Cascade

Ted Foster
ted@class-sc.demon.co.uk
Class Software Construction Ltd., U.K.

Liping Zhao
liping@cs.rmit.edu.au
Department of Computer Science, RMIT, Australia

ABSTRACT

Cascade is a generic pattern for layering and ordering the parts of a complex whole. Each layer is itself a Composite pattern which composes objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. Cascade makes the layering and ordering of objects and their compositions explicit. Cascade has the ability to treat one Composite layer as a primitive Component of its parent layer and hence allows us to express the relationship between parts and wholes naturally: a whole for one particular layer is simply a part for the next layer up. In our modelling of transport systems, we have observed frequent recurrence of the Cascade pattern. We believe that many real world problems exhibit the features of Cascade. In this paper, we illustrate Cascade with more examples from transport systems.

INTRODUCTION

"Living systems are organised in such a way that they form multileveled structures, each level consisting of subsystems which are wholes in regard to their parts, and parts with respect to the larger wholes… All these entities — from molecules to human beings, and so on to social systems — can be regarded as wholes in the sense of being integrated structures, and also as parts of larger wholes at higher levels of complexity. In fact, we shall see that parts and wholes in an absolute sense do not exist at all" [Capra82, p. 27].

In our modelling of transport systems, we have observed many manifestations of such layered structures. For example, a route [Zhao+96] on a transport network is made up of route components at different levels; a route component for a particular layer is a part for the next layer up, a whole for the next layer down. Similarly, a driver duty [Zhao+98] and a driver duty builder [Zhao+97] are made up of other components that represent either parts or wholes for any one level within its overall structure.

In this paper, we present Cascade as a generic pattern, which can be used to structure parts and wholes in complex hierarchies. We use **points** on a transport network and the **links** between points to illustrate Cascade. A point may appear to be a simple concept in a transport network, the smallest component corresponding to a point in space. However, it is "amazingly complex" and its contexts are "astonishingly rich" [Riehle97]. As a whole, a point can play many different roles in a transport network and we have modelled this wholeness view using our Point pattern [Foster+96, Zhao+96]. However, being a point is much more than just being a 0-dimensional point in space; as our view expands, the inner structure of our Point pattern reveals another Cascade [Foster+97], a whole of integrated and layered parts. The concept of a point is uncoupled from its role as a spatial location.

Similarly, a link is more than just a spatial connection between two points. Here we present a microscopic view of Point and Link.

