*Pink is my favorite crayon.*
Aerosmith

# 1 Archetypes, Color, and the Domain-Neutral Component

Color profoundly affects how we see the world around us.

Just as the transition from black-and-white photography to color is so profound, the transition from black-and-white modeling is an awesome one.

Welcome to the world of modeling with archetypes, color, a domain-neutral component, and a dozen domain-specific compound components.

In this chapter, you'll learn and apply archetypes, color, and the domain-neutral component. In Chapters 2-5, you'll read and apply domain-specific components. In Chapter 6, you'll discover feature-driven development, the process for all of this into best practice.

This book focuses on modeling with color, archetypes, and components—along with a process for putting it into practice.

How does Java fit in? The model shapes are Java inspired. You'll find composition rather than inheritance. You'll also see a judicious use of interface plug-in points—for added flexibility. In addition, the CD includes all of the models including Java skeleton source code.

We've developed this book as a front-end companion to Java Design. That book delivers specific strategies for designing with composition, designing with threads, and designing with notification.

Examples throughout this book use Unified Modeling Language (UML) notation. Class-diagram notation and some conventions used in this book are shown in Figure 1-1. Sequence-diagram notation and some conventions used in this book are shown in Figure 1-2. We suggest you scan those figures now, then refer back to them from time to time along the way.

## 1.1 Archetypes

Now let's turn our attention to this chapter's first major topic: archetypes.

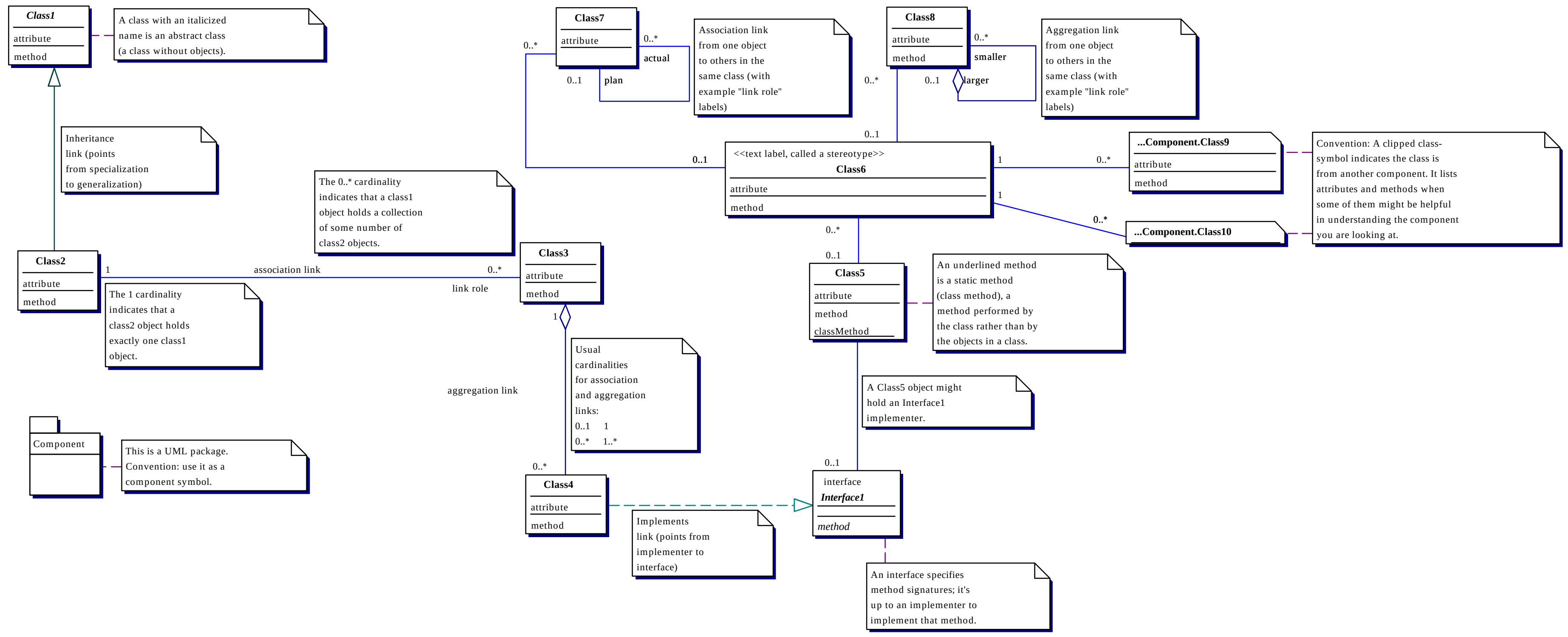

Class1
attribute
method
A class with an italicized name is an abstract class (a class without objects).
Inheritance link (points from specialization to generalization)
Class2
attribute
method
1
association link
0..*
link role
The 1 cardinality indicates that a class2 object holds exactly one class1 object.
The 0..* cardinality indicates that a class1 object holds a collection of some number of class2 objects.
Class3
attribute
method
1
aggregation link
Usual cardinalities for association and aggregation links:
0..1 1
0..* 1..*
0..*
Class4
attribute
method
Implements link (points from implementer to interface)
Component
This is a UML package. Convention: use it as a component symbol.
Class7
attribute
0..*
0..*
actual
0..1
plan
Association link from one object to others in the same class (with example "link role" labels)
Class8
attribute
method
0..*
smaller
0..*
0..1
larger
Aggregation link from one object to others in the same class (with example "link role" labels)
0..1
0..1
<<text label, called a stereotype>>
Class6
attribute
method
1
0..*
...Component.Class9
attribute
method
1
0..*
...Component.Class10
Convention: A clipped class-symbol indicates the class is from another component. It lists attributes and methods when some of them might be helpful in understanding the component you are looking at.
0..*
0..1
Class5
attribute
method
classMethod
An underlined method is a static method (class method), a method performed by the class rather than by the objects in a class.
A Class5 object might hold an Interface1 implementer.
0..1
interface
Interface1
method
An interface specifies method signatures; it's up to an implementer to implement that method.

*Figure 1-1. Class-diagram notation and conventions.*

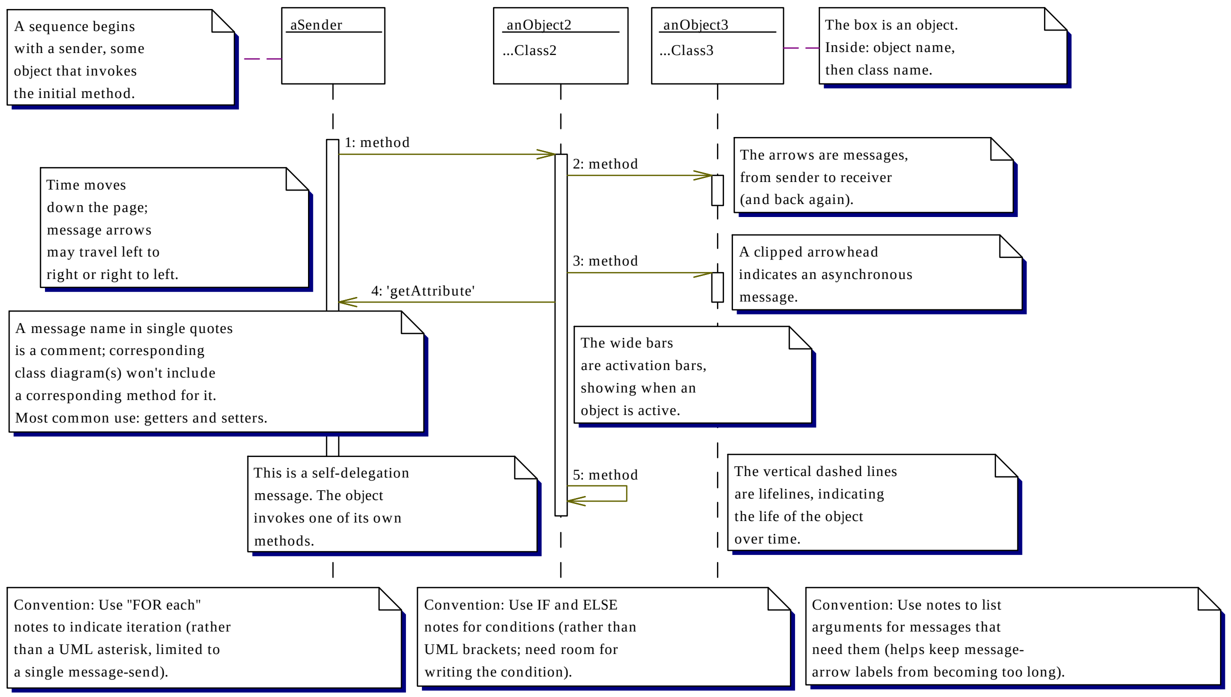

A sequence begins with a sender, some object that invokes the initial method.
aSender
anObject2
...Class2
anObject3
...Class3
The box is an object. Inside: object name, then class name.
1: method
2: method
The arrows are messages, from sender to receiver (and back again).
Time moves down the page; message arrows may travel left to right or right to left.
3: method
A clipped arrowhead indicates an asynchronous message.
4: 'getAttribute'
A message name in single quotes is a comment; corresponding class diagram(s) won't include a corresponding method for it. Most common use: getters and setters.
The wide bars are activation bars, showing when an object is active.
This is a self-delegation message. The object invokes one of its own methods.
5: method
The vertical dashed lines are lifelines, indicating the life of the object over time.
Convention: Use "FOR each" notes to indicate iteration (rather than a UML asterisk, limited to a single message-send).
Convention: Use IF and ELSE notes for conditions (rather than UML brackets; need room for writing the condition).
Convention: Use notes to list arguments for messages that need them (helps keep message-arrow labels from becoming too long).

