIS – OCCURRENCE PATTERN

1.1 Motivation

During the modeling work by [OIMT] in [ONF], several challenges were encountered. These appeared to initially be separate, but as solutions were developed there appeared to be a common pattern. This pattern has proved to be beneficial when modeling telecommunications networks for Control. The motivation is best illustrated by an example from the [OIMT] work.

A telecommunications network device is often built from a chassis with a backplane and slots where an instance of network device will have a combination of different types of module as in the figure below.

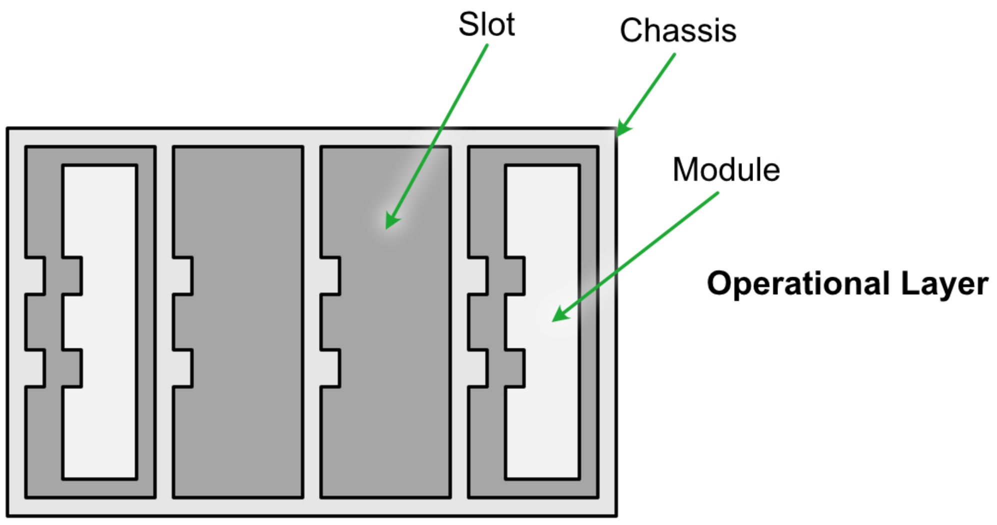

Fig. 1. Physical Inventory example.

Each slot is capable of taking one module from a range of types. Different slots may have different type ranges. Many combinations of modules are valid, but some combinations are not allowed even where the slots would support the combinations. The modules are complex, and each module has complex functional capability where some (and sometimes most) of that capability is emergent from running software. The modules operate in conjunction to form the overall function of the telecommunications device. The devices are built into networks where many network functions are emergent from the specific arrangement of devices. When modeling a telecoms equipment there is a need to represent the varieties of equipments that can be installed in the slots of a specific type of chassis.

A controller of the network of devices benefits from uniformity of representation. This leads to a model of classes where are each class is relatively general. In the model all physical things (Chassis, Module etc.) are represented by the Equipment class and all slots are represented by the Holder class. There are various basic techniques for "specialization" but these do not deal well with the combinatorial variety, with the rule patterns and with emergent behavior.

To further illustrate a key aspect of the challenge in a more common context, consider a house design where an estate of many identical houses is to be constructed. The house design has 12 windows, there are three sizes of window available, large, medium and small. Each house has 4 of each. The purchaser is given a choice of color for the windows in their actual house. The house design shows the windows each referencing a spec and having an unset color property. The representation of window in the house design is neither an instance nor a class.

1.2 Problem

How do I represent the structure of a system design where there are various distinct uses of the same type of part and where, in those uses, the parts may take distinct roles causing distinct constrained configurability? Extending this, how do I express the structure of a system that is to be monitored and adjusted?

1.3 Intent

Occurrence[1] provides a way of stating multiple uses of the same class by allowing expression of distinct and differentiated occurrences of the same class.

1.4 Context

A model of a domain[2] provides a structure of classes that are interrelated where some of the interrelationships have multiplicity greater than one. Where the multiplicity is greater than one there may be more than one instance of the related class in the realization.

In general, there are many instances of each class in a real solution. These instances form arrangements that abide by the model, but that have further constraints often not represented in the model. These arrangements tend to repeat through the solution, either because of some natural law, or as a result of some intention. These arrangements are not easily described in a language such as UML.

The arrangements and patterns of constraints that repeat through the operational layer can be described in the knowledge layer. In the knowledge layer there may be a representation of the potential/opportunity for relating alternative cases of specific classes. This is again a situation where there is a multiplicity greater than one. This is true even where the specific instantiable relationship may have a multiplicity of one as there may be multiple alternative sets of properties for the one related thing. Hence, the semantics of a relationship change from the operational layer to the knowledge layer as in the knowledge layer variety of opportunity is also being represented.