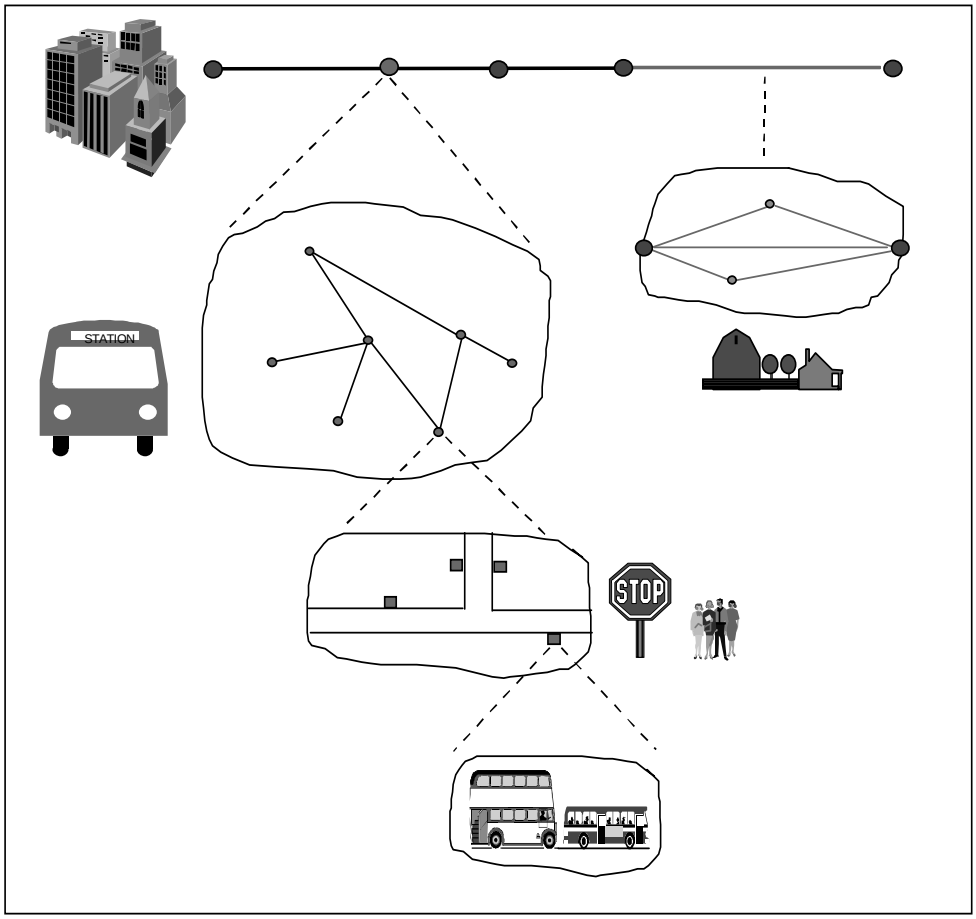

Figure 1. Transport Networks are Multileveled

CASCADE

Intent

The Cascade pattern is used for layering and ordering the parts of a complex whole. Each layer is itself a Composite pattern which composes objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. Cascade makes the layering and ordering of objects and their compositions explicit.

Problem

A point on a network can represent a whole town, or place within a town, an individual vehicle stop, or the bay position at a stop depending on the level of granularity of interest to an application. A point may be viewed as a complex point or a simple point at any one level of abstraction. A complex point may be made up of one or more simple points at the next level down; it may also be a part of an even more complex point at the next level up. Similarly, links between points on a network can represent the spatial or temporal connection between these different points. Thus points and links are bound together into a complex multileveled structure (Figure 1). In both point and link, an object of one type can be viewed as an aggregation of

more than one other type of object at the next level down; both a point or a link can become a collection of other links and/or points. Since any one point or link on a network may play many different roles and may be wholly or partially contained within higher level objects, there is an almost infinite number of possible ways of viewing them. Areas bounded by links and containing points either on or within their boundaries are also layered structures. So too are solid structures in 3-dimensional models of a transport network. A very flexible structural pattern is required for modelling them.

The Composite pattern [Gamma+95, p. 163] can be used for this purpose, but we have found that domain experts sometimes feel uncomfortable because the natural layering and ordering of their world is not made sufficiently explicit by Composite. Furthermore, the structure of the real world problem can be difficult to communicate to programmers with a single composite.

What can be done to improve the visibility of the many possible hierarchies than exist between parts making up such aggregate structures?

Solution

– Structure

Represent each layer of an aggregate structure by its own Composite pattern. Replace one of the leaves of the Composite at the upper level by a complete Composite to represent the whole layer at the next level down. If required, one of the leaves of this lower layer can then be replaced by another complete Composite, and so on ad infinitum (or in practice until it accurately models all the levels of interest).

Figure 2 illustrates a Cascade of three Composites in three layers. The Component class A is an abstract interface for the Composite object A and Leaf objects A1, A2, … in which A2 has been replaced by the Composite pattern B. The Component class B is an abstract interface for the Composite object B and Leaf objects B1, B2, … in which B2 has been replaced by the Composite pattern C. The leaf objects in any one layer can be instances of different classes if required or multiple instances of the same class but must be type conformant with their parent Component class.

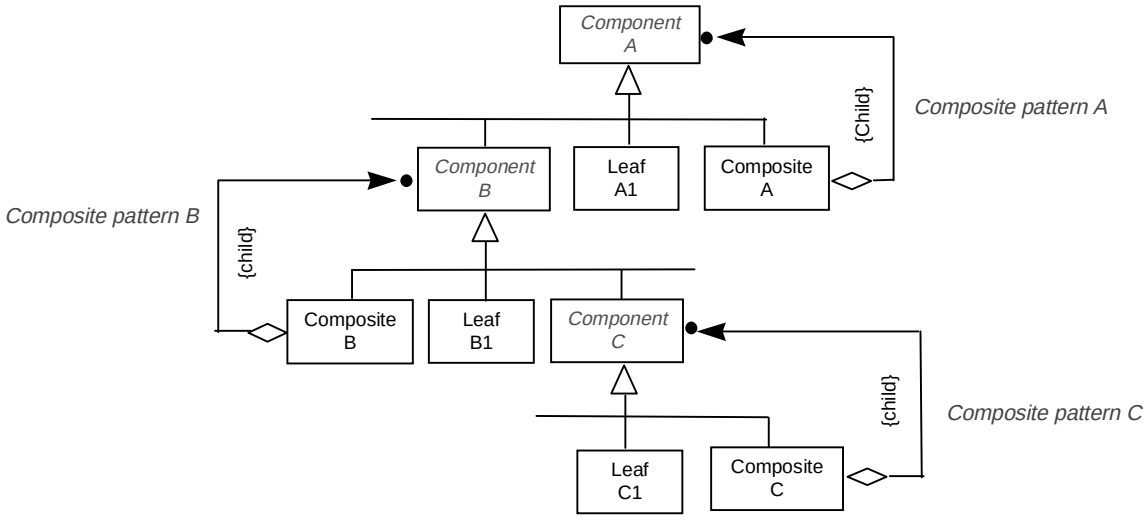

Figure 2. Cascade of Three Composites in Three Layers

Since each Composite class is an aggregation of its own parent Component objects, it follows that a Composite object can be a collection of other Composite objects like itself. In other words, Composite object A can be a collection of other Composite objects A. However, in some situations, we may want to restrict this option. A Composite pattern constrained in this way could also be represented as an aggregation relationship directly from the Composite class to an abstract Leaf class that provides an interface to all the other Leaf classes within that layer (see Figure 3). However, by retaining the classic Composite pattern structure, we can avoid introducing this additional abstract Leaf class. The Component class within the Composite pattern provides the interface for all the leaf classes in that layer.