*Figure 1-2. Sequence-diagram notation and conventions.*

Here's the concept we want to communicate:

*A form or template for one of a small number of class categories. It specifies attributes, links, methods, plug-in points, and interactions that are typical for classes in that category.*

Which is the better term for this concept?

*Stereotype*

1. *An* unvarying *model, as though cast from a mold*
2. *A text tag for annotating a UML diagram element*
3. *A broad categorization of classes*

*Archetype*

*A model from which* all things of the same kind more or less follow."

[Derived from Webster75 and Haykawa68]

"Archetype" says it best.

Yet which archetypes prove most useful in building better models?

We've developed hundreds of models in dozens of business and engineering domains. Along the way, we continually looked for ways to "abstract up" to a domain-neutral component – a small model of archetypes that we could apply again and again in our workshops and mentoring assignments. Why? We felt we could teach more in less time and accomplish more in less time. Better for our clients, more interesting for us, win-win.

Over time, we've discovered four interconnected archetypes that form a domain-neutral component:

- The moment-interval archetype
- The role archetype

- The "catalog-entry-like description" archetype
- The "party, place or thing" archetype.

We'd like to acknowledge that Peter Coad and Mark Mayfield laid the early groundwork for these four archetypes, first described in [Coad92] and later extended with David North in [Coad95-97].

### *1.1.1 The moment-interval archetype*

The first archetype in importance is a moment in or interval of time. It represents something that one needs to work with and track for business or legal reasons that occurs at a moment in time, or over an interval of time. For short, we call it a "moment-interval" to help remind us that we are looking for either a moment *or* an interval of importance in the problem domain.

A sale is made at a moment in time, the date and time of that sale.

A rental happens over an interval of time, from checkout to check-in. A reservation occurs over an interval of time, from the time that it is made until the time it is used, canceled, or expires.

A sale could even be an interval of time, if you track the duration of the sale for performance assessments.

What's important is that it is one of these two, not which one of the two it is. So we establish it as one archetype, moment-interval.

#### 1.1.1.1 Using archetypes to identify classes and much more

In any domain, one can look for moment-intervals and begin building a model. In material-resource management, we can move from request to RFQ to PO to delivery to invoice. In manufacturing management, we can move from a planned process and its steps to an actual process and its steps.

So one of the ways that archetypes help in guiding model building is in identifying classes that need to be included in the model.

Yet archetypes are more than simply a categorization of classes. They are also a categorization of the responsibilities (the attributes, links, methods, plug-in points, and interactions) that such classes usually have.

#### 1.1.1.2 Labeling an archetype

What we need is a text tag, so we can indicate which archetype we are applying when establishing a class. In UML, that text tag is called a stereotype, an extension mechanism within that notation (Figure 1-3).

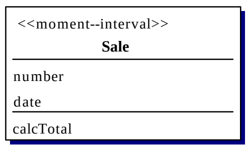


*Figure 1-3. Using a UML text tag to indicate a moment-interval.*

The problem is that text tags like <<moment-interval>> hide some very important meaning in a rather plain and simple text label. In a family of diagrams, that little label is lost in the noise as it begins to look like all the other labels. And that is too bad; expressing the archetype is far more important than it would be getting credit for. It would be nice if one could give this added layer of information added punch, so it could:

- Grab your attention to work on that part of the diagram first
- Help you discover a progression of moment-intervals over time
- Guide you in linking other classes into the moment-interval you are working with
- Quietly move you to considering what is linked to that moment-interval and how it works with others to get things done.

Expressing archetypes with color is the extra dimension, the added punch that does all that and more.

#### 1.1.1.3 Implementing archetypes

What does an archetype look like in source code?

An archetype describes a model that classes within that archetype more or less follow. It's the "more or less" aspect that is important here.

Could we implement an archetype as a superclass and then inherit from it? No! Here's why. The very nature of an archetype is that each and every class that follows an archetype only more or less follows it. Inheritance is far too rigid for what archetypes are all about, what they need to express.

The other way to implement an archetype is by using a keyword-coded comment, one that modeling tools can recognize and use effectively. In Java, we do this with a javadoc-style comment, using a coded keyword. For example, here's such a comment with an embedded keyword (following the @ symbol)"

```
/** @archetype moment-interval*/
public class Sale {
  public BigDecimal calcTotal(){
  }
  private int number;
  private Date date;
}
```

That gets the job done nicely.

### *1.1.2 The role archetype*

The second archetype in importance is a role. A role is a way of participation by a person, place, or thing.

Another way to say this is that a role is a way of participation by a party (person or organization), place or thing. We like this better, since many times a person or organization are eligible to play the same role (for example, owner) within a problem domain that we are working in.

So we model the role-player (a party, place, or thing) as well as the role (the "hat" that the party, place, or thing is wearing). The role player captures core attributes and behaviors that apply no matter what combination of hats it might be wearing. For person, that often includes attributes like legal name and date of birth. It also includes methods that enforce business rules across the collection of roles being played, for example, a method "authorized for" that interacts with each role and applies rules across that collection of roles to determine if it is authorized to take a given action (Figure 1-4).

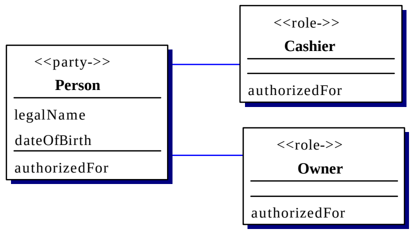


*Figure 1-4. A party and its roles.*

Party, person, and organization roles are the norm. Occasionally you'll find place and thing roles too (for example, a product and its two roles, "product in a sales process" and "product in use").

### 1.1.3 The description archetype

The third archetype is a description. More specifically, it's a catalog-entry-like description. It is a collection of values that apply again and again. It also provides behavior across the collection of all things that correspond to its description.

For example, your red pickup is a vehicle; it's a thing with its own serial number (called a vehicle identification number), purchase date, color, and odometer reading. The corresponding catalog-entry-like description is vehicle description; it establishes manufacturer, model number, date of manufacture, and available colors; it also is a good place to locate business-related methods like, "how many of these trucks are in good working order?"

### 1.1.4 The "party, place, or thing" archetype

A party (meaning, a person or an organization), place or thing is someone or something who plays different roles. A person might be both an employee and a customer. A place might be both a retail outlet and a wholesale outlet. A thing might play a role in a manufacturing process and a different role in a purchasing process.

## 1.2 Color

Initially, we used textual archetype labels. Yet we found it increasingly difficult to look at and really see the overall shape of the model including the extra dimension that those archetypes expressed.

And that's where color came into play.

In September 1997, we started building models with four colors of Post-it™ Notes: pink-yellow-green-blue. Some of those new to model building on the team commented a number of times long the way, "But how could you have possibly built effective models in the past without color?" We developed this technique in practice, published initial findings [Coad97a], and presented this approach in an OOPSLA '97 tutorial [Coad97b].

As is often the case, practice preceded theory. Seeing these ideas work so well in practice, we began investigating color and why it appears to have such a profound effect on building better models.

### *1.1.1 Why color?*

Why use color in component models? Color gives us a way to encode additional layers of information. The wise use of color increases the *amount of content* we can express.

More importantly, one can use color to add *layers of new content* to models. Those layers are visible from a distance, so that "big picture" model content comes across even before one starts reading the details. We call this effect "spatial layering"; it means that a model is capable of delivering an overview and a detailed view all within itself, without needing to break visual context by jumping to some other representation. Color makes spatial layering possible.

