PLOTS: pattern language of transport systems

– Point and Route Patterns –

Liping Zhao
Department of Computer Science
Bundoora East Campus, RMIT
Plenty Road, Bundoora
Victoria, 3083 Australia
liping@cs.rmit.edu.au

Ted Foster
Class Software Construction Ltd
4 Duchy Road, Harrogate
England HG1 2EP
ted@class-sc.demon.co.uk

Abstract

A point in a transport network plays many roles. The Point pattern provides a flexible structure that decouples a point's role from the point itself and supports multiple concurrent roles. Points (or the links between points) are used to define routes. A route is a path through an ordered sequence of points (or links) in a transport network. A route may build upon different layers of components. The Route pattern provides a consistent tree structure for layering and ordering the route components. Route can cope with many variations in defining routes and allows clients to treat all objects used for route definition uniformly. Point and Route are patterns in PLOTS, a pattern language for building transport systems. This paper shows how Point and Route provide a design solution for describing the spatial structure of a transport network, and how they interact with other patterns in PLOTS.

1. Introduction

PLOTS is a pattern language for building transport systems. The PLOTS language is a collection of TOPS (transport object patterns) [4] for solving transport problems.

PLOTS focusses on pattern interactions, as "none of the individual patterns in this system are isolated. Each pattern depends on the smaller patterns it contains, on the patterns it interacts with, and on the larger patterns it is contained within" [1].

TOPS are higher level patterns in that they build upon other patterns [2, 6]. Some of TOPS are domain specific, whereas others are more generic. PLOTS intends to show how these patterns collaborate with each other to achieve specific tasks for transport applications. "It is through pattern languages that patterns achieve their fullest power" [3].

In this paper, we first present the Point pattern, which is the most basic component in a transport network. We then show how the Point pattern is used in the construction of the Route pattern. We have defined our own pattern form for describing TOPS.

2. The Zhao-Foster Pattern Form

Our pattern form is adapted from the Gang-of-Four [6] and Coplien pattern forms [3], which includes the following sections.

- **Pattern Name:** Same as Gang-of-Four and Coplien
- **Intent:** Same as Gang-of-Four
- **Also Known As:** ditto
- **Problem:** The real world problem that the pattern aims to solve. This section is often started with a question and followed by a detailed description of the problem and previous attempts to solve the problem, if any. It sets up a background for you and helps you to understand the solution to the problem better.
- **Solution:** A design solution to the problem, containing the following aspects.
  - **Structure** A diagram showing the pattern structure and a description of the pattern
  - **Example** To illustrate the use of the pattern
  - **Participants** Same as Gang-of-Four, but absorbed into our solution
  - **Collaborations** ditto
  - **Implementation (Optional)** ditto
  - **Sample Code (Optional)** ditto
- **Pattern Interactions** How does the pattern interact with other patterns? What is the purpose of the interaction? What are the underlying patterns of the pattern under discussion, if any?
- **Applicability:** The context in which the pattern is applied, same as Gang-of-Four and similar to Coplien's context.
- **Forces:** Same as Coplien's forces and resulting context.
- **Design Rationale:** Same as Coplien.
- **Similar Patterns and Applications:**

3. The Point Pattern

Intent

A point in the transport network plays many roles. Point provides a flexible structure that decouples a point's role from the point itself and supports multiple concurrent roles played by the point.

Also Known As

Node.

Problem

How do we represent points in a transport network?

A point is a node on the graph of a transport network. It is the smallest component in the transport network and may play multiple concurrent roles. For example, a point role can be:

- a network point, or
- a route definition point, or
- a junction point, or
- a journey definition point, or
- a garage, or

- a reserve point, or
- a stop point, or ... ...

In the past we used a class hierarchy to represent these roles. Figure 1 illustrates this representation. It shows that even with just seven different roles, we soon reached five levels of inheritance (n.b., in a full description of the transport network, a point plays many more roles than these).