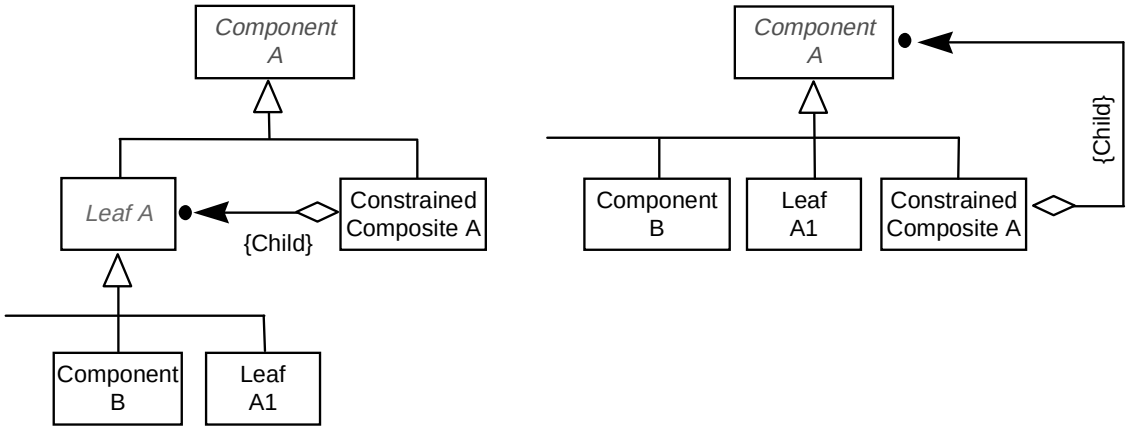

Figure 3. Two representations of a constrained Composite object.

Two leaves on any one level can be replaced by different Composites providing they are both type conformant with their parent Component class. In Figure 4, the Leaf objects A2 and A3 have been replaced by the Composite patterns B and C respectively which are both type conformant with Component A.. In effect more than one Cascade has its root in the Composite object A.

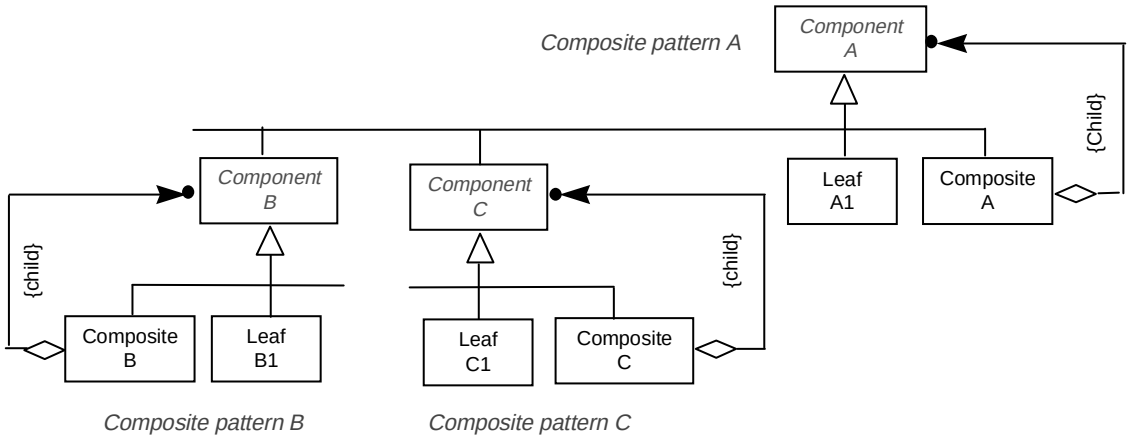

Figure 4. Cascade of Three Composites in Two Layers

Alternatively, the topmost Composite pattern A may be replaced by a simple aggregation relationship between Composite pattern B and Composite pattern C. There is no longer any

requirement that Component objects in B and C be type conformant with a common supertype. Two different representations of part-whole relationships are being used in the same structure. In Figure 5, the Component objects in the hierarchy rooted at Component B are aggregates of the Component objects in the hierarchy rooted at Composite X.