The challenge is especially apparent with composition relationships where:

- There are multiple composed parts each of the same class (multiplicity *)
- There is a variety of options of composed part detail (multiplicity *)
- The option of composed part detail is different for some (potentially each) of the multiple composed parts

The simple composition relationship in the operational model becomes more than just composition in the knowledge layer.

There are several challenging scenarios:

1. When defining the rules for a case of use of a model, where those rules for the case constrain the use of generally applicable classes (this will be called a system spec)
2. When specifying the capability of a thing that is described in terms of the classes of the model (this will be called a capability spec)
3. When describing an example of use of a model in the form of a structure of interrelated model parts (this will be called an example spec)

For each of these situations:

1. The structure is described in a constrained pattern in terms of the classes of the model
2. There are usually several uses of the same class in the structure
3. The multiple uses of a class are differentiated but they are not fully stated instances:
    a. Some properties of the classes are single fixed value, but others aren't.

[1] This is specifically an occurrence of a thing in space as opposed to occurrence of an event in time.

[2] It is important to distinguish this form of model from a model of software behaviour. The model of a domain describes entities in the domain and how they are related. The controllers talk about these entities, they do NOT talk to them. A controller may express the current state of a structure of entities or request changes to be made to a structure of entities by talking about the entities or about constraints bounding desirable structure outcomes. Hence, there are no methods on the classes. Properties settings are governed by constraints (some as tight as a single value) in requests for change from a client system to a provider system.

   b. Where the property is not a single fixed value it may be the unconstrained definition of the class or may be constrained to some degree due to the situation
4. Some properties for a specified group of cases may have a single fixed value for all members of that specified group

The term "model spec structure" will be used to cover these scenarios. These scenarios appear in the knowledge layer.

The pattern discussed in this document applies where there is a need to state distinct and differentiated cases of the same class (or model structure fragment of multiple classes) in some model structure where that structure is not of fully expressed instances. This appears especially in a model spec structure (in the knowledge layer) where the purpose is to express opportunity for alternatives.

1.5 Forces

Considering techniques currently available and the dimension of the family of problems being targeted:

- Creating subclasses to introduce combinations of constraints (partitioning of semantics forming a taxonomy)[3]:
    - Will create a high number of subclasses
    - Will lead to an extreme depth of subclasses
    - Will probably lead to futile debates about which ordering of sub-classing is correct
    - Will lead to challenges partitioning combinations of constraints
    - Will lead to partition transition challenges as the solution evolves
    - Will not deal adequately with the complexity of constrained relationships. As relationships between classes are inherited, but subclasses only have specific relationships, all relationships will need to be between leaf sub-classes, however, there is no fundamental leaf
- Using conditional composition to add capabilities to a skeleton class (grouping of semantics forming various defined sets):
    - Will cause the parent to have to change each time a new conditional composition is added
    - Will not deal adequately with the complexity of constrained relationships.
- Using a decoration technique where the decorating addition references the part it is being added to
    - Will not, by itself solve the constrained relationship problem

At one extreme a model can have one class, "Thing", with a self-join. Thing encapsulates all possible semantics. Constrained Things (explicitly defined semantic subsets) are used to represent any aspect of any domain. Thing_Spec constrains Thing.

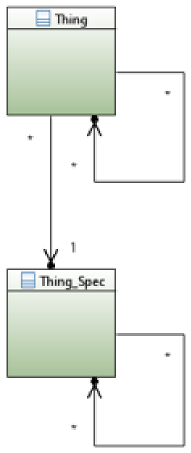

Fig. 2. Thing and Thing_Spec

[3] The sub-bullets are understood from many years of experience in the modelling of telecommunications systems for the purpose of control and management.

At the other extreme, every instance is represented by a distinct class where the classes are formed by recursively narrowing the semantics embodied in the class name (every differentiating property is encapsulated in the class name). This is essentially the formation of a deep taxonomy by repeated partitioning. In this case, a composition only ever has a multiplicity of one as when there are multiple instances, each has a different class.

There appears to be a need for balancing of Sub-classing (partitioning of semantics forming a taxonomy) and semantic constraints of existing classes (forming subsets of semantics by constraining/fixing values and relationships).

1.6 Solution

1.6.1 Summary

Take a skeleton model of a domain and form system spec structures from Occurrences of classes from that model where each Occurrence is enriched by decorating it with constrained properties from a referenced class spec, where that spec details properties for the case of use of the class and where the system spec structure is formed by specific use of relationships abiding by the skeleton model.

1.6.2 Explanation

A semantic domain of interest (e.g., physical equipment) can be modeled somewhere between the two extremes noted in the Forces (i.e., between Thing class and class per-instance), in terms of constrained types of Thing, each formulated as a class, with interrelationships appropriate for the domain (e.g., equipment and holder)[4]. The domain model focusses on representing things in the domain in as abstract/general a way as possible whilst remaining within the semantic boundaries of the domain and emphasizing all key structure. As with Thing, the classes of the domain model encapsulate all semantics at their level of constraint. The domain model is a skeleton structure with only properties that relate to the structure and fundamental semantics.