> *"Among the most powerful devices for reducing noise and enriching the content of displays is the technique of layering and separation, visually stratifying various aspects of the data." [He then describes how to do this: use distinctions in shape, lightness, size, and especially color.]*
>
> Edward R. Tufte [Tufte90]

Hence, we can use color enrich the content of models. In fact, we can apply color to achieve four objectives:

> *"The fundamental uses of color in information design: to label (color as a noun), to measure (color as a quantity), to represent or imitate reality (color as a representation), and to enliven or decorate (color as beauty)."*
>
> Edward R. Tufte [Tufte90]

What this means to modeling is that we can use color to:

- Label added layers of information (for example, layers of classes with similar characteristics).
- Indicate the progression of time (for example, one might use different shades of lightness to show such a progression)
- Represent key categories of information within a model
- Add visual impact to the model.

Added visual impact is important. Modeling is by its very nature a visually oriented activity. Those with strong spatial intelligence are especially drawn to model building and model reading.

> *"Spatial knowledge can serve a variety of scientific ends, a useful tool, an aid to thinking, a way of capturing information, a way of formulating problems, or the very means of solving the problem."*
>
> Howard Gardner [Gardner83]

> *"This ability to idealize results, to see through the mess of real-life observations to what ought to be there, is one of the marks of genius."*
>
> Robert Scott Root-Bernstein [Root-Berstein85]

Adding color better engages the spatial intelligence of both model-builders and model-readers alike.

### *1.1.2 How many colors?*

We started with four colors. Yet how many colors should we be using?

In visual design, it's a good idea to limit the number of colors in a color scheme. Why? Simply put: it's a good way to increase the likelihood of color harmony within that color scheme.

> *"Two or three colors is usually enough; five is too many. Four-color combinations must be selected with great care: nothing looks worse than too many colors, particularly when they lack common elements."*
>
> Hideaki Chijiiwa [Chijiiwa87]

To support visual modeling in color, the last thing we want to do is end up with something that is visually distracting. We want to support better design, not distract from it. Hence, no matter how many semantic variations we might come up with, using four colors seems like a good place to start.

### *1.1.3 Which colors?*

The three-primary system, first proposed around 1731, defines primary colors as red, blue, and yellow. It defines secondary colors as orange, green, and violet.

The perceptual-primary system, first proposed by Leonardo da Vinci, defines primary colors as red, yellow, blue, and green. These are the perceptual primaries, those colors that do not appear to have any other color in them.

The six-primary system, first proposed in the 1990s, gives equal importance to red, yellow, green, blue, orange, and violet. The basis for this system is that blue and yellow don't make green; instead, bits of green impurities within so-called blue paint and so-called yellow paint makes green. Hence green (and for that matter orange and violet) deserve to be considered primary colors too. [Wilcox94]

We can mute these colors by adding a little white to them. That makes text placed on those colors much easier to read. So, for the four archetypes, we can use pink, pastel yellow, pastel blue, and pastel green. Let's see how!

## 1.3 The Four Archetypes in Color

Our models always consist of four archetypes: role, moment-interval, thing, and description.

Let's match up archetypes with colors, to deliver that added impact we're looking for.

Moment-intervals tie together a component model. Moment-intervals express the heart and soul of what that component is all about. In a model, moment-intervals often encapsulate the most interesting methods. Let's make moment-intervals *pink*, the most attention-grabbing of the colors.

Roles played by a party, place, or thing are often the next most important part of a model. Roles include methods like assess performance or assess value. Let's use the next most attention grabbing color, *yellow*.

Things are often the next in line. Things often act as containers for other objects and usually include methods like assess performance or assess value. Descriptions are last. Descriptions often include methods like how many are available and calculate total for quantity. Things might have corresponding descriptions, too. Hence, *green* (for thing) followed by *blue* (for description) best fits the color scheme. So we finally end up with these assignments (Figure 1-5):

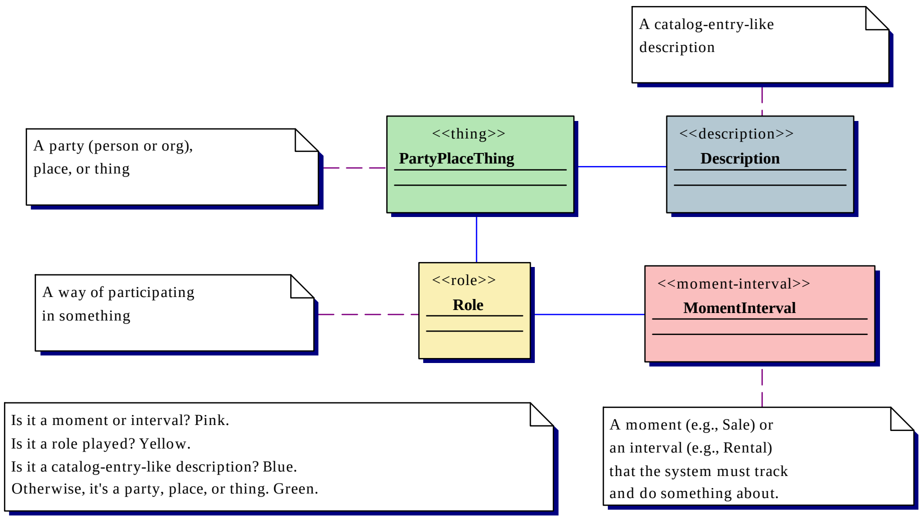

A catalog-entry-like description
A party (person or org), place, or thing
<<thing>>
PartyPlaceThing
<<description>>
Description
A way of participating in something
<<role>>
Role
<<moment-interval>>
MomentInterval
Is it a moment or interval? Pink.
Is it a role played? Yellow.
Is it a catalog-entry-like description? Blue.
Otherwise, it's a party, place, or thing. Green.
A moment (e.g., Sale) or an interval (e.g., Rental) that the system must track and do something about.

*Figure 1-5. The four archetypes and their colors.*

Each of these four colors corresponds to an archetype's characteristics, the attributes, links, methods, plug-in points, and interactions that corresponding classes follow, more or less.

An archetype's characteristics include attributes and links (Figure 1-6). A blue description knows its type, description, item number, and default value(s). A green party, place or thing knows its serial number, address, and custom value(s). A yellow role knows its assigned number and status. A pink moment-interval knows its number, date (or date-time or interval), its priority, its total, and its status.

A blue links to a green links to a yellow links to a pink. Sometimes we don't need a green and yellow in the mix, in which case a blue links to a pink.

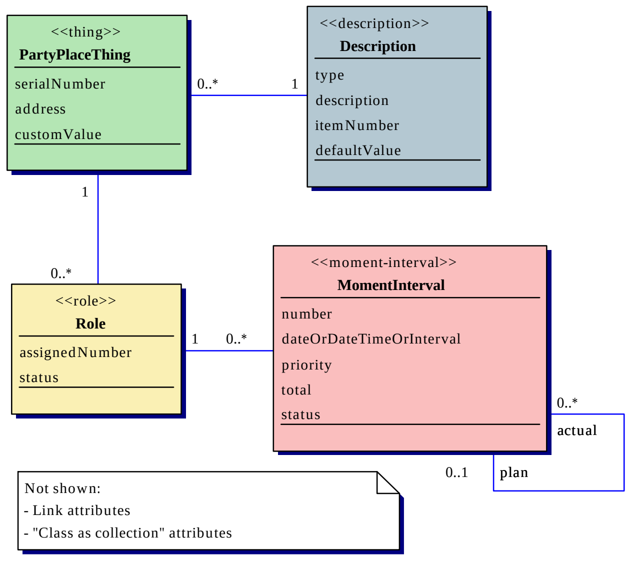


*Figure 1-6. Archetypes—and their attributes and links.*

An archetype's characteristics include methods (Figure 1-7).

A blue description finds an available one and calculates the quantity available (in both cases, interacting with its corresponding green party, place or thing objects to do so).

A green party, place, or thing determines if it's currently available (checking status or interacting with its yellow roles). It gets its custom value or if not present asks its corresponding blue description for its default value. It also assesses its value to the business and assesses its performance (in both cases interacting with its yellow roles).

A yellow role determines if it's available to play its role (might be busy), assesses its value to the business and assesses its performance, in all three cases by interacting with its pink moment-intervals.

A pink moment-interval makes one (supports the business process for making one, that is), adds details (parts), and calculates its total (interacting with its parts to do so). It recalculates its total (forcing a recalculation, regardless of any internally buffered value). It accepts messages asking it to complete or cancel the moment-interval. It also provides behavior across other pink moment-intervals (designated by the prefix "mi_"): generate next, assess with respect to prior moment-intervals, assess with respect to subsequent moment-intervals, and compare plan vs. actual. It also has two (underlined) methods with behavior across all of the objects in the class: list all of the moment-interval objects and calculate the average moment-interval (usually average amount, although it could be something like average weight or average fulfillment time).

