Tomorrow Never Knows

Jun 26, 2001
Masao Tomono, Terry Fujino, Neil B. Harrison

We use whatever facts we have as knowledge. In addition to the facts, we make assumptions based on the facts, and treat these assumptions as knowledge. Such knowledge is effective when we adapt to the environment, even though it is not certain. For instance, we can do a conversation smoothly, by doing *conjecture* of the sense of values of the partner. Even when the *conjecture* may fail, it gives us the opportunity to try to know each other and advance the conversation smoothly after all. This is similar even in the conversation between human and systems. This pattern language describes how we profit from such conjectures.

Since the pattern language handles relatively abstract matters, we put the Context and Resulting Context in a form of examples and we use stock keeping scenarios as an example case to explain its context in more concrete manner to help the readers understand. Note that the focal point of the patterns is not about stock keeping, but about the way to cope with uncertainty, or our ignorance.

Chapter1 Conjecture

Let's start from the simplest form of the conjecture. The patterns described in this chapter show how to make assumptions based on complete, but difficult to derive, information.

1. Quick'n Dirty Shortcut
2. Presumed Actual
3. Caught Red-handed
4. Periodic Inspection

1. Quick'n Dirty Shortcut

Example:

Take a look at a warehouse. There is a huge amount of stock in it, and many articles of each item[1] are carried into and carried out of it continuously. The general manager of the warehouse needs to know the stock amount in the warehouse but counting them up takes great time. It is usually impossible to count the stock, because articles are moved so frequently that the count would be inaccurate. We can not stop the shipping activity every time we need to know how much stock we have, because this would disrupt the primary purpose of the warehouse.

Context:

Economically speaking, not only monetary but also time, efforts, and cognitive aspects, one can adapt an estimated or approximated value instead of calculating an actual or precise information concerning a matter of interest, as long as those estimations or approximations are useful.

Problem:

We need timely information, but frequent calculation of the information is expensive, painful, and/or time consuming.

Force:

- The matter of concern is observable. Lack of knowledge about this value can cause a problem but small a discrepancy is acceptable.
- The matter of concern is required frequently, as it changes its value by time.
- It takes longer to get the value of the matter of concern than it takes for the value to change.

Solution:

Instead of performing a full calculation every time the value is needed, use an inference rule that conjectures the matter of concern from the fact that is easier to grasp. This inference must be easy to calculate. When it is difficult to infer the accurate value, ignore conditions that have relatively small effects on the conjecture according to the accuracy needed (e.g. in the example, we ignore the loss inside the warehouse to calculate the stock amount from arrivals and shipping data). If the inference rule needs an initial value, then use *Caught Red-handed* or *Periodic Inspection*.

Resulting Context:

Since we keep records about arrival and shipping of inventory of the warehouse, we use these records to estimate the inventory level. We usually keep the value as a running total. This is accurate if there is no loss inside the warehouse.
Because the value essentially becomes a range, handling a value as conjectured might make logic a cumbersome. To simplify the logic, see *Presumed Actual*.

[1] We must distinguish articles of certain item, and item itself. In the scenario an item is somewhat like a description of the articles that have same properties, when an article is a physical entity that can be physically treated, such as move, handle, or ship.

2. Presumed Actual

Context:

Instead of counting up inventories one by one, the stock amount inside the warehouse can be calculated using arrival and shipping information. But we know that the amount we get in this way may not match with the actual amount, because some inventory could be broken or lost inside the warehouse and some records might be mistaken.

Problem:

We want to make use of a conjectured value though we know that it may be inaccurate.

Force:

- It is expensive or slow to calculate the actual value.
- It is expensive to calculate the difference between the conjectured value and the actual value, because it generally involves calculating the actual value.
- In most cases, it is more important to have an approximate value than it is to know how much it varies from the actual value.

Solution:

Set a rule to take steps to correct the discrepancy between the conjectured value and the actual value as much as practical. Then use the conjectured value as if it were the actual value.

Resulting Context:

Because we set a rule to correct the discrepancy when we run short of stock amount of an item (this is an example of *Caught Red-handed*), we usually can use the stock amount value as if it were actual value.

To correct the discrepancy, we can use *Caught Red-handed* or *Periodic Inspection*. They can be used at the same time. If the discrepancy may bring a serious problem, *Periodic Inspection* should be used.