For each use case, each class of the domain model can be enriched using decorating specs[5] to illuminate relevant properties that are valid within the semantic space identified by the class.

The degree of decoration and depth/precision of specification of the properties depends upon the use case. At an extreme the representation of a run time instance of a class of the domain model is simply a domain model class constrained by a very specific and very tight spec.

1.6.3 Observations

- There will usually be multiple occurrences of same class in a system model structure
- Occurrences of a class in a system model may each have the same spec or each have a different spec etc.
- A spec may be used for occurrences in different system model structures
- System model structures may be assembled together to form larger system model structure
- The specialization in the domain model should be minimal
- The domain model is beneficially formed from DDD Aggregate structures where the root of the aggregate is the "globally accessible" class used in the system model
- The spec provides constraints on the elements of the Aggregate and may augment the aggregate with finer detail
- The implementation model and "instances" are in terms of the skeleton model classes appropriately refined by the specs as governed by the role for the instance in a corresponding system model structure
- The system spec structure abides by the domain model but is not an instantiation of that model. The domain model is a metamodel for the system spec.

[4] As will become clear, these classes are actually Occurrences of Thing.
[5] A spec is a structure of definition providing properties and constraints

- The domain model is also a metamodel for the class specs.
- The class specs will express properties/attributes that were not present in the domain model.
    - In the ONF work, these are derived from work of other standards bodies where these bodies work at a finer detail.
- Many class specs will also exhibit occurrences of subordinate parts
    - The class spec is essentially a system spec at a finer granularity

1.6.4 Occurrences in the Spec

In general, in the spec structure, introduce occurrence opportunities wherever there is a multiplicity:

- Of parts (composition) where the parts are of the same class but where each has the potential for distinct constraints (each is potentially of a different "type" of that class)
- Of related things where each related thing is of the same class but where each has the potential for distinct constraints (each is potentially of a different "type" of that class)

1.6.5 Illustrating the Occurrence pattern

The Occurrence pattern is best illustrated via a sequence of UML sketches.

The figure below shows a simplistic sketch of a Domain model, "Specific class model", that will be used as a basis for the illustration. It has two classes with a simple association between them (where Class1 knows about Class2) and a generalized Spec[6] class in the knowledge layer (where both Class1 and Class2 reference the spec).

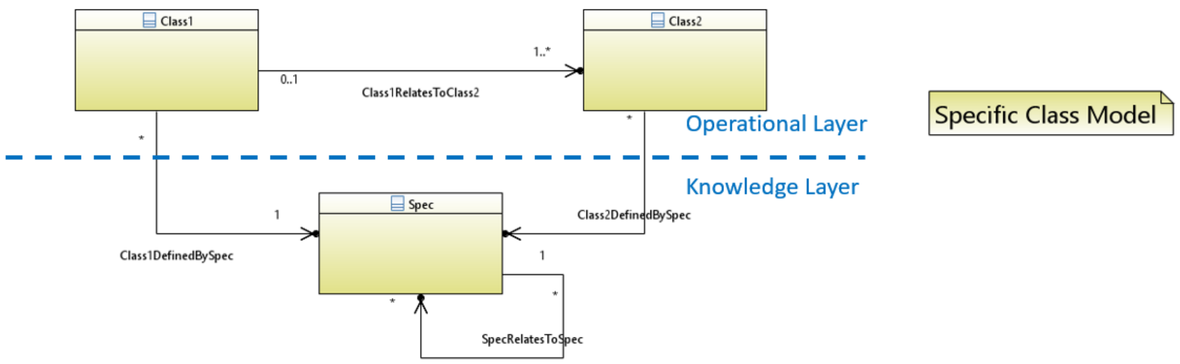

Fig. 3. Specific Class Model

This simple domain model can have the Occurrence pattern applied to form a (rough) system model as shown in the figure below. The sketch illustrates that there will be Occurrences of the Classes and that each will have an Occurrence of a specific Spec associated. It also illustrates that Class1_Spec_Occurrence will necessarily have some role in defining the allowed Class2_Occurrence.

In this specific example, only a partial expansion has been applied to start to illustrate the idea of Occurrence. As the specific property definitions of one Occurrence differ from those of another Occurrence they have to be explicitly stated. As the properties are mainly[7] of attribute definition form, the Occurrences appear to essentially be classes. However, the intention is that the classes are NOT instantiated. In the realization, there are only instances of Class1, Class2 and Spec.

[6] An instance of a class acquires properties (some may have to take a specific single invariant value) and rules from an associated spec. This is not covered in detail here.

[7] Some properties will be fully resolved to a single value. These can be considered as highly constrained definitions and with this consideration, "mainly" becomes "all".

In an instantiation, the Spec provides definition of properties in the specific instance of a Class. The Class definition provides the boundary on a semantic space where the spec then fills in relevant details in that space.