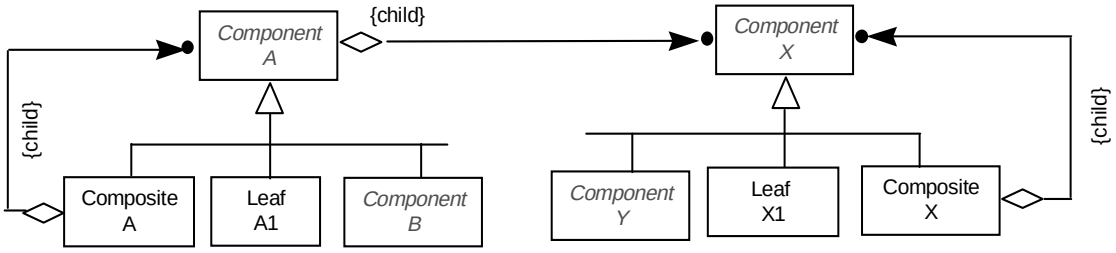

Figure 5. Mixing two different whole-part representations.

The aggregation relationship between a Composite object and its parent abstract Component class is quite often an ordered sequence within transport network models. This is not however a requirement of the Cascade pattern and other containing relationships (e.g., sets and bags) may be more appropriate for the problem at hand..

For this aggregation relationship, we generally favour the precise semantic interpretation proposed in [Cook+94, p.39]:

> "We choose aggregation to mean life-time dependency; in particular, that the life-times of the 'parts' are contained within the life-time of the 'whole'. The 'parts' are permanently attached to the 'whole', and cannot be removed from it without being destroyed. Conversely, destroying the 'whole' destroys the 'parts'. Aggregation is shown as a diamond on the association line adjacent to the type whose instances have the containing life-time (the 'whole' or 'aggregate')".

This life-time dependency continues from the top to bottom of a Cascade of Composite patterns.

– Example

In Figure 6, we show a high level link between two points (Journey Variant) expressed as an ordered aggregate of other lower level links (Timing Link On Variant, Commercial Link On Variant, or Journey Definition Point Link On Variant (JDP Link)).

A Journey Variant describes the way in which one or more journeys operate along routes through the physical network. It describes a journey passing through either an ordered sequence of Journey Definition Points, or across JDP Links. In this example, a link-oriented definition of a Journey Variant is used for the purpose of illustrating a Cascade of Links (compare this with the structure illustrated in Figure 4). Also in this example, the strict life-time dependency interpretation of the aggregation relationships is enforced.

The closely related layering of the physical routes that journeys follow is described in [Zhao+96]. A more detailed example of a Cascade of Links, with more layers and components within each layer representing subdivisions of the working day of a driver, is described in [Zhao+97, Zhao+98].

Partitioning a network into separate parallel Cascades is essential for structuring the many different links in a transport network. If one attempts to define some monolithic structure incorporating all the links of interest to different applications, then the links between any two

adjacent points tend to lose any real world meaning. Partitioning makes it possible to subdivide links so that the sub-links do not overlap the ends of the link at the next level up. For example, a Commercial Link On Variant can not necessarily be subdivided into Timing Links On Variant; nor can a Timing Link On variant be subdivided into Commercial Links On Variant. They are therefore represented as separate parallel Cascades.

The life-time dependency interpretation of the aggregation relationship is important for interpreting these Cascades of Links. For example, a Timing Link On Variant only exists in the context of one particular Journey Variant; it is not the Link per se and can not be shared. However, two or more Timing Links On Variant may refer to the same Link object. It is the Link object, or one of its role objects, that is shared and not the Timing Link On Variant. The responsibilities assigned to a Timing Link On Variant are those which refer to a Link but are only meaningful in the context of a particular Journey Variant, for example, journey running times.

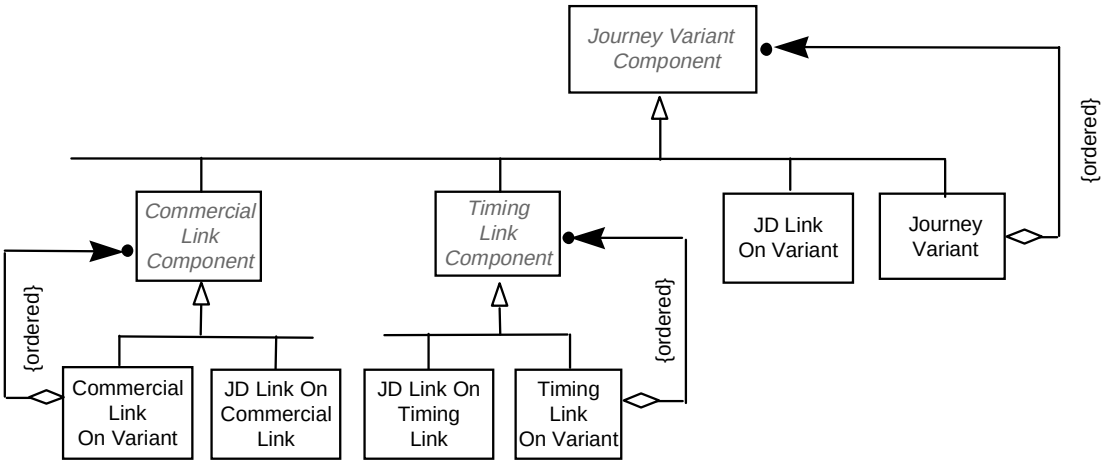

Figure 6. Parallel Link Cascades for parts of a Journey Variant.