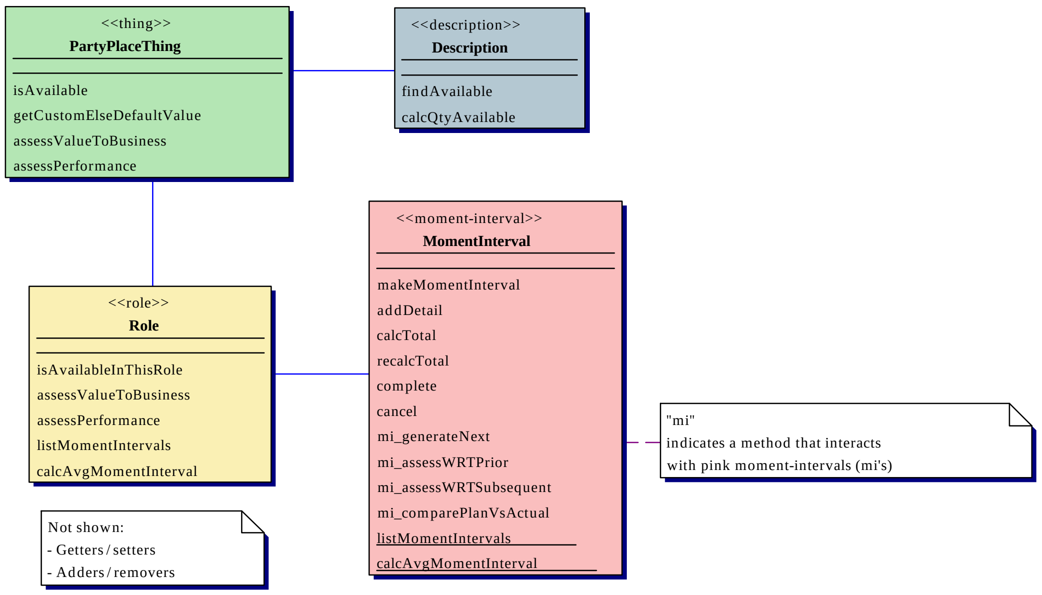


*Figure 1-7. Archetypes and their methods.*

Note that we follow this basic sequence in naming methods:

- Make (to make the object, including conducting the business process to get there)
- Object-specific calculations and assessments
- Moment-interval (MI) methods, ones that interact with other pink objects (we add the "mi" suffix to separate these methods from others in the list)
- Underlined methods, indicating static (or class) methods, ones that act across the collection of all of the objects in that class

An archetype's characteristics include plug-in points for adapting the behavior of an archetype (Figure 1-8). A blue description needs a plug-in point when it has algorithmically complex behavior and we want the option of plugging in an alternative behavior at times. A pink moment-interval needs a plug-in point whenever the business process is complicated enough that we really should design-in plug-in flexibility to accommodate (anticipated or unanticipated) business process change over time.

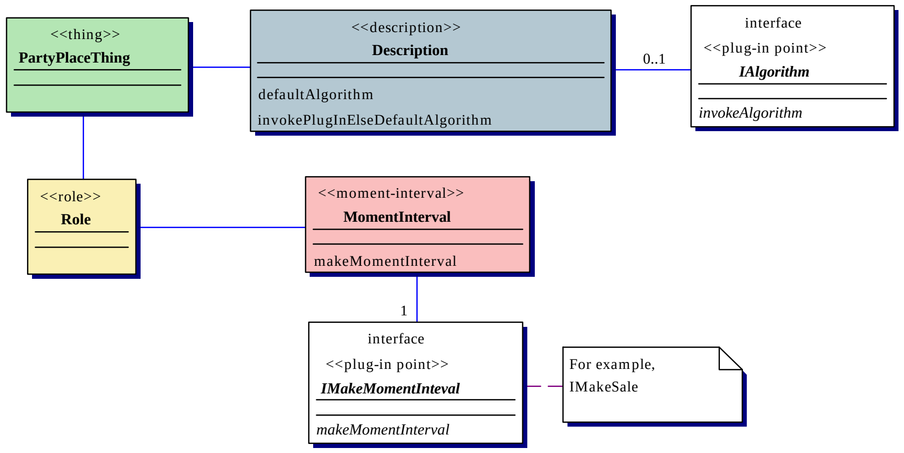


*Figure 1-8. Archetypes and their plug-in points.*

Figure 1-9 summarizes the attributes, links, methods, and plug-in points of archetypes. We add interactions later in this chapter.

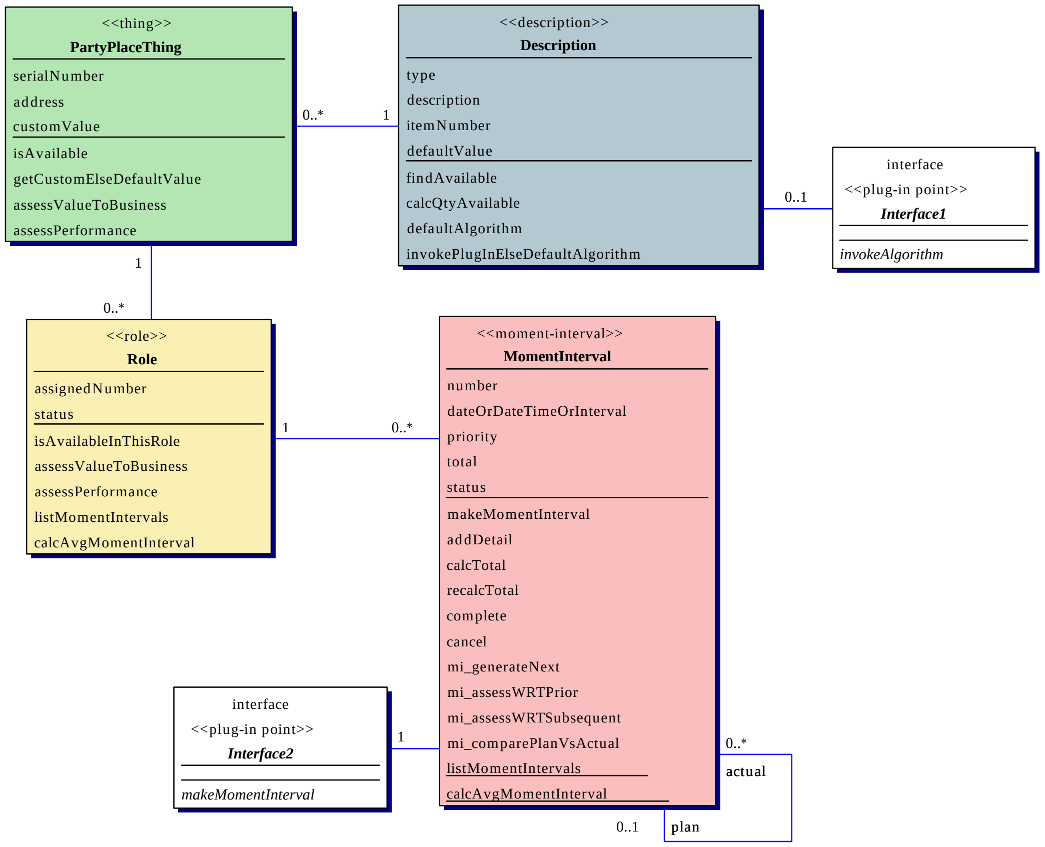


*Figure 1-9. Archetypes—and their attributes, links, methods, and plug-in points.*

We usually include plug-in points to complex moment intervals (meaning, one with parts) and to calculation-intensive blue descriptions.

Another point is worth mentioning here. Often, a pink moment-interval has parts, called moment-interval details. Think of them as being a little piece of a moment interval, something it needs to do its job (Figure 1-10).

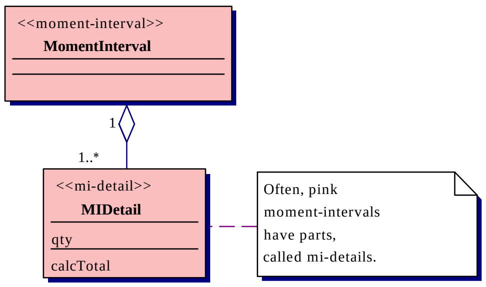


*Figure 1-10. A pink moment-interval and its details.*

A pink moment-interval detail knows its quantity and calculates its total.

## 1.4 Given a Class, What's the Color, What’s the Archetype?

So given a class name, what archetype or color should you use? Use this checklist:

1st - Is it a moment in or interval of time, something the system needs to track for business or for legal reasons? If so, it's a pink moment-interval.