The Occurrences are partially specialized variants of the Class where the specialization provides a boundary on a semantic space that is smaller than that of the original class. Hence a Spec that applies to an Occurrence of a Class must be compatible with that smaller semantic space and must not attempt to define things outside that space.

The Occurrences are chosen to have specific roles in the usage of the model where each role utilizes a narrow variant of the original Class.

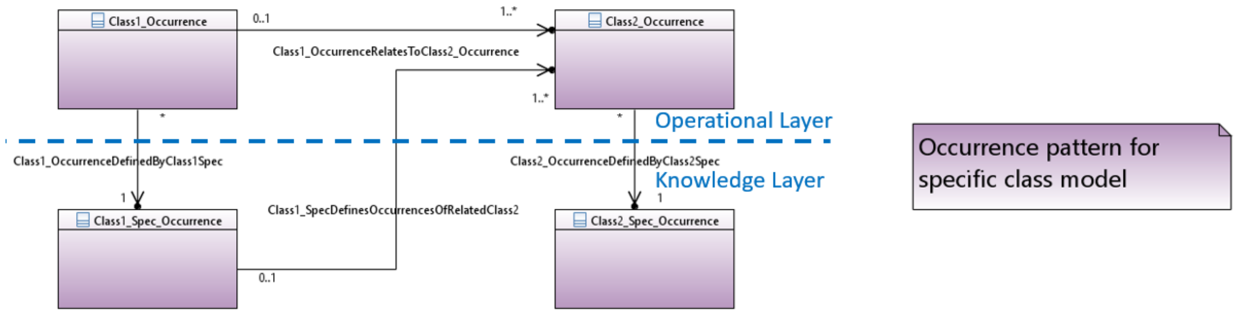
Class1_Occurrence
0..1
1..*
Class2_Occurrence
Class1_OccurrenceRelatesToClass2_Occurrence
1..*
*
*
Operational Layer
Class1_OccurrenceDefinedByClass1Spec
Class2_OccurrenceDefinedByClass2Spec
Knowledge Layer
1
1
Class1_Spec_Occurrence
Class1_SpecDefinesOccurrencesOfRelatedClass2
Class2_Spec_Occurrence
0..1
Occurrence pattern for specific class model

Fig. 4. Occurrence for Specific Class Model

The figure below shows an expanded form of application of the Occurrence pattern where all opportunities are illustrated (in red). The figure is an illustrative example extending the example above.

In this specific Occurrence Opportunity model, a specific single Occurrence of Class1 is shown where this specific Occurrence is only allowed to have one specific Spec Class1_Spec_OccurrenceX. There may be many other Spec Occurrences compatible with Class1 in the environment, but for this specific Class 1 Occurrence, only one is allowed.

There may be many instances of Class1 that abide by Class1_OccurrenceX in a running solution

Class1_Spec_OccurrenceX happens to allow for up to three associated Class2 Occurrences where one is mandatory and two are optional.

The mandatory associated Class 2 Occurrence must abide by either Class2_Spec_OccurrenceA or Class2_Spec_OccurrenceB, etc.

The figure also shows a more specific Example (in orange) that abides by the Occurrence Opportunity model. There may be many instances of Class 1 and Class 2 in an actual realization that follow this Example. There may also be many instances of Class 1 and Class 2 in the same actual realization that follow some other pattern of usage specified by another specific Occurrence Opportunity model. In some cases, these Occurrence Opportunity models may intertwine and that intertwining may be expressed in a broader Occurrence Opportunity model.