In practice, a point-oriented definition of links is often preferred to a link-oriented one, especially if some of the links between adjacent points in the Cascade have no real world interest. Deeply nested links within links are difficult to picture within one's mind. Separate parallel cascades overcome the necessity for such deep nesting and make it easier to work with link-oriented definitions.

Points as well as links can be cascaded. Some of the journey definition points are commercial points, published places to and from which passengers are offered transport services. They can be individual stops or collections of stops. For the purposes of this illustration of Cascade, we shall define a commercial point as a collection of stops. Furthermore, we will assume that each stop can be further sub-divided into stop bays so that more than one vehicle can park there. Timing points are used for expressing timing information. Figure 7 shows the sub-division of commercial point into stop points and stop bays expressed both as a Composite and as a Cascade.

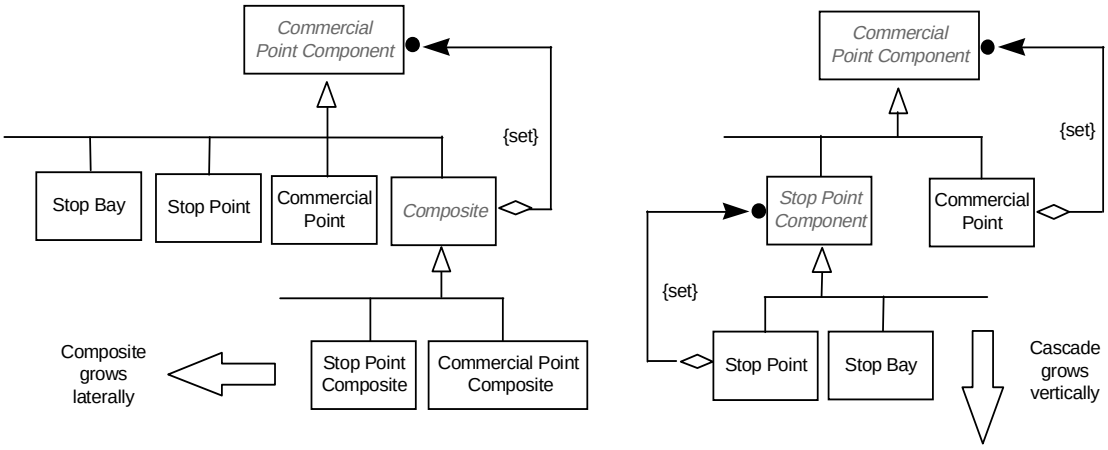

Figure 7. A Composite and Cascade of Commercial Point Components.

The layering of points and links within the same Cascade can be achieved by defining a common Network Component supertype to which they both conform. A Network Composite is defined as an aggregate of these Network Components. However, in practice, people usually find it easier to separate them and express Link Components as an ordered aggregate of two or more Point Components. This is illustrated in Figure 8 (compare this with the structure illustrated in Figure 5). The Point component objects are in fact Point On Link objects and not the Point per se. In effect, this diagram is adding point-oriented definitions of links to link-oriented definitions of links. Point-oriented definitions of links support some algorithms better than link-oriented definitions.

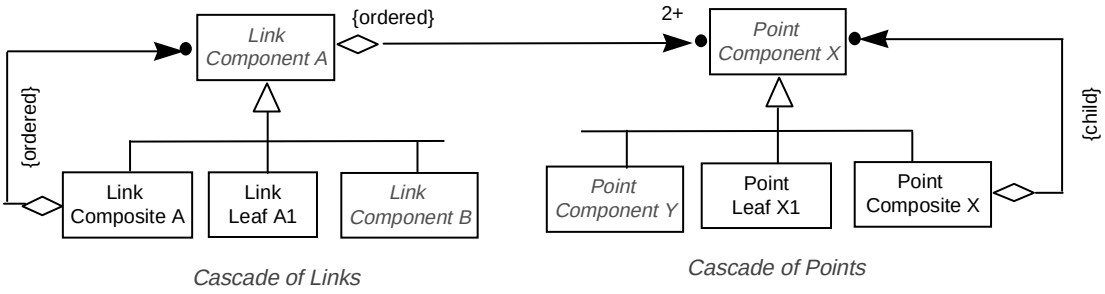

Figure 8. Introducing point-oriented definitions of links..