3. Caught Red-handed

Context:

When we were going to ship an item from the warehouse based on our conjecture of the stock amount, we saw that the actual stock amount is running short.

Problem:

We learn the actual value and see that the conjectured value is wrong.

Force:

- It is expensive to calculate the actual value.
- Sometimes the actual value happens to be clear.
- The discrepancy between the conjecture value and the actual value does not cause a serious problem. (This is important.)
- Without maintenance, the discrepancy between the conjecture value and the actual value steadily grows.

Solution:

Correct the value at the point the discrepancy (i.e. the actual value) happens to be clear. If such a point can be defined (e.g. when we run short of the stock amount), make it a business rule to correct the value whenever that event occurs. It may also be possible to correct the inference rule itself at this time.

Resulting Context:

We have a chance to know the actual stock amount of an item when it happens to run short. At that point, we fix the loss of the stock amount of the item and let the stock amount to zero. As we can get the item in two hours by putting a buying order to the vender of it, so running short of the item itself is not a problem.

4. Periodic Inspection

Context:

Because an item needs long time to procure, it would be serious problem when we run short of it when on shipping. As damages and loss inside the warehouse and recording errors accumulating, the conjectured stock amount gradually going to separate from the actual stock amount.

Problem:

When the discrepancy between the presumed value and the actual value becomes clear, it is a serious problem.

Force:

- Without maintenance, the discrepancy between the conjectured value and the actual value steadily grows.
- The cost of getting the actual value of the matter of concern is permitted, if its frequency is low enough.
- The discrepancy between the conjectured value and the actual value may cause a serious problem.

Solution:

Calculate the actual value of the matter of concern at an acceptable frequency and reflect it to the inference rule.

Solution example in a/the context:

We take inventory on every month so that we can grasp the actual stock amount periodically. To make the correction, we might put an imaginary shipping or arrival so to counterbalance the difference between the actual and the conjecture.

Chapter2 Manifestation

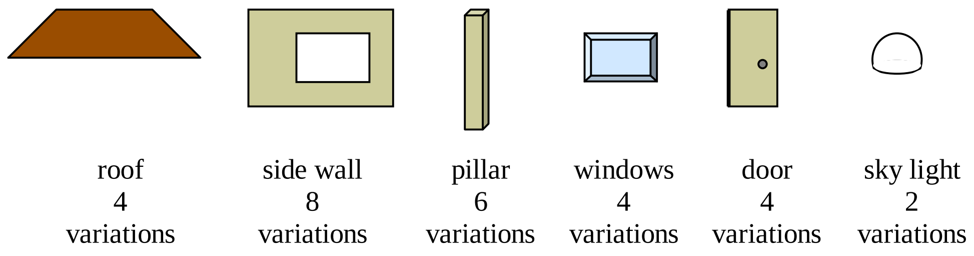
In the previous chapter, we presented several patterns that enable us to cope with volatile information that is difficult to obtain. Since the existence of the data is well known, utilization of approximation of some facets of the data works well and actual values can be left uncertain. In this chapter, we are going to discuss a more complex situation, where the existence of the data itself is uncertain.

5. Montage
6. Reference Has Clues
7. Reference Constructor

5. Montage

Example:

Let's expand the scenario of warehouse. Our warehouse handles a prefabricated shed as one of the items we are providing to our customer. However, since a completed shed usually takes a large amount of space and the variation of shed models is considerable, displaying all variations of shed models is impossible. For instance, the possible combination of each part type (shown in a figure below), such as roof type, side wall type, structural pillar type, windows type, door type, basement type and optional sky light roof type might easily exceed the capacity of human usual recognition. In this case we will have at least 6,144 possible combinations.



Moreover, some combinations of certain part types might be prohibited due to lack of structural strength and/or by some specific environmental conditions. For instance, a large windows should be used in pair with a side wall which has a large window frame, and a settlement of a sky light on a roof may require not only a roof which has a sky light frame but also a pillar just aligned inside the side wall below the sky light itself. Additionally, in case of a heavy snow area, a thick pillar type should be used.