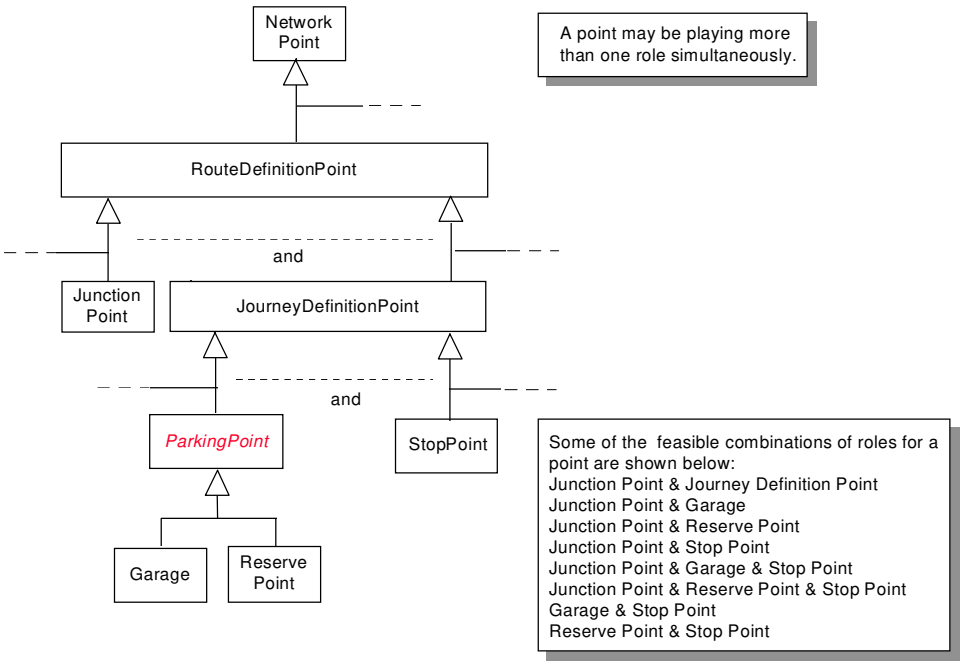


Figure 1. A class hierarchy for point roles

In Figure 1, RouteDefinitionPoint and JourneyDefinitionPoint are not abstract classes. We can have (1) a RouteDefinitionPoint that is neither a JunctionPoint nor a JourneyDefinitionPoint, and (2) a JourneyDefinitionPoint that is neither a StopPoint nor a ParkingPoint. Figure 2 shows multiple overlapping point roles in a Venn diagram.

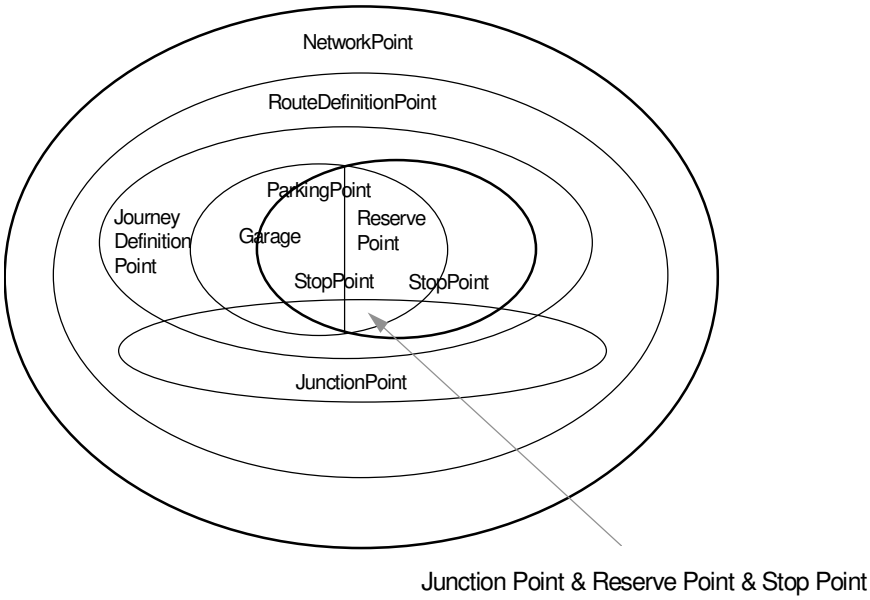


Figure 2. Venn diagram showing multiple overlapping point roles

When one considers all the myriad of other things which can happen at a point, then this type of classification hierarchy gets very complicated indeed. There is also a lot of implementation inheritance taking place in this structure. Such an inheritance monster is impossible to maintain and hard to reuse.

Solution

– Structure

The Point pattern is shown in Figure 3. We represent a point object as an aggregation of one or more point roles [4]. All the other roles that a point plays will be defined as specializations of the abstract class PointRole. A point will hold references to all the instances of its concrete role classes. This point will delegate to these concrete role classes.