2nd – Otherwise, is it a role? If so, it's a yellow role.

3rd – Otherwise, is it a catalog-like description, a grouping of values that you can apply again and again? If so, it's blue description.

4th - Otherwise, it's a party, place or thing. It's a green party, place, or thing (green is the default; if not pink yellow, or blue, it's green).

We also use white occasionally, for notes, for plug-in points, and for system-interaction proxies.

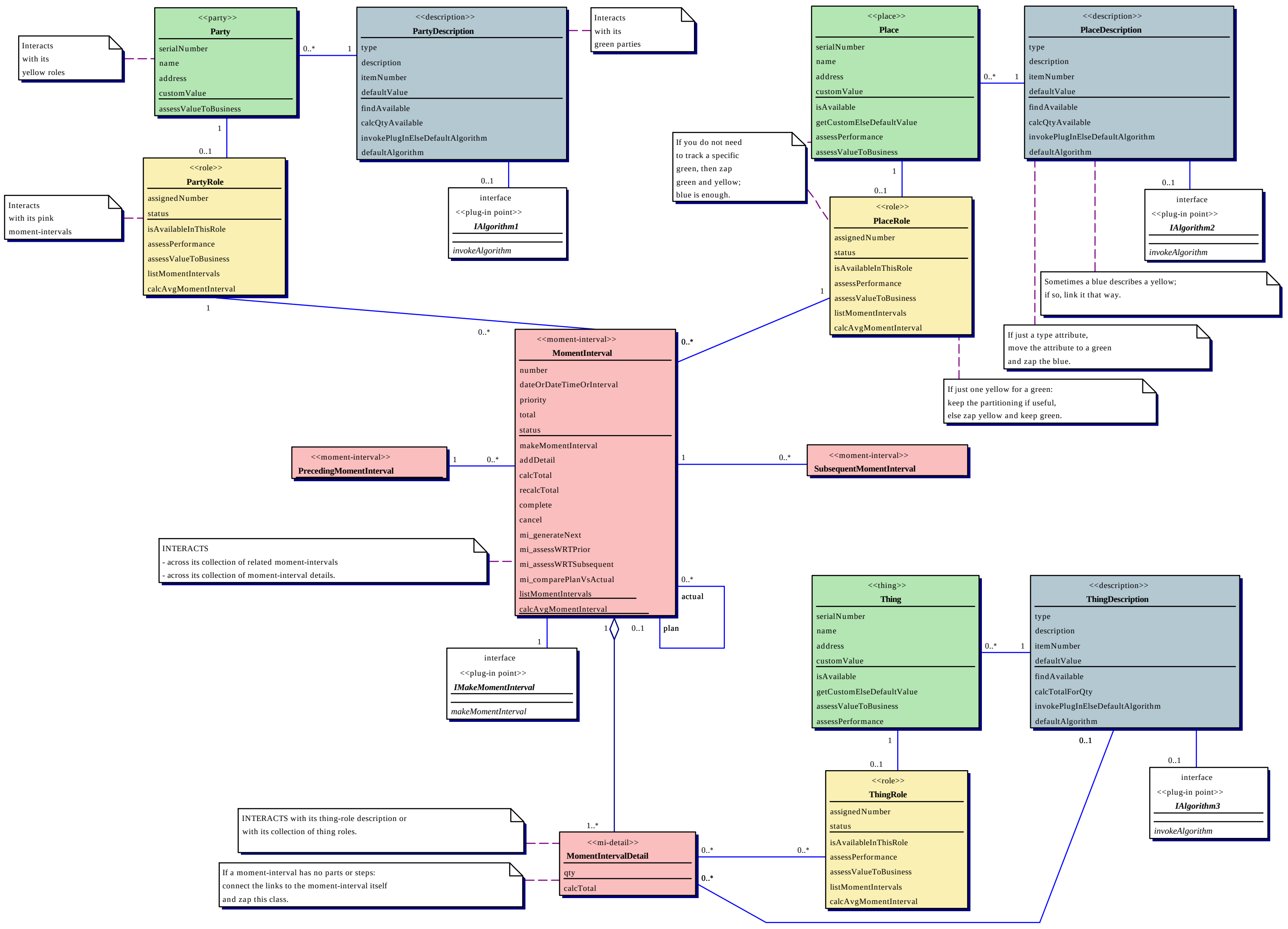
## 1.5 The Domain-Neutral Component

Archetypes in color are very useful little building blocks.

Let's take them a step further.

These four archetypes in color plug into each other in a very repeatable and predictable way. We call it a "domain-neutral component".

We've built hundreds and hundreds of models. All of them follow this domain-neutral component model. In Chapters 2-5, you'll find 51 domain-specific components. All of them follow the domain-neutral component shown in Figure 1-11.



Interacts with its yellow roles

Interacts with its green parties

Interacts with its pink moment-intervals

If you do not need to track a specific green, then zap green and yellow; blue is enough.

Sometimes a blue describes a yellow; if so, link it that way.

If just a type attribute, move the attribute to a green and zap the blue.

If just one yellow for a green: keep the partitioning if useful, else zap yellow and keep green.

INTERACTS
- across its collection of related moment-intervals
- across its collection of moment-interval details.

INTERACTS with its thing-role description or with its collection of thing roles.

If a moment-interval has no parts or steps: connect the links to the moment-interval itself and zap this class.

*Figure 1-11. The domain-neutral component.*

## 1.6 Interactions within the Domain-Neutral Component

This section focuses on archetype interactions within the domain-neutral component.

Some refer to class diagrams as static and sequence diagrams as dynamic. Actually, neither diagram moves! It takes some imagination, some additional internal visualization, to see the motion.

Class diagrams are implicitly dynamic. (It takes a bit of imagination to see the interactions although one can really learn to see them!) Sequence diagrams are explicitly dynamic (well, as good as it gets without having on-screen simulations in front of you).

Archetypes include typical interactions too. Once you know those interactions, even class diagrams spring to life.

Now rest assured, this is not an excuse for not including sequence diagrams in this book! Actually, you will find more sequence diagrams than class diagrams in here.

We feel compelled to make this added point though, that with archetype interactions you really can learn to look at a class diagram and visualize its most important interactions.

So let's look at how to do this. First, how to visualize an association in three-dimensions. Then, how to visualize message-sends within that new dimension. And finally, walk through a series of class diagram and sequence diagram pairs, so you can begin to visualize those interactions for yourself.

Here we go.

Consider an association with a 0..* marking on one end (Figure 1-12).

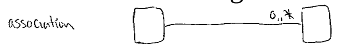


*Figure 1-12. An association link.*

Already, the class diagram is asking us to visualize beyond what it explicitly represents. An object on the left side links to some number of objects on the right side. Visually, you should translate it to something like this, in your mind (Figure 1-13):

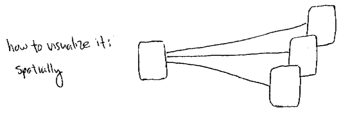


*Figure 1-13. How to visualize that link, spatially.*

Again, the class diagram is asking us to see beyond what is explicitly represented. The object on the left interacts with the objects on the right, by sending messages to each one. Here's how to visualize the implicit dynamics of an association link (Figure 1-14):

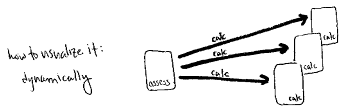


*Figure 1-14. How to visualize that link, dynamically.*

Thankfully, sequence diagrams are *explicitly* dynamic, giving added visual clues about the sequence of interactions inherent within a class diagram.

Here are the archetypal interactions for the domain-neutral component—and indeed the archetypal interactions for all components in this book (Figures 1-15 to 1-20).