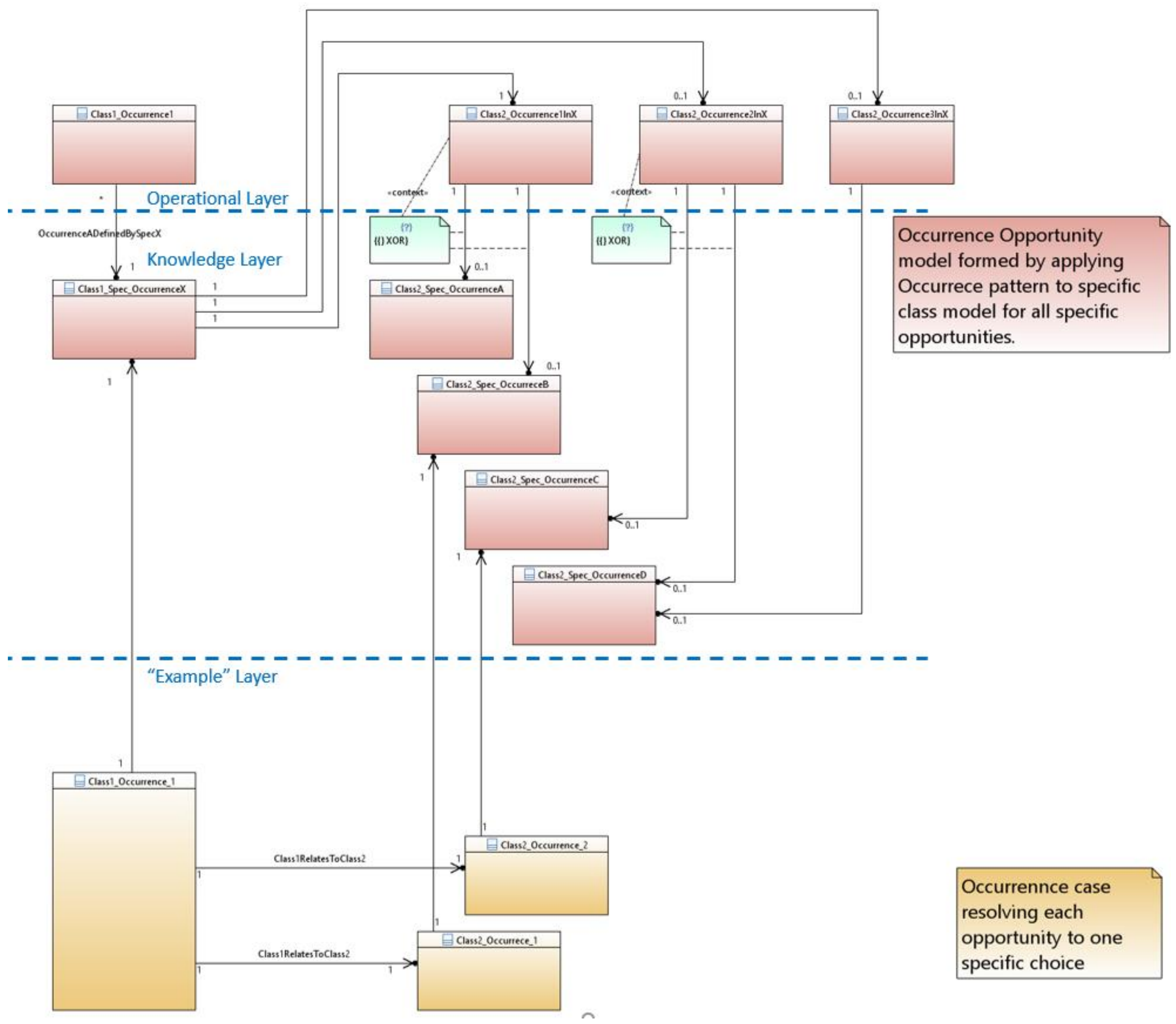

Fig. 5. Occurrence Opportunity and Occurrence Case

Clearly, the Occurrences are neither instances nor classes in a traditional sense. The Occurrences are all partially resolved forms of the Classes shown in the Class model but are not full instances.

An aspect of the realization of the representation of Occurrence would involve Stereotypes of Class and other UML artefacts.

When using the pattern, the associations may be reversed, and the model may be refactored to remove irrelevant opportunity.

The brief illustration above using the sketch models shows the purpose of the Occurrence pattern. The figures show that by applying the Occurrence pattern to a model, constrained opportunities for variety can be represented and examples of specific constraints can be formed.

As noted earlier, a realization only has instances of Class1, Class2 and Spec. The client system specializes the usage of Class 1, Class2 and Spec on-the-fly as it discovers Spec Occurrences and Occurrence Opportunity models from the provider systems. Using these Spec Occurrences and Occurrence Opportunity models the client system can interpret existing instances and request the building of new instances as appropriate.

It is recognized that the class model shown at the beginning of the discussion above could also actually be an occurrence model perhaps derived from the general component-system pattern model[8] such that class1 is actually component-occurrence1 and class 2 is component-occurrence2 etc.

The pattern and derivation method can be used in a layered fashion starting from an extremely general model and working towards a highly specialized model. As noted earlier, it even appears that the UML instances are really actually very tight Occurrences and are not distinct forms.

1.7 Consequences

The pattern provides rules defining valid instantiation. Realizing the pattern will enable uniform description of capability and opportunity.

The pattern appears to bring significant complexity when view along with the class model, but it actually exposes the inherent complexity of the problem that is obscured by the simplistic representation of the class model.

The pattern appears to provide an appropriate basis for a highly flexible coding approach as it provides a basis for dynamic model construction. Appropriate use can also support basic systems with tightly coded models and highly versatile dynamic modeling systems using the same uniform approach across the spectrum.

It appears that current model tooling is not sufficient to support this approach.

1.8 Example

1.8.1 Physical Inventory Operational Model

This paper will use a simple example of networking physical inventory, where there is a Chassis (an Equipment) that has slot positions (Holders) that Modules (Equipments: cards, circuit packs etc.) can be mounted in.