A point's role in the network is changeable and not a property of the point itself. Anything that is changeable, or is not a direct component of an object, should always suggest an indirect composition. Point should have a relationship with a separate class from which each specific role is derived. In this way, the various behavioural traits supplied by each of the roles can be accessed through a point by delegation.

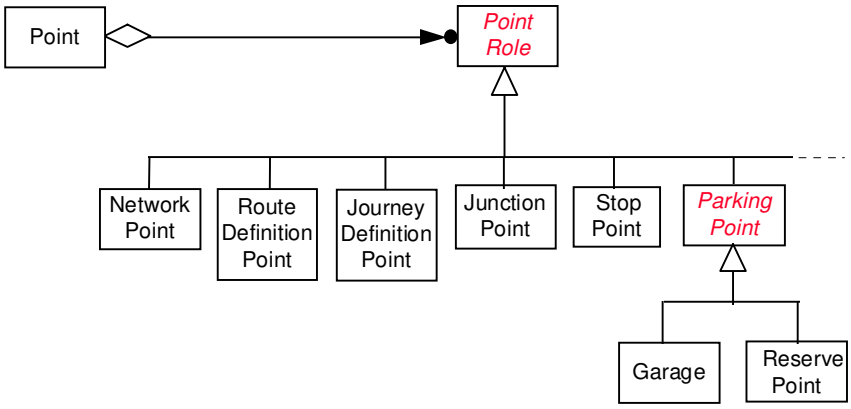


Figure 3. The Point pattern

– Participants

- **Point**
  - defines the Point interface to clients
  - holds references to all the instances of its concrete point roles
  - delegates the request from clients to specific point roles
- **PointRole**
  - defines an interface for encapsulating the behaviour associated with a particular point role
- **ConcretePointRoles** (NetworkPoint, RouteDefinitionPoint, JunctionPoint, ...)
  - each point role implements a behaviour delegated by Point
  - can be in different subsystems (Figure 4)

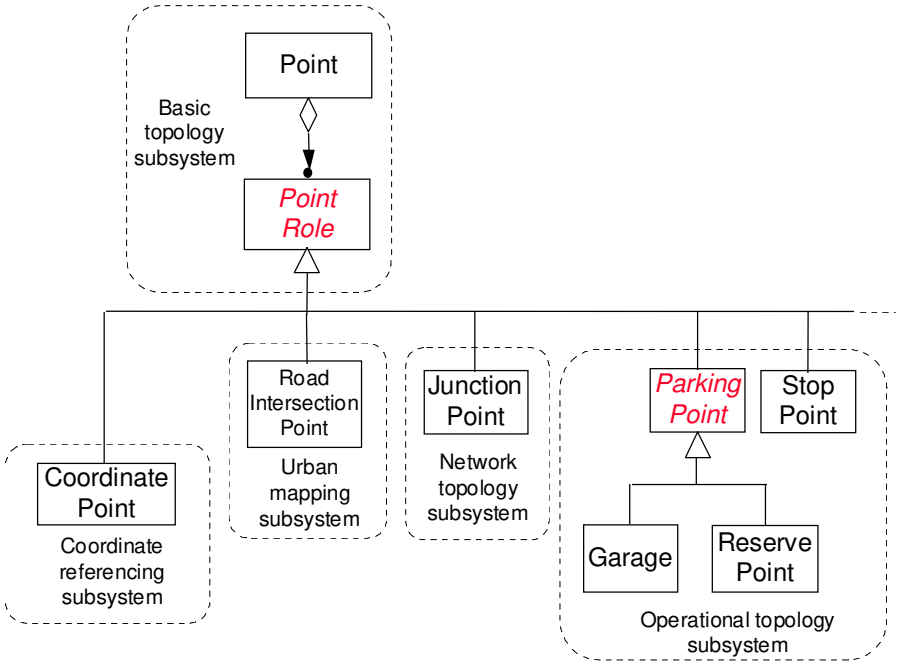


Figure 4. The concrete roles played by points can be in different subsystems

Pattern Interactions

– Role or State

The underlying pattern of Point is Role [2] or State [6]. Point extends Role/State to support the multiplicity of roles.