Once again we emphasise the importance of the life-time dependency interpretation of the aggregation relationship. The Point Component objects in Figure 8 only exist in the context of one particular Link; it is not the Point per se and can not be shared. However, two or more Point On Link objects may refer to the same Point object. It is the Point object, or one of its role objects, that is shared and not the Point On Link. The responsibilities assigned to a Point On Link are those which refer to a Point but are only meaningful in the context of a particular Link, for example, Journey Waiting Times for a Timing Point On Journey Variant.

– Participants and Collaborations

Consider the Cascade in Figure 2.

- Components (Component objects A, B, and C)

  - Declare a uniform interface for Component objects in the respective Composite patterns.
  - Implement default behaviour for interfaces common to the respective Component objects.
  - Declare an interface for managing the ordered sequence of the respective Component objects.
- Composites (Composite objects A, B, C)
  - Define behaviour for compositions of their Component objects.
  - Store an ordered sequence of their Component objects.
  - Implement operations for managing this ordered sequence of Component objects.
  - Make compositions of objects based on client requests.
- Leaves (Leaf objects A1, B1, C1)
  - Define behaviour for leaf objects.
  - Handle the requests from clients.
- Clients
  - Manipulate objects in the compositions through Component interfaces.

Context

Use the Cascade pattern when you want to:

- Represent part-whole relationships between objects and express the relationships between parts and wholes explicitly.
- Ignore the difference between objects and their compositions, and treat them uniformly.
- Represent the natural layering and ordering of your application world explicitly.

Pattern Interactions

- **Composite**
  Each layer in Cascade is represented by one Composite [Gamma+95, p. 163].

- **Chain of Responsibility**
  Composites in Cascade form a Chain of Responsibility [Gamma+95, p. 223]. For example, in Figure 2, Component objects are Abstract Handlers and Composite and Leaf objects are Concrete Handlers. Component C is the successor of Component B, which in turn is the successor of Component A. When a client issues a request, the request propagates along the chain until a Concrete Handler can handle it.

- **Flyweight**
  "The Flyweight pattern is often combined with the Composite pattern to implement a logically hierarchical structure in terms of a directed-acyclic graph with shared leaf nodes" [GOF p206]. Point On Link and Link On Link objects can be implemented using the Flyweight pattern, so that logically there will appear to be an object for every Point On Link and Link On Link in the network, even though physically there will be only one shared flyweight for each Point and Link (or more accurately one of their role objects). However, our existing implementations of Cascades define Point On Link and Link On Link objects explicitly because these concepts exist in many public transport data models. For example, in data modelling terms, a Point On Link is an intersection entity to resolve a many-many relationship between a Point and a Link.

- **Role Object**
  In [Foster+96, Zhao+96] we described Point as an aggregation of one or more Point Roles and Link as an aggregation of one or more Link Roles. The Role Object pattern is discussed in more detail in [Bäumer+97] where the need to provide an abstract interface for a Component and Component Role is very clearly and rightly stressed. It is also worth adding that there is also an interesting interaction between Role objects and Part-Whole hierarchies. In data modelling terms a Link Role is an intersection entity to resolve the many-many relationship between Link and Link Type. This differs from a Link On Link which is representing the relationship between a part and its whole. Any one Link object can be simultaneously playing the role of a Timing Link and a Journey Variant but only if it is a Timing Link on a different Journey Variant. The placing of an object in a part-whole relationship with another object is in effect defining the roles of the object. An object's place in the world requires that the object play certain roles.

Forces and Design Rationale

Why do we need Cascade?

The forces that drove us to Cascade have been discussed in our Route, Driver Duty, and Driver Duty Builder patterns [Zhao+96, Zhao+97, Zhao+98]. Here we present these forces in a more general form.

- *Cascade provides consistent interfaces to parts and their wholes.* The relationship between parts and wholes was traditionally represented as an aggregate [Rumbaugh+91, p. 59]. However, the problem with this representation is that clients have to treat parts (Components) and wholes (Compositions) differently. This makes the program more complex. The Composite pattern [Gamma+95, p. 163] has overcome this problem by providing an abstract class (Component) that represents both parts and wholes. Since Cascade builds upon Composite, it also inherits this feature of Composite.

- *It makes the layering of objects and their compositions explicit.* In the Composite pattern, different types of Composite are supported via decomposing or sub-typing the Composite classes; but all the primitive objects or leaves are treated at the same level. In Cascade, primitive objects appear in layers corresponding to their positions in real world multileveled structures. Composites grow laterally, whereas Cascades grow vertically (Figure 9).