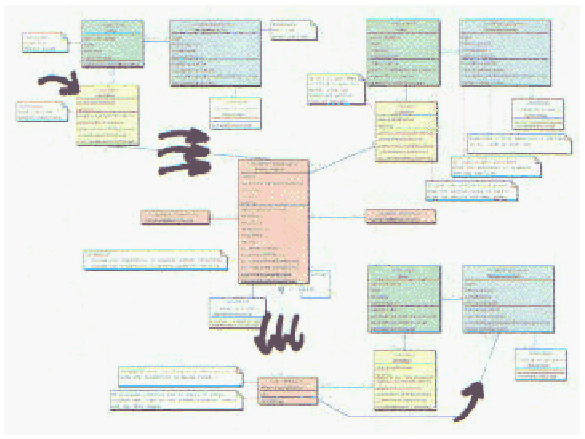


*Figure 1-15a. Assess value to business: implicit dynamics.*

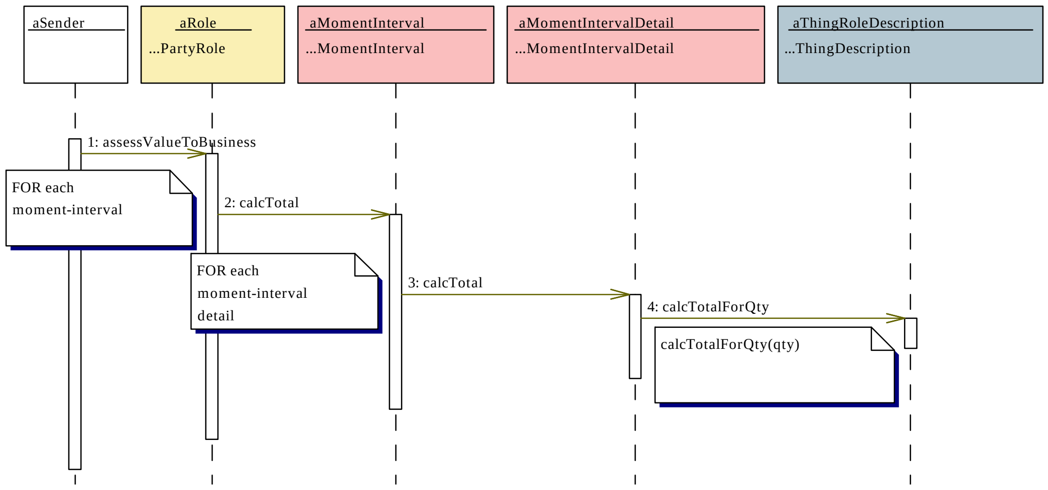


*Figure 1-15b. Assess value to business: explicit dynamics.*

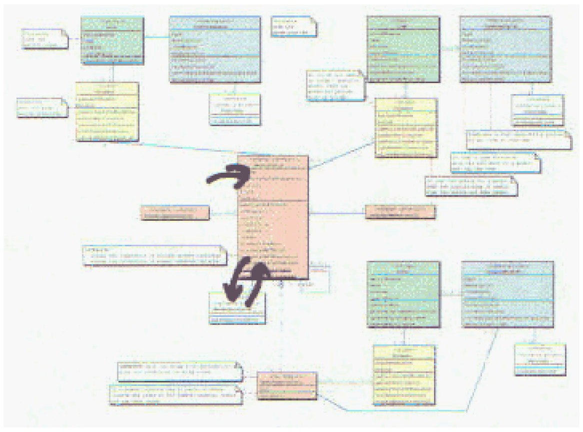


*Figure 1-16a. Make moment-interval: implicit dynamics.*

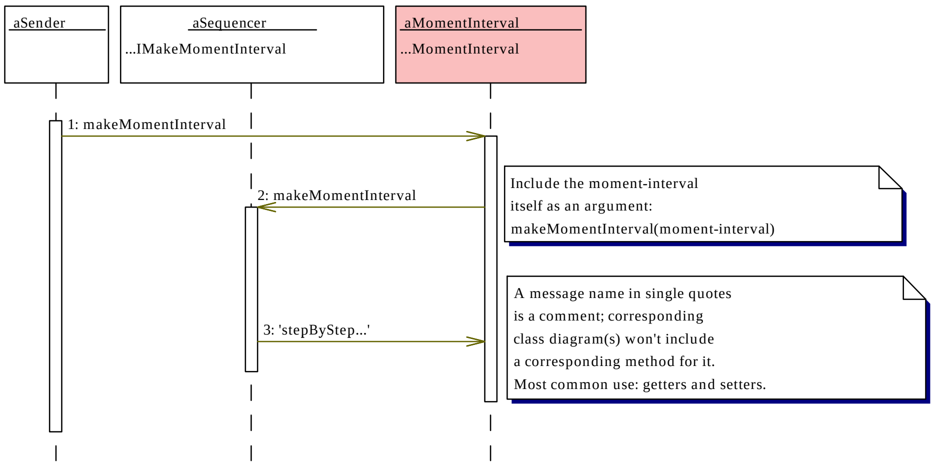

aSender
aSequencer
...IMakeMomentInterval
aMomentInterval
...MomentInterval
1: makeMomentInterval
2: makeMomentInterval
3: 'stepByStep...'
Include the moment-interval
itself as an argument:
makeMomentInterval(moment-interval)
A message name in single quotes
is a comment; corresponding
class diagram(s) won't include
a corresponding method for it.
Most common use: getters and setters.

*Figure 1-16b. Make moment-interval: explicit dynamics.*

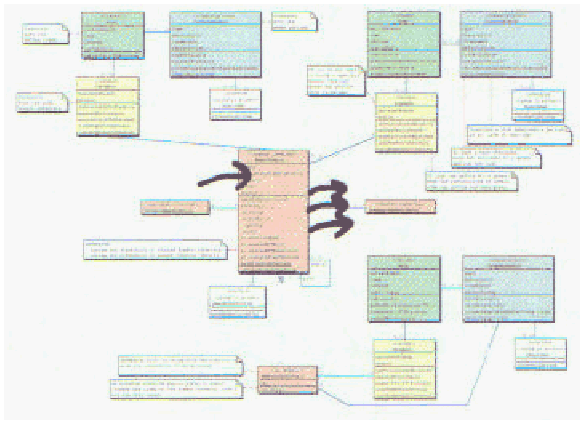


*Figure 1-17a. Assess with respect to a subsequent moment-interval: implicit dynamics.*

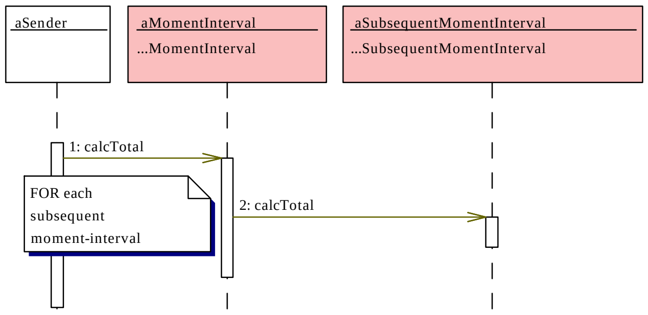

aSender
aMomentInterval
...MomentInterval
aSubsequentMomentInterval
...SubsequentMomentInterval
1: calcTotal
FOR each subsequent moment-interval
2: calcTotal

*Figure 1-17b. Assess with respect to a subsequent moment-interval: explicit dynamics.*

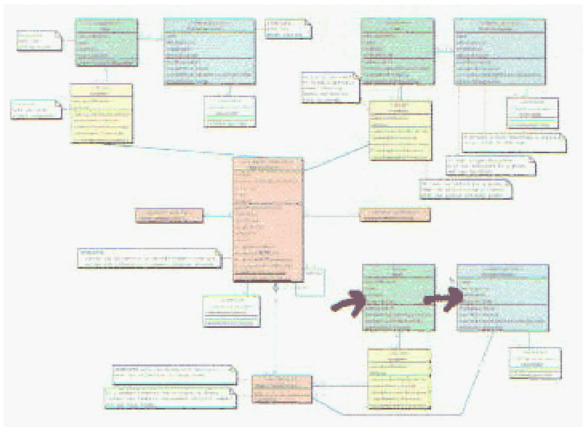
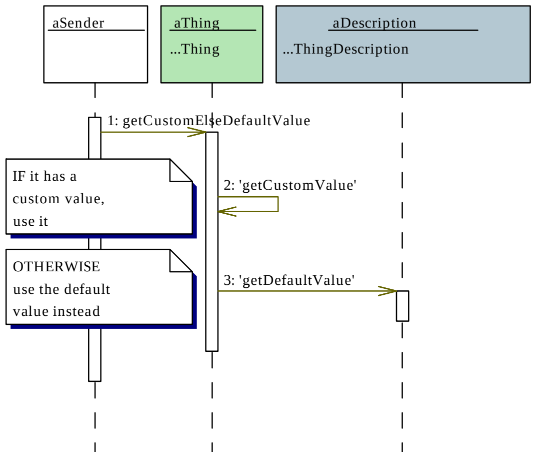


*Figure 1-18a. Get custom else default: implicit dynamics.*


aSender
aThing
...Thing
aDescription
...ThingDescription
1: getCustomElseDefaultValue
IF it has a custom value, use it
2: 'getCustomValue'
OTHERWISE use the default value instead
3: 'getDefaultValue'
*Figure 1-18b. Get custom else default: explicit dynamics.*