The Role/State pattern is in our opinion particularly suitable for modelling independent objects like point. Objects that are defined upon other objects like route, section, and link imply a layered tree structure and cannot simply be modelled as roles. We have defined a pattern [11] for cascades of roles.

– Route

A route is a path through an ordered sequence of points in a transport network. A route can also be constructed using routeComponents (i.e., basic links between its points) which are clients of the Point pattern. For a detailed description of the Route pattern, see section 4.

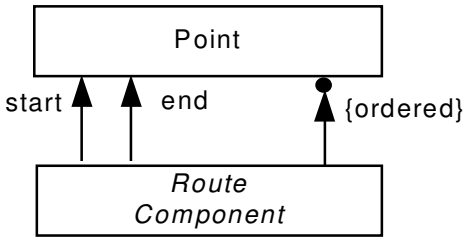


Figure 5. Pattern interaction: RouteComponent as a client of Point.

Applicability

Use the Point pattern to structure roles played by a point in a transport network. This pattern has the following features.

- It can cope with the addition of new roles as they are required without disruption to the existing specification.
- It reifies variant behaviour of a point and allows a change of point roles at run time.

Forces

The proliferation of lots of point roles within our system does involve some overhead. We must always question whether the increased flexibility provided by the Point pattern justifies the departure from other ways of representing roles such as using attributes, which are familiar and therefore more readily accepted by traditional data modellers.

As in many aspects of design, we feel that the answer to this question does not lie in opting for one particular method of representation, rather it requires a balancing act between different methods. In cases where the roles do not have any influence on the behaviour of the software system, then attributes will suffice, as quite often, public transport operators only want a description of the roles their transport objects play for presentation purposes.

The big pay off from reifying roles comes when different behaviours occur in different roles. As soon as this occurs in the software development process, it is advisable to create a new concrete subclass of PointRole and abandon the use of attributes.

Design Rationale

The Point pattern copes with the numerous concurrent roles played by a point or node in a transport network and allows roles to be changed much more easily than using multiple inheritance. It overcomes the restricton imposed by many object-oriented languages where an object can only belong to one class which can not be changed; the point object appears to belong to many classes which can be changed even at run time.

Similar Patterns and Applications

Point is a domain specific application of the Role pattern [2] or State pattern [6] which have been used in many different applications. Odell (pp. 407-410 [7]) used a similar technique, which he called object slicing, for multiple dynamic classification of objects. This was incorporated in the Ptech CASE tool which we used in our first object model of this domain. The movement of a vehicle through its various states during daily operations can be represented this way.

4. The Route Pattern

Intent

To provide a consistent tree structure for layering and ordering the components that define a route in a transport network. Route can cope with many variations in defining routes and allows clients to treat all objects used for route definition uniformly.

Also Known As

RouteVariant.

Problem

How do we represent routes in a transport network?

A route is a path through an ordered sequence of points (or links between those points). Four closely related routes belonging to a single service line are shown in Figure 1:

A-C-D-E-F B-C-D-E-F
A-C-D-E-G-H B-C-D-E-G-H

In this example, all these points are stopPoints (e.g., bus stops) where passengers can board and/or alight from a vehicle. The links between adjacent stopPoints have the role of interstopLinks.