For this paper only the simplest of cases is used, where a Module occupies one Slot and each Slot may be empty or may hold one Module. In addition, the class model has been overspecialized via classification/inheritance to remove complexity of Occurrences at every level. In the ONF work the class model for physical devices only has Equipment and Holder and these are refined via Occurrence. Here, Equipment is specialized to Chassis and Module via classification.

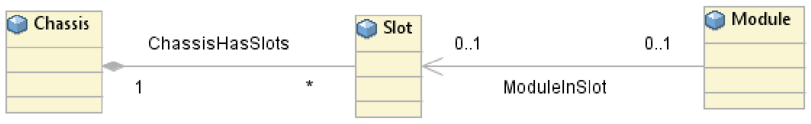

Fig. 6. Physical Inventory model.

[8] This Component-System pattern is only described in the [ONF CIM]. It will need some expansion to help describe the full application of the Occurrence pattern. The Component-System pattern can be seen implicitly in other work in the industry (e.g., at the core of the [OASIS TOSCA] meta-model).

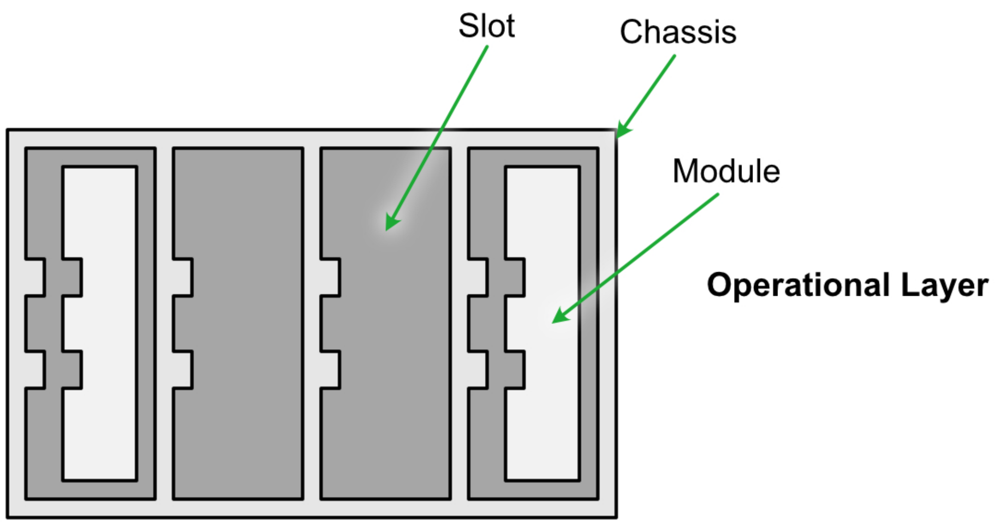

Fig. 7. Physical Inventory example.

The outer box represents a Chassis, a frame that has Slot 'holes' (dark grey) where cards or modules can be plugged in. The Modules usually plug into connectors at the back of the Slots, and the diagram represents this using a simple 'square wave' pattern on both the Slots and the Modules.

1.8.2 Physical Inventory Knowledge Model

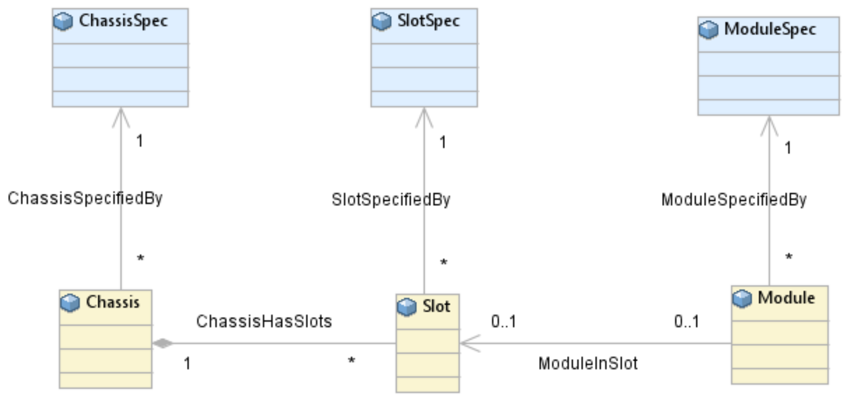

Fig. 8. Physical Inventory knowledge model – a starting point.

Here we follow the convention of [Coad] in showing the specification classes in blue fill.

A starting point for the knowledge layer is to map the operational classes one-to-one with the operational layer classes and link each of them via a 'SpecifiedBy' association. Again, the example has intentionally over classified to simplify the description.

There can be a number of variations used, but for simplicity it is assumed that each operational instance needs a related Spec, and that there can be many operational instances per Spec.
Also, it makes sense that the operational instances know about their Spec, but that the Spec doesn't keep references to all of the related operational instances.