To avoid falling into the so-called *combinatory explosion*, we produce only certain limited number (e.g. most popular combinations and its neighbors) of variations of the shed as a ready-made, and other shed models are treated as custom-orders. Thus, these limitations help sales people to remember the most popular combinations more easily. However, from the product line point of view, there is actually no difference between the ready-made model and the custom-ordered one; we still have to manage the stock for each part that will be used to constitute each model of the shed.

Context:

We have quite a number of product variations, including custom-ordered ones, and stock is managed for each component that can go into the product variations. We produce custom-ordered items in business as usual. It is not possible for human beings to memorize all possible combination of the items we have in hand. And some of the combinations are meaningless or prohibited by regulations or other constraints.

Problem:

There are too many objects or combinations thereof and we cannot exactly figure out the relevant combination of each of them.

Forces:

- No two product models have exactly the same features.
- Different models can be classified by their features.
- Some specific features of a model may be used to designate it.

Solution:

Designate each complete product by one or more of its characteristics. (e.g. a bicycle can be designated by its product line (MTB-Novice), color (red), wheel-size (26 inch), and light-type (battery-driven)). This way, one can understand the relatively complex but relevant combinations of items that produces several product lines of intended things without difficulties, and deduced results are usually very useful and meaningful for ordinary customers who are usually pay less attention to the detailed differences of those produced things.

Resulting Context:

A sales person knows the product line and attributes of the required item, when he accepts an order.

> For instance, suppose the requested shed should have two sky lights, a regular size window in both side wall, and will be settled in a heavily snowing area. This could be interpreted as following possible combinations of items, such as;
>
> - a regular size roof which has two sky light frames (explicitly requested)
> - a regular combination of four walls, two of which are wider than the others, and also each of them has a regular size window frame (presented as a ready-made combination)
> - a pair of thick pillars which have enough strength to sustain both the roof with two sky lights and in case of heavily snow
> - etc.,
>
> In this way, we can figure out the several remarkable characteristics of intended shed, and map them into the corresponding constituent parts items of it, which fulfills the requested and characteristic in some way.

So we made this way of deduction as a rule to designate item (both a finished item and the constituent parts of it) by their own characteristics upon which a specific rule could be applied.

If you use this pattern, see also *Reference Has Clues*.

By the way, we can also designate a product line by its properties, instead of characteristics of components, but you should know it causes another difficulty. See *Rigid Designator*.

Rationale:

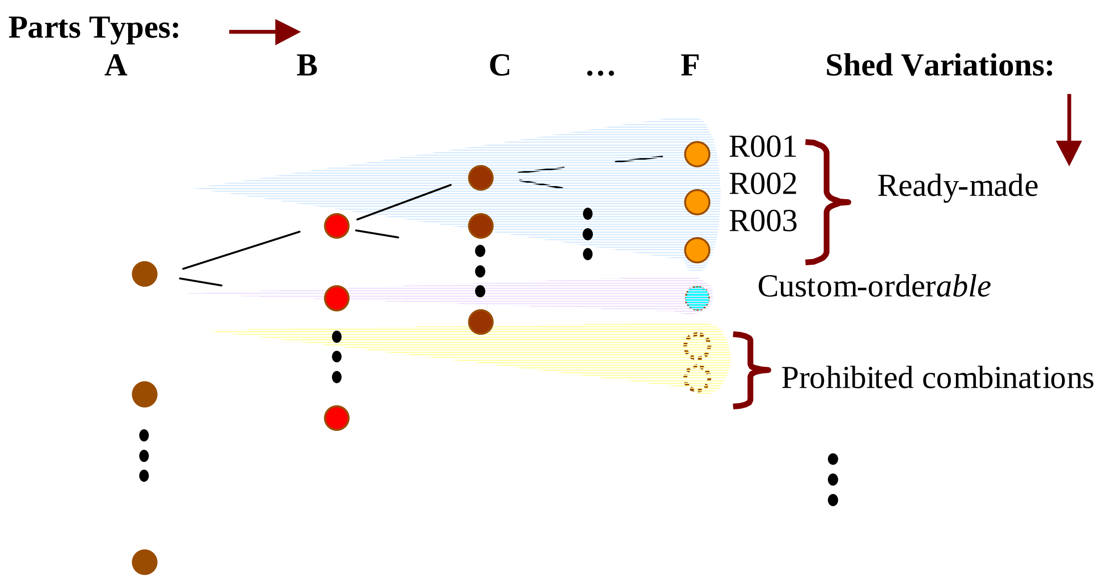
This pattern is useful because we don’t have to remember every combination of components, because we focus on characteristics of the object rather than its components. Without this pattern, we have to deal with the full details of the variations of the element that can be as much as the multiplication of possible values for each feature.