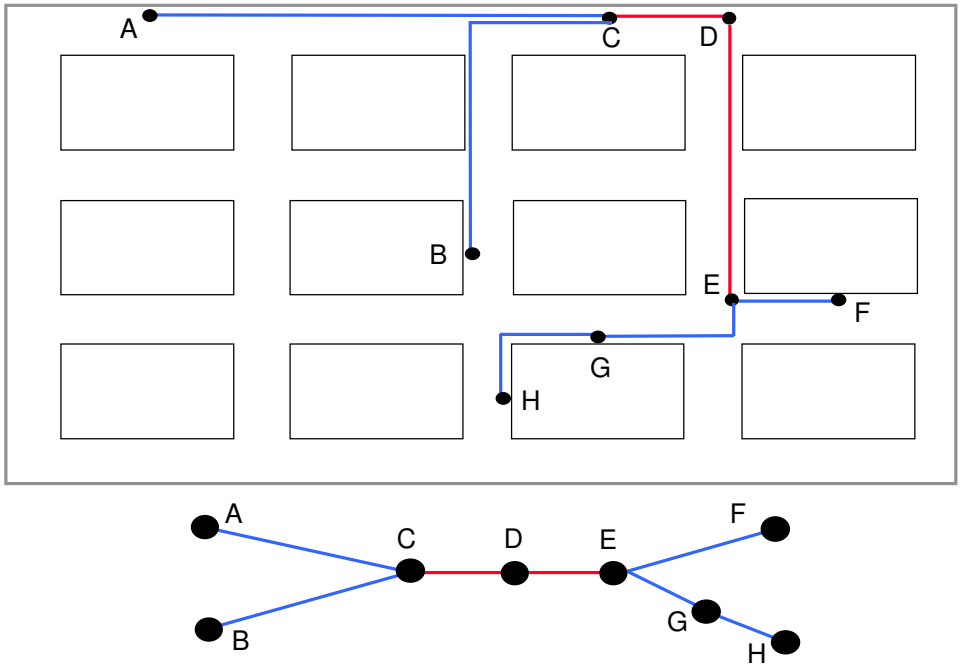


Figure 6. Routes through stopPoints.

Points are the most basic components of a route. Particular roles can be assigned or removed from a point and yet it is still the same point. This is not the same for links. The roles played by a link are determined by the roles of its begin and end points. Points are components of a link so changing the role of the point at the beginning or end of a link changes its role. This is one of the reasons why some transport operators prefer to define routes in terms of points rather than links. However others do have link based definitions which we think need to be catered for as well as point based definitions.

Some transport operators introduce even higher level concepts than links between stopPoints to describe a route. For example, the concept of a section for the path between two junctionPoints, or between a junctionPoint and a terminalPoint is introduced. In figure 7, there are five sections (AC, BC, CE, EF and EH) between terminal points (A, B, F and H) and junction points (C and E). In other words, the idea of layering is introduced. The sections are made up of links.

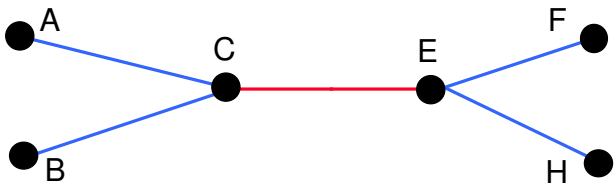


Figure 7. Sections defined using stopPoints.

If a section is defined using only links that are wholly contained within it (i.e., no link stretches across its ends), then a route can be defined as a 2-tier structure made up of sections and links as shown in figure 8. This representation is accepted by some companies, but not others. Some companies (or different departments within the same company) may want to view a route as having sections but not links; or links but not sections; or consider links at a finer level of detail (i.e., consider points hitherto ignored) and create a further layer of sublinks. General acceptance of one particular representation is very hard to achieve.

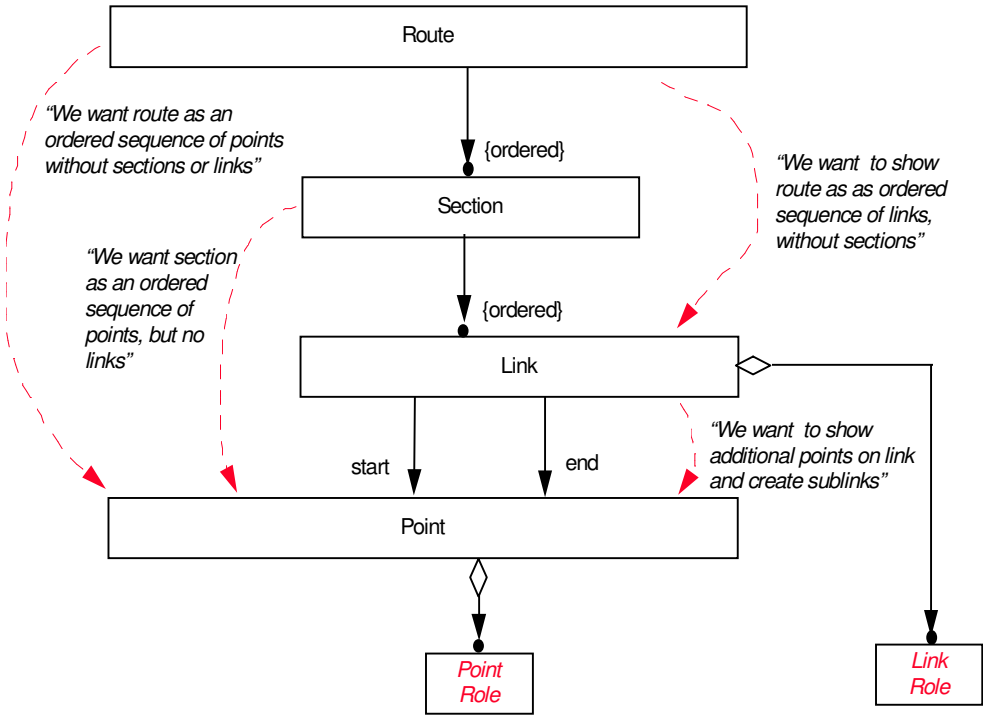


Figure 8: Different ways of defining a route.