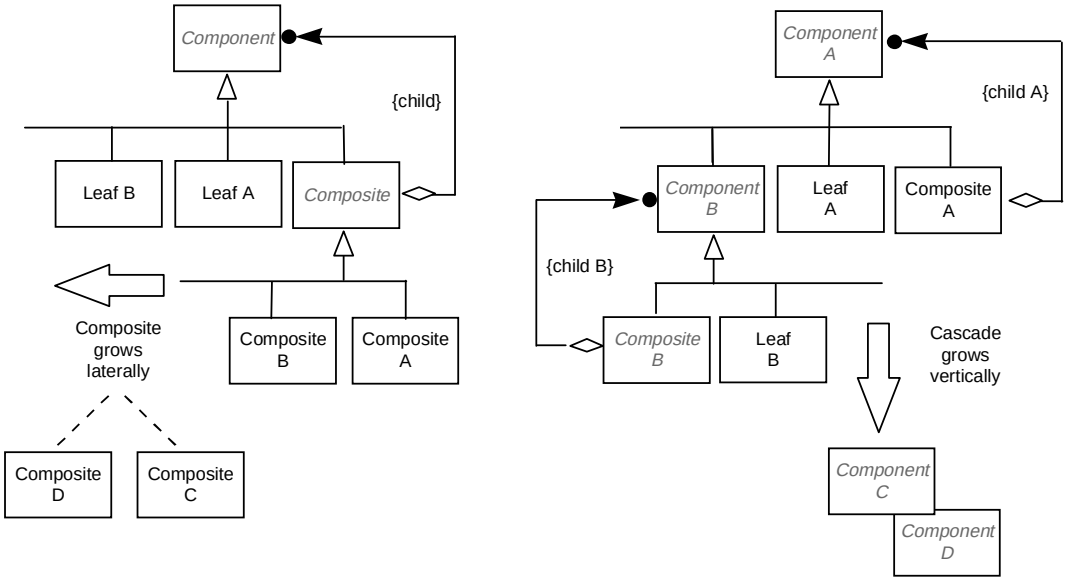

Figure 9. Composite versus Cascade growth.

- *Any ordering of objects can be made explicit with an appropriate parent-child association between each Composite and Component class within a Cascade.* This is not usually done within a traditional aggregation diagram.

- *It can represent the parent-child relationship between a Composite and a Component for each layer differently.* Sometimes it is useful to have different program structures for different layers. With Cascade, we can represent the relationship between a Composite and Component as, for example, a linked list at one level, a set at another.

- *It allows its parts to preserve their individual autonomy and at the same time facilitates their togetherness.* The ability to treat one Composite layer as a primitive Component of its parent layer allows us to express the relationship between parts and wholes naturally: a whole for one particular layer is simply a part for the next layer up. "The whole supports the parts which themselves constitute the whole" [Davies95, p 221].

- *A constraint can be imposed within Cascade that prevents a Composite from having another Composite of the same type as one of its children.* This means that a Composite object for any one layer is a collection (e.g., ordered sequence) of leaves and not a tree; one of the leaves is an abstract class (i.e., Component) for the next layer down. For example, Composite A in Figure 2 need not have another Composite A as a child. However, indirectly through the abstract interface provided by Component B, it can parent another type of Composite B. In this way we treat any one Composite layer as a primitive Component of its parent layer.

- *It simplifies the specification of the interface for Component classes.* For any one layer, the Component interface only needs to be able to respond to message calls invoking operations within that layer and not those that have already been implemented in higher layers.

- *Composites can be slotted in and out of a Cascade as easily as Leaf objects in a Composite.* Entire layers represented by Composites can be "plugged in" without impacting

on the rest of the system providing the new component can still respond to the same message calls as the original.

- *The roles played by generic objects can be assigned to different levels and subsystems.* In other words, Cascade is making explicit the different roles played by parts within the whole and where the responsibility for playing those roles resides.

- *Cascade may require extra coding.* This is a trade-off for a more explicit specification. A clearer specification of patterns can facilitate the communications between domain experts and software designers; it can also help programmers to correctly implement patterns.

- *Cascade focuses on the interaction between one or more Composites and therefore the discussion of forces in the implementation section of the Composite pattern in [Gamma+95, p.166] applies here too.* However, there is no need to repeat the issues discussed by the GOF, except perhaps to point out that we have opted for transparency rather than safety in assigning the responsibilities for the child management interface at the root of the class hierarchy. This reflects the high priority that we place on achieving transport software systems that can grow.