For example, if we have 3 features and 8 possible values for each, then we have to know 8*8*8 = 514 elements without this pattern, where we only need to know 8+8+8=24 elements at largest with this pattern.

6. Reference Has Clues

Example:

Most of orders for the sheds are for the ready-made models, which consist of a limited and relatively easy-to-remember number of variations, so that the warehouse can serve the customers in spite of the limited space and stock of the intended items. However, since special orders are also allowed, the warehouse must provide the information needed to create any shed model. (e.g. a sky-blue circle indicated as '*Custom-order*able' shown in a following schematic Actually, most special orders are minor variants of standard models, and they only differ in color, size, etc.



Context:

Custom-ordered items are seldom requested, but our production management division must still figure out the needed manufacturing steps for them when an order for one of them is accepted. Most of such items are only a variant of existing items and they only differ in color, size, etc.

Problem:

There are too many objects and some of them are seldom used, so that only for the common ones the production method of them are established and repeatedly used. However, it is troublesome to pursue the relevant production method each time a special case arises.

Forces:

- Reference to an object contains important features of the object.
- Each objects do not differs much each other but they are not the same.
- No way to explore and store the all-possible production method before hand.

Solution:

Conjecture and create the target object from the information given by the reference that uses significance values to designate an object (e.g. in the former example, a bicycle object can be created from those values that reference has, that are, MTB-Novice, red, 26 inch and battery-driven).

Resulting Context:

Manufacturing steps of an item can be derived from the attributes that are used to designate the item. So we remodeled our BOM to generate the manufacturing steps automatically from the information that the reference has, and as a result the intended item can be produced in accordance with the manufacturing steps once thus inferred.

Sometimes maintenance for thus inferred a lot of manufacturing steps for objects become troublesome, when minor modifications for them are happened. If so, see *Reference Constructor*.

This *Reference has Clue* pattern can be used in combination with Flyweight, when one needs to identify certain object in consistent manner. In this case, use existing object when there is an object that matches the values of the reference and create new object when there is no object that matches the values of the reference. This will help when you need consistent identification of object.

7. Reference Constructor

Context:

We have remodeled our BOM to derive the manufacturing steps once the requests for them are happened. Later we are noticed that it is also troublesome to maintain them when a minor changes and/or modifications happen for some portion of the manufacturing steps once established.

Problem:

Some portion or parts of the objects may be changed or substituted with a bit superior one, in time. But there are too many objects and their production is complicated, it is troublesome to make a relevant change for them while not to conflict with each other.

Forces:

- Reference to an object contains important features of the object.
- Each objects do not differs much each other but they are not the same.
- The logic of the mechanism to maintain redundant information is very similar to that of inference rule. So if we incorporate it into the system, we have to maintain both of them not to conflict each other, when we need to change the inference rule.

Solution:

As in the case of *Reference has Clues*, conjecture and create the target object from the information given by the reference. In this case, be sure to create the object every time it is referenced (means create the object when its method is invoked), so that the updated (newest) production (factory) method for it can be applied and one can obtain the object whose portion or/and parts is kept in latest.

Resulting Context:

We remodeled our BOM to generate the manufacturing steps automatically every time it has referenced, from the information that the reference has. After that, we are enabled to change the manufacturing steps of all the items of a product-line by one action.

Some time, a production manager may optimize or detail the thus generated manufacturing steps of the item to meat the special request the customer made. When on the case, the scope of the change or optimization he/she made should be localized upon the item in interest, and also may not take effect on the other items in the same product-line. And though he/she changes the manufacturing steps of the items of the product-line later, the change may not take effect on the item once previously optimized or changed. In this case, see *Caught Red-handed*.

When *Reference Constructor* pattern is used in combination with DBMS that provides the proper isolation level, we still can use Flyweight and store an object on memory to consistently identify the object that is conjectured and created. In this case, be sure not to store the conjectured object into DBMS.