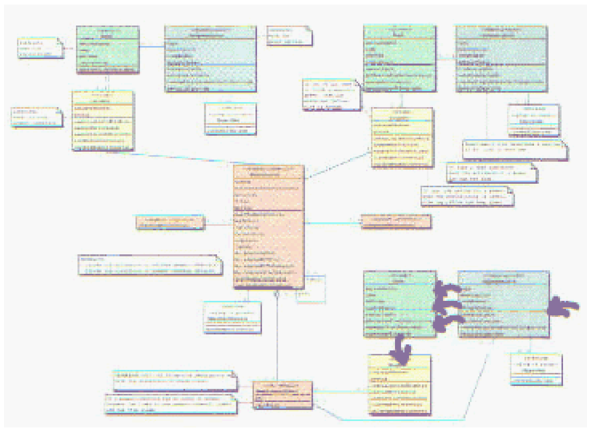


*Figure 1-19a. Find available: implicit dynamics.*

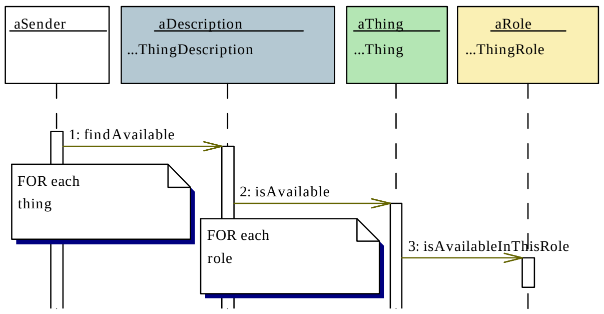


*Figure 1-19b. Find available: explicit dynamics.*

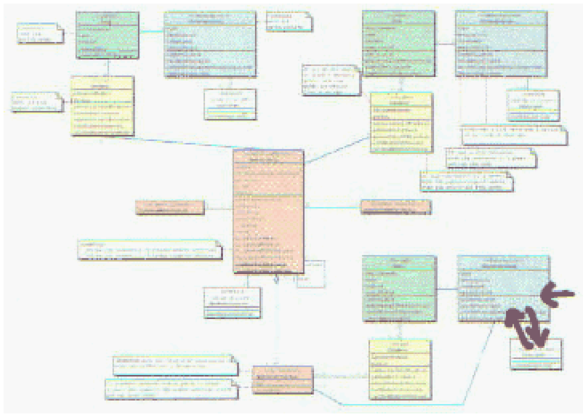


*Figure 1-20a. Invoke plug-in else default: implicit dynamics.*

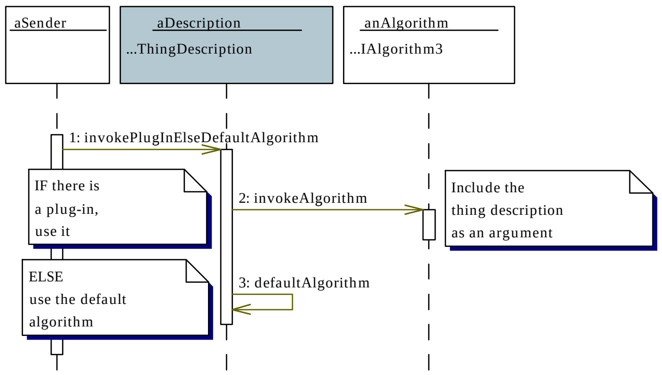

aSender
aDescription
...ThingDescription
anAlgorithm
...IAlgorithm3
1: invokePlugInElseDefaultAlgorithm
IF there is a plug-in, use it
2: invokeAlgorithm
Include the thing description as an argument
ELSE use the default algorithm
3: defaultAlgorithm

*Figure 1-20b. Invoke plug-in else default: explicit dynamics.*

## 1.7 Component Connectivity

On printed circuit boards, some components directly connect (that is to say, they are hardwired). Others components plug in. Why not make every element something you can plug-in? Sockets everywhere! Well, the reason why is that it's simply not cost-effective to do so.

The same is true with software components. Some are hardwired. Although we could have plug-in points everywhere, it's not cost-effective to do so. So we choose and design-in plug-in points at those places where we need and can afford to implement added flexibility.

The rest of this section moves into a level of detail that you might wish to skip for now and come back to at another time. Yet for those with inquiring minds that want to know, here are the details behind component connectivity.

A "direct connect" links an object in one component with objects in another component (Figure 1-21).

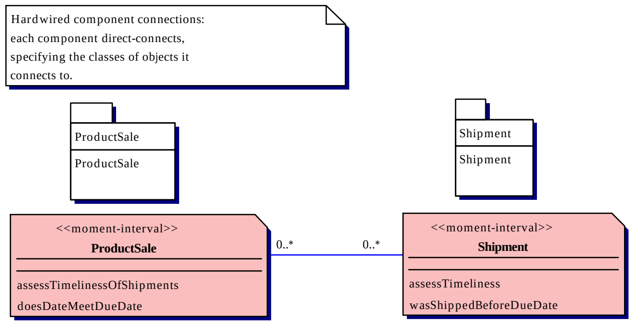


*Figure 1-21. Direct connectivity.*

A product-sale object holds a collection of some number of shipments. And a shipment holds a collection of some number of product sales.

We can ask a product sale to assess the timeliness of its shipments; it interacts directly with its shipment objects (Figure 1-22).

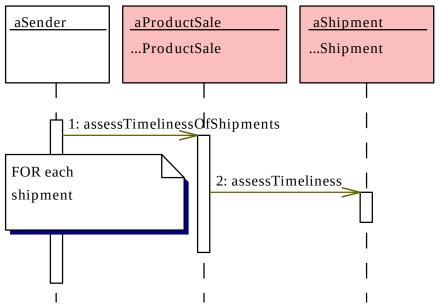


*Figure 1-22. Direct connectivity: a product sale interacts with its shipments.*

And we can ask a shipment if it was shipped before the due dates for its corresponding product sales; it interacts directly with its product-sale objects (Figure 1-23):

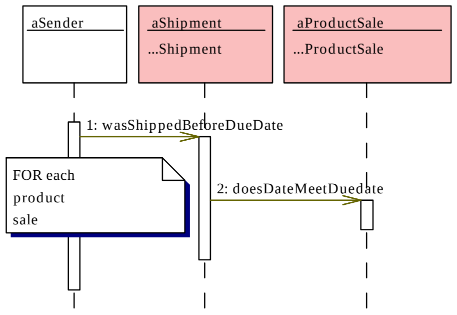


*Figure 1-23. Direct connectivity: a shipment interacts with its product sales.*

Now consider plug-in connectivity. An object interacts with whatever is plugged into a plug-in point (Figure 1-24).

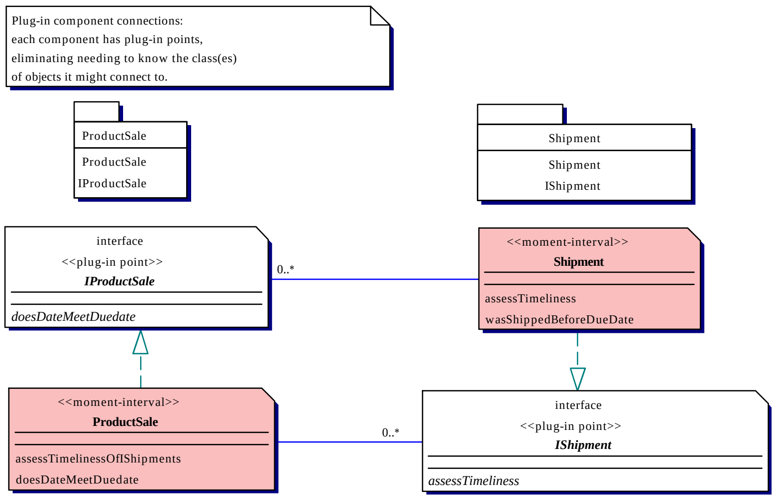


*Figure 1-24. Plug-in connectivity.*

A product sale holds a collection of objects from classes that implement the IShipment interface. A shipment holds a collection of objects from classes that implement the IProductSale interface. It really does not matter which classes the connecting objects are in, only that the interface must be implemented.

We can ask a product sale to assess the timeliness of its shipments; it interacts with its IShipment implementers, whatever is plugged into that plug-in point (Figure 1-25).

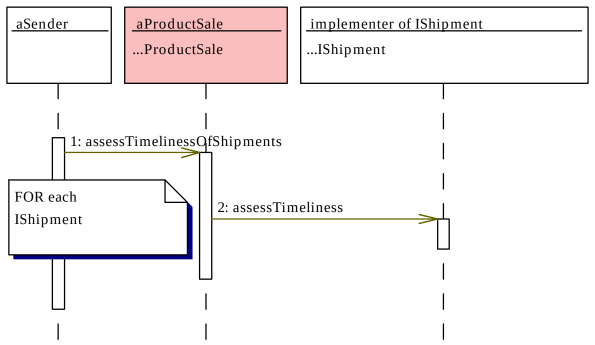


*Figure 1-25. Plug-in connectivity: a product sale and its shipments.*

And we can ask a shipment if it was shipped before the due dates for its corresponding product sales; it interacts with its IProductSale implementers, whatever is plugged into that plug-in point (Figure 1-26).