In figures 6 and 7, links and sections are defined solely in terms of stopPoints and interstopLinks. This may be sufficient for some applications (e.g., off-line vehicle scheduling), but not others (e.g., automatic vehicle monitoring and real-time control). For those who want to plot routes on street maps, points

corresponding to the ends of roads (role: roadIntersectionPoint) have to be introduced.

As new points or point roles are introduced to support further applications, new links and sections can be defined. For example, in figure 9, a new point X (not a stopPoint) has been introduced where links AC and BC physically meet. Three new links are created (AX, BX and XC). Since X is not a stopPoint, none of these new links play the role of interStopLinks. Some new link role name is required.

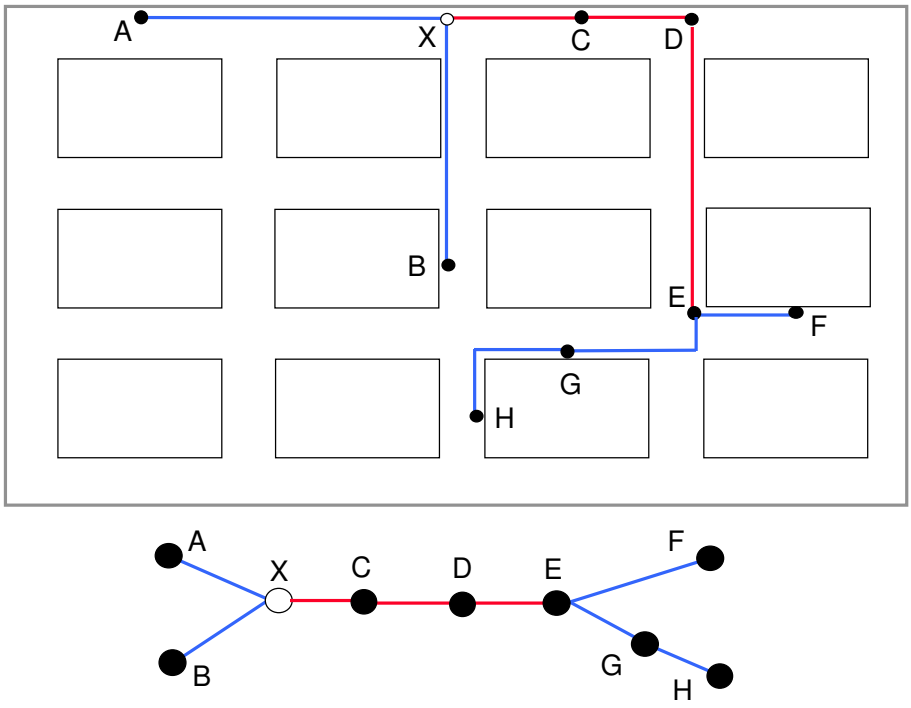


Figure 9. Routes through stopPoints and one non-stopPoint.

Our definition of a section may also need to be revised. If we define a junctionPoint role as the point where two links (role: interStopLink) meet, then points C and E (Figure 6) are candidates for this role. This produces five sections (AC, BC, CE, EF and EH in Figure 7).

If, however, we define a junctionPoint role for a point that is not a stopPoint (e.g., X in Figure 9), then this also produces five sections (AX, BX, XE, EF and EH in Figure 10), but three of them differ from those in the first example.

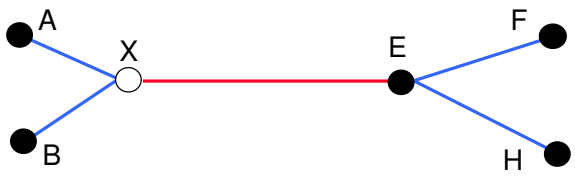