- *The very strict life-time dependency interpretation of the aggregation relationship may appear to be too constraining for some real-world situations.* For some applications, the properties or behaviour of a part may not vary with its presence or position within a whole at the next level up. For example, it may suffice for some applications to define a default running time that would apply to all journeys crossing a particular link. A shareable Link or Timing Link object rather than an unshareable Timing Link On Variant object is enough to meet this requirement. Several variations in the interpretation of this relationship are described in [Buschmann+96, p.233-234]

  > "Part objects all have the same type. Parts are usually not coupled to or dependent on each other. You can apply this variant when implementing collections such as sets, lists, maps, and arrays"
  >
  > [Collection-Membership relationship]
  >
  > "This variant relaxes the constraint that each Part must be associated with exactly one Whole by allowing several Wholes to share the same Part. The life-span of a shared Part is then decoupled from that of its Whole".
  >
  > [Shared Parts relationship]

  Choosing the appropriate variant or combination of variants is determined by the specific semantics of the real-world objects in the Cascade and the requirements of the application.

Similar Patterns and Applications

The Cascade pattern recurred over and over again in our modelling of transport systems. For example, Route [Zhao+96], Driver Duty [Zhao+98], and Driver Duty Builder [Zhao+97] are Cascades. In this paper, we have given an example of a Cascade for Journey Variant and Commercial Point components. Points (0-dimensional) and Links (1-dimensional) are the basic components of a network. We have not yet described areas (2-dimensional) and solids (3-dimensional), but of course Cascade and other related patterns will be needed even more for layering these even more complex network objects.

The Whole-Part pattern in [Buschmann+96, p.225] refers to several variations arising from progressive relaxation of the constraints in the interpretation of the aggregation semantics.

There is a reference to the GOF Composite as one such variant but no explicit focus on the Cascading of one or more such Composites.

The Role Object pattern in [Bäumer+97] describes a pattern for representing the multiple concurrent roles that a component can play and provides a good setting for considering the interaction between part-whole hierarchies and roles. This paper also discusses the recursive use or cascading of role objects.

FINALE

Through patterns like Cascade we are seeking to balance a "hierarchical reductionist" approach with a holistic approach. Hierarchical reductionism "explains a complex entity at any particular level in the hierarchy of organization, in terms only one level down the hierarchy" [Dawkins, p.13]. Holism places a strong emphasis on the functioning of the parts as a whole at the next level up.

"The kinds of explanation which are suitable at high levels in the hierarchy are quite different from the kinds of explanation which are suitable at lower levels" [Dawkins, p.13] and therefore we like to make each layer very explicit within our software simulations. We believe that Cascade will be a useful pattern for modelling many real world problems in other domains as well.

ACKNOWLEDGEMENTS

We wish to thank all the people involved in various Science and Engineering Research Council (UK) and European Union (EU) funded projects and others with whom we have discussed the problems of modelling public transport operations; and our two students at RMIT University (Australia), Lance Dourlay and Bradley Kazazes, for implementing applications of Cascade during their vacation. Special thanks are also due to Alejandra Garrido, our PLoP shepherd, in particular for some of her observations on the semantics of sharing parts.

REFERENCES

[Bäumer+97] D. Bäumer, D. Reihle, W. Siberski, and M.Wulf. "The Role Object Pattern". Submitted to *PLoP97.*

[Buschmann+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M.Stal. *A System of Patterns: Pattern-Oriented Software Architecture*. John Wiley & Sons, 1996.

[Cook+94] S. Cook and J. Daniels. *Designing Object Systems: Object-Oriented Modelling with Syntropy.* Prentice Hall, 1994.

[Capra82] F. Capra. *The Turning Point.* Simon & Schuster, New York, 1982.

[Dawkins88] R. Dawkins. *The Blind Watchmaker.* Penguin Books, England, 1988.

[Davies95] P. Davies. *Superforce.* Revised Edition Penguin Books, England, 1995.

[Foster+96] T. Foster and L. Zhao. "Modelling Transport Objects with Patterns". *Presented at EuroPLoP96,* 1996. *Object Expert Vol. 2(5), July/August 1997* and also to appear in *Journal of Object-Oriented Programming,* 1997. SIGS Publications.

[Foster+97] T. Foster and L. Zhao. "Structuring The Network Model To Service More Functions". Internal Report for the EU Titan project.

[Fowler+97] M. Fowler "Dealing with Roles". Submitted to PLoP97.

[Gamma+95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley, 1995.

[Riehle97] D. Riehle. "Part 8: Domain Specific Patterns". In *Pattern Language of Programming Design 3.* Addison-Wesley, 1997.

[Rumbaugh+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.

[Zhao+96] L. Zhao and T. Foster. "A Pattern Language of Transport Systems (Point and Route)". *Proceedings of PLoP96,* 1996. To appear in *Pattern Language of Programming Design 3.* Addison-Wesley, 1997.

[Zhao+97] L. Zhao and T. Foster. "Driver Duty Constructor".. Submitted to PLoP97

[Zhao+98] L. Zhao and T. Foster. "A Pattern Language of Transport Systems (Driver Duty)". To appear in *Journal of Object-Oriented Programming,* 1998.