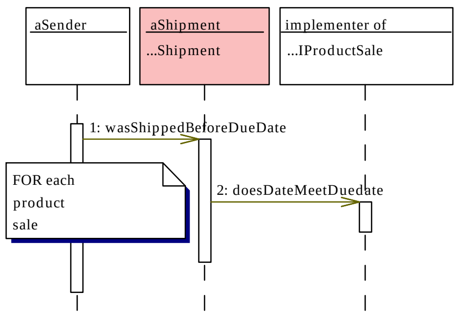


*Figure 1-26. Plug-in connectivity: a shipment and its product sales.*

For modeling simplicity, we build models with direct connections (like wire wrap, to extend the circuit-board analogy a bit further). Then we choose where we want flexibility and add plug-in connectivity (like deciding where we want sockets on a circuit board; it's a decision in adding flexibility).

## 1.8 Twelve Compound Components

What if you had a substantial collection of enterprise-component models? Each component would define a fundamental model shape for supporting some aspect of your business. Each component would establish the most common responsibilities for the classes and interfaces to support some aspect of your business. Each component would define the plug-in points for extending capabilities. In addition, each component would give your modeling team something more than a blank whiteboard blank screen for getting started in building a model for your application or family of applications.

That is what this book delivers.

These components are ready to use and reuse as you see fit. You can put them to work in a number of ways, for example:

- Use as-is.
- Extend a component by plugging-in new capabilities at the plug-in points.
- Extend by adding additional content
- Use as a cross-check, an outside opinion, one that you can compare and contrast with your own on-going work.

The twelve compound components are shown in Figure 1-27.

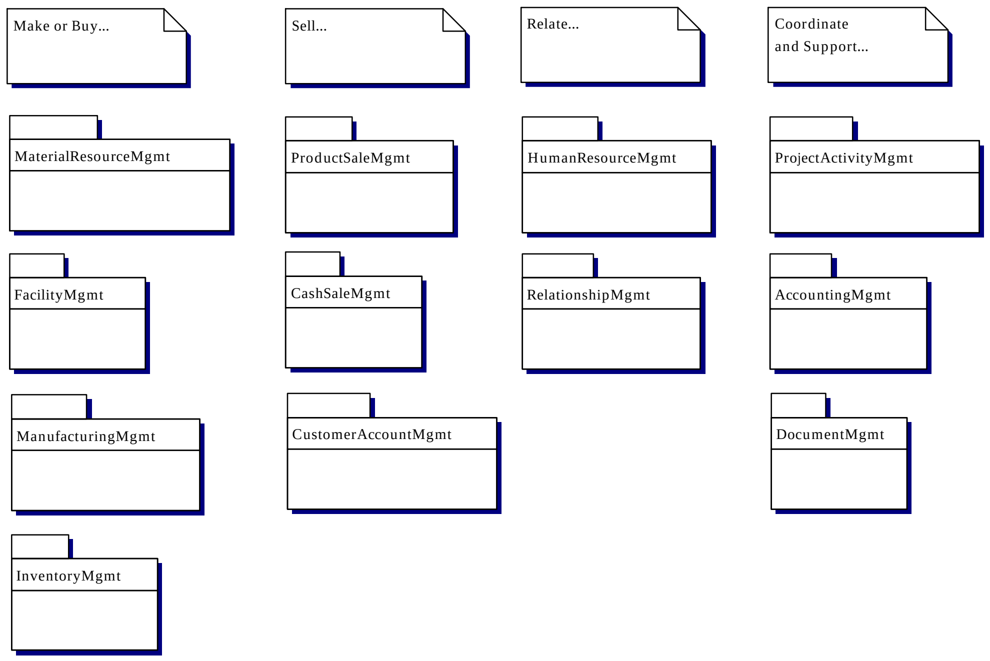


*Figure 1-27. The twelve compound components.*

## 1.9 Suggested Reading Paths

You might choose to read this book in any number of ways.

If you are interested in *a sampling of interesting compound components*, read the first major section of each chapter (that is to say, 2.1, 3.1, 4.1, and 5.1).

If you are interested in a *top-down understanding of the components*, begin with project-activity management and then to its subordinates: material-resource management, facility management, manufacturing management, inventory management, and human-resource management. Then scan the others.

If you are interested in *learning specific modeling tips*, begin by scanning the modeling-tips appendix. Then carefully study Chapter 2. Finally, scan Chapters 3-5 for additional tips. (We present each tip right after we've shown its application by example. Once we present a tip, we do not repeat it in subsequent material. So Chapter 2 has the most tips.)

If you are interested in *the domain-neutral component and its steady application across 51 components*, copy the domain-neutral component with four colors of Post-it Notes and then study the class-diagram shapes in Chapters 2-5, looking at what's the same and what's different, observing the little things along the way.

If you are interested *in templates, plans, and plan executions*, study manufacturing management and then activity management.

If you are interested in *material resources that eventually become products* and how the two interrelate and interact, read material-resource management, then product-sale management, and then inventory management.

If you are interested in *system and device interaction*, read manufacturing management (the section on device interaction) and accounting payment (the section on authorization-system interaction).

If you are interested in *process*, read chapter 1, scan chapters 1-5 and then study chapter 6.

## 1.10 Summary

This chapter introduced enterprise-component models in color.

We are developing enterprise-model components and a process for building, applying, and adapting those components. Along the way, we've discovered that encoding added layers of information (roles, moment-intervals, things, and descriptions) was an essential ingredient for both building and reading component models; we found that color was especially suitable for adding those layers of information.

Component modeling with color is so effective that we expect that we will never again return to the monotonous flatland of monochrome modeling.

So get a set of four-color Post-it® Notes and try this out for yourself. Take an existing model you are working on—or start with a new one, if you wish. Add "stickers" for the moment-intervals (pink), the parties and roles (yellow), the things (green), and the descriptions (blue). Then stand back and check it out. Discuss it with a colleague. Walk through it with a domain expert.

Or, if you already have a large model, get a set of color highlighting pens (pink, yellow, green, and blue) and highlight the class names in your model. This is another good way to get started.

### References

Color and Visualization

[Chijiiwa87] Chijiiwa, Hideaki, *Color Harmony*. Rockport Publishers, 1987.

[Gardner83] Gardner, Howard, *Frames of Mind: The Theory of Multiple Intelligences*. Basic Books, New York, 1983.

*The Elements of Color*. Van Nostrand Reinhold, 115 Fifth Avenue, New York, 1970.

Root-Bernstein, Robert Scott, "Visual Thinking: The Art of Imagining Reality." *Transactions of the American Philosophical Society*, Volume 75, 1985.

[Tufte90] Tufte, Edward R., *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990.

Walker, Morton, *The Power of Color*. Avery, 1991.

[Wilcox94] Wilcox, Michael, *Blue and Yellow Don't Make Green*, Revised Edition. North Light Books, Cincinnati, 1994.

Modeling

Booch, Grady with James Rumbaugh and Ivar Jacobson, *UML User Guide*. Addison Wesley, 1999.

[Coad97a] Coad, "Boundary; Colors; Timeline; Status". *The Coad Letter*. Object International (www.oi.com), September 30, 1997.

[Coad97b] Coad, Peter, "How to Build Better Object Models" Tutorial. *OOPSLA*, Atlanta, October 1997.

[Coad92] Coad, Peter and Mayfield, Mark, "Object-Oriented Patterns". *Communications of the ACM.* September 1992.

[Coad95-97] Coad, Peter with Mark Mayfield and David North, *Object Models: Strategies, Patterns, and Applications*. Second Edition. Prentice Hall, 1997.

Curran, Thomas and Gerhard Keller with Andrew Ladd, *SAP R/3 Business Blueprint*. Prentice Hall, 1998.

Fowler, Martin, *Analysis Patterns*. Addison Wesley, 1996.

Fowler, Martin, with Kendall Scott, *UML Distilled*. Addison Wesley, 1997.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns*. Addison Wesley, 1995.

Words

[Haykawa68] Hayakawa, S.I., Editor, *Use the Right Word*. Now published under the title *Choose the Right Word*. Reader's Digest, Pleasantville, 1968.

[Webster78] *Webster's New Twentieth Century Dictionary*. Collins World, 1978.