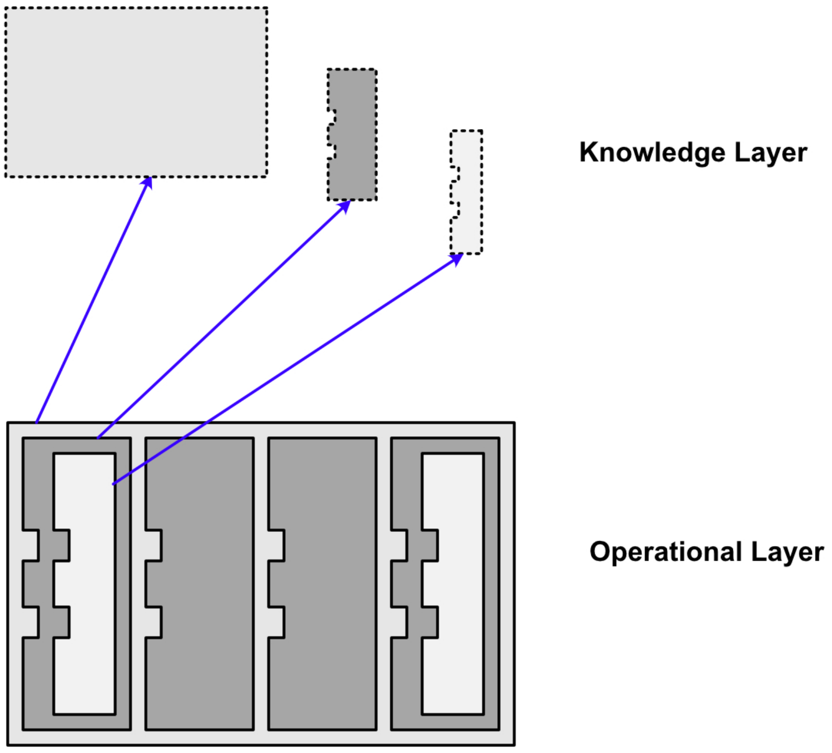

Fig. 9. Physical Inventory mapping example. Knowledge layer concepts are shown with dashed lines.

What is not so straight forward is how to map the operational associations to the knowledge layer.
A simple approach is to 'multiply' the association end multiplicities.
For example, "a Module occupies one Slot and each Slot holds zero or one Module" could be mapped to "a type of Module can occupy many types of Slot and each type of Slot can hold many types of Module".
The issue is that a module may be able to work in slot 1 of chassis type A but not in a similar Slot of another chassis type. It may also work in slot 2 of chassis type A but not in slot 3 of chassis type A.

Creating SlotSpecs for every ChassisSpec will make maintenance of the knowledge layer catalogue unduly complex. This approach would also lose significant reuse opportunities.

The solution is to understand that the composite association can't map directly into the knowledge layer and that an additional class needs to be added into the knowledge layer.

For historical reasons we decided to call this an "OccurrenceSpec" class.

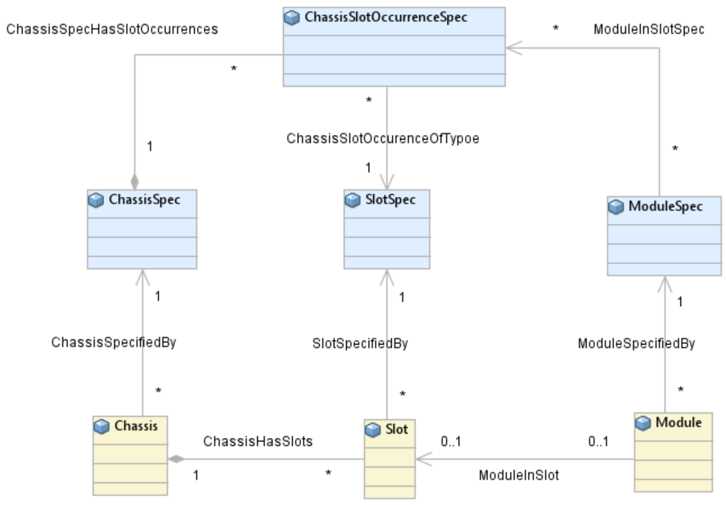

Fig. 10. Physical Inventory knowledge model – with OccurrenceSpec class added.

Note that as shown in the figure above, the ChassisSpec uses a composite association to the OccurrenceSpec class which then refers to the SlotSpec class. Also, that the ModuleSpec class doesn't directly relate to the SlotSpec class, but rather via the OccurrenceSpec class.