To clarify the difference between *Reference has Clues* and *Reference Constructor*, the former is intended to express the way to designate and produce the certain object that has large varieties, and pays no attention to whatever the changes are made to the inference rule, the production steps, or the potential parts, that will be used to constitutes the same type of object after ward. So it can be said that the resulting object is somewhat *frozen* in a sense of its structure once it was instantiated. In contrast, the later is focused on to produce the object that always reflects the latest structural information, since the latest inferred rule, the latest production steps of it, and the newly parts will be applied to construct it everywhere the object becomes in focus.

Chapter3 Possible Worlds

In the former chapters, we have discussed about our way to handle our ignorance in the world that we live in. Now, this chapter is about our way to handle the world we will meet *tomorrow*, possible worlds.

8. Rigid Designator

Context:

To the question that asking about the manufacturing history of a product-line that is newly designed these days, our BOM answered that it has been manufactured for ten years. This is because its name is the same as that of a product-line, which has already been out of date.

Problem:

Two distinct objects are looked up accidentally as the same object via a *Montage* reference.

Forces:

- We can recognize each objects if they are not so many.
- Though the features of two objects are the same, they are recognized as the distinguished objects.
- Identity is needed for distinction of the recognized object.

Solution:

Use a rigid designator that has a relevant scope and is agreed in ones community (e.g. OID, UID) to designate an object. Sometimes user must determine which object is the one he/she really wants looking at its details (e. g. the product-lines in the context can be determined by its manufacturing history).

Resulting Context:

We made it a rule to designate product-lines using OID. After that, we are enabled to discriminate the two product-lines that have same name.

But, a rigid designator has its valid scope. For example, some community may recognize a crescent and a full moon as two completely separated things and some may recognize them as the identical thing with different light angles. In such a situation, see *Federation Pattern* [3].

Chapter4 Summary

This pattern language is described in a point of view that the fact has its value and cost. If the cost is low enough, we can gather such facts and of course we will. We, as programmer, measure the lines of code everyday the work is done and whenever needed. And there is no problem to do so. But there are facts whose cost is high. If the value is low, we can simply ignore it and still ignorance of such kind of fact makes no harm. When its value is high, ignorance of such kind of fact would be troublesome. It is natural for us when we encounter such a situation, we simply use conjecture instead of obtaining the actual fact, no matter how risky it is.

For example, in the early stage of a software development, predicting the workload is needed and essential for managing the progress of development, we use estimation techniques to produce useful plan or schedule. Still they are useful but merely conjecture at the best and there is no warranty about the accuracy of the result. This kind of risk management is mainly discussed in the area of expectation management. In a more critical situation such that there is no way to predict the road to be chosen, we use hypothetical thinking and accept the burden of trial and error that is caused as the consequence of the application of the hypothesis. I always get into that situation when I faced with the problem that is totally strange to me, such as building a brand new style of architecture or cooperate with new comers. The patterns mentioned in this paper were induced from such kind of my own experience, and such a situation is commonly seen where the patterns are useful and exhibits its potential power. Those patterns are described as to specify and instruct the behavior of software system, but the behavior mentioned in those patterns reflects the essential behavior of the dynamic human hypothetical thinking, though the range it can treat is far more limited. And human learns through this thinking process. The process gathers the facts and constructs the reign of the fact, the reliable knowledge base, inside the system.

I want to round off this pattern language, in the word next to Peirce.

" The truth is that the whole fabric of our knowledge is one matted felt of pure hypothesis that is confirmed and refined by induction. Not the smallest advance in knowledge can be made beyond the stage of vacant staring, without making an abduction at every step.”[2]

Bibliography

[1] Gamma, E. et al, Design Patterns. 1995. Reading, MA: Addison Wesley.

[2] Kripke, S. Naming and Necessity. 1982. Cambridge, MA: Harvard University Press.

[3] Fujino, T. Federation Pattern. http://www.kame-net.com/jplop/

[4] Peirce, C. S. Collected papers of Charles Sanders Peirce (C. Hartshorne & P. Weiss, Eds.) (4th ed., Vols. 1 & 2). 1960. Cambridge, MA: Harvard University Press.