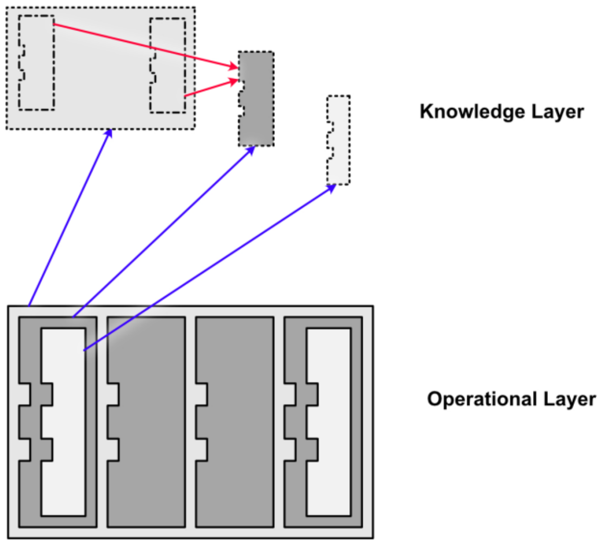

Fig. 11. Physical Inventory mapping example. Knowledge layer 'occurrence' concepts shown with dash lines.

1.9 Related patterns

We are essentially using some variant of AOM in our specification approach "The first step to the AOM model is to define the Type of the object that we wish to add metadata to. This class holds all possible attributes for an entity object," We define rules and explicit properties in our spec classes.

We have explored Type Square and effectively we use Rule Objects.

2. REFLECTIONS ON THE OCCURRENCE PATTERN

What started out as a solution to a particular model issue, that had been unsolved for about 20 years in our work area, with representation of networking protocol stacks, turned out to be the basis for a solution to a seemingly unrelated issue that had also resisted resolution relating to physical inventory capability (the example used above).

However, when attempting to set out actual solutions using our initial method, we found that we needed to deal with multiple uses of a type in a single structure (we had a variety of temporary terms for this including case of use of a class and semi-instance of a class - we eventually settled on the bland term Occurrence.). This multiple use challenge appeared both inside the specifications of a specific type and in the specification of a system of use of the types.

As we have become more familiar with this pattern through use, we have found that the base pattern with some variation can be applied in more and more areas of our work in the ONF (developing a standard network management model) [ONF CIM].

This led to a recognition of a continuum of refinements. The Occurrence appears to be the primary form in that continuum.

We are modeling in a space of functional things and hence tend to think in terms of Components and Systems. The Component-System pattern is essentially a generalized from of that structure that can be applied fractally [ONF CIM].

The Occurrence represents any degree of narrowing from the fully flexible capable component to a fully defined “instance” (which is not really an instance either).

There are other scenarios that also appear to have the same challenge and yield to the same solution. These are not covered here but experimenting with the pattern described here suggest it is broadly applicable in the knowledge layer and is also applicable in the operational layer.

The hope is that in sharing this pattern that it will help others in their quest to produce powerful, versatile, simple and elegant domain models.

2.1 Forces in more detail

So, there appear to be two approaches to increase in specificity

- subclassing which increases the specificity by adding implicit property constraint such that each subclass is differentiated by invisible property values (that get promoted to be encoded in the name).
    - Subclassing starts with a class devoid of properties and with limited capability and adds capability/properties to increasingly constrained cases. This is essentially via a technique of partitioning.
- Subsetting which takes each visible property and constrains its value
    - Subsetting starts with a fully capable thing and produces increasingly constrained things that are only distinguished by values of particular exposed properties. Subsets can overlap

Although traditional modeling tends to focus on formation of a deep inheritance hierarchy (a taxonomy) to try to distinguish cases, for any practical model, there is still a need to deal with Occurrences; There is still a need to distinguish each use of a class in a specific structure.

Application of the Occurrence pattern reduces the need for inheritance as it enables constraints to be applied in a subsetting approach.

REFERENCES

"Policy-Level Specifications in REA Enterprise Information Systems"
http://www.msu.edu/user/mccarth4/typesPaper.pdf

[Larman] Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)
ISBN-0131489062

[Fowler] Analysis Patterns – Reusable Object Models, by Martin Fowler
ISBN 0201895420
http://www.martinfowler.com/

[Evans] Domain Driven Design – Tackling Complexity in the Heart of Software
by Eric Evans
ISBN 0321125215

[Coad] Peter Coad, Eric Lefebvre, Jeff De Luca, Java Modeling in Color with UML, Prentice Hall PTR, 1999
ISBN 013011510X

[Coad Letter] The Coad Letter: Modeling and Design Edition, Issue 82, Description Class Archetype
http://bdn.borland.com/article/29672

[Specifications] by Eric Evans <evans@acm.org> and Martin Fowler fowler@acm.org
https://martinfowler.com/apsupp/spec.pdf

The Type Object Pattern
Ralph Johnson Bobby Woolf
http://www.cs.ox.ac.uk/jeremy.gibbons/dpa/typeobject.pdf

[TypeInstanceHomonym]
https://www.martinfowler.com/bliki/TypeInstanceHomonym.html

[ONF CIM] https://www.opennetworking.org/wp-content/uploads/2018/12/TR-512_v1.4_OnfCoreIm-info.zip
[OASIS TOSCA] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[OIMT] https://wiki.opennetworking.org/display/OIMT/OIMT

[ONF] www.opennetworking.org

Received October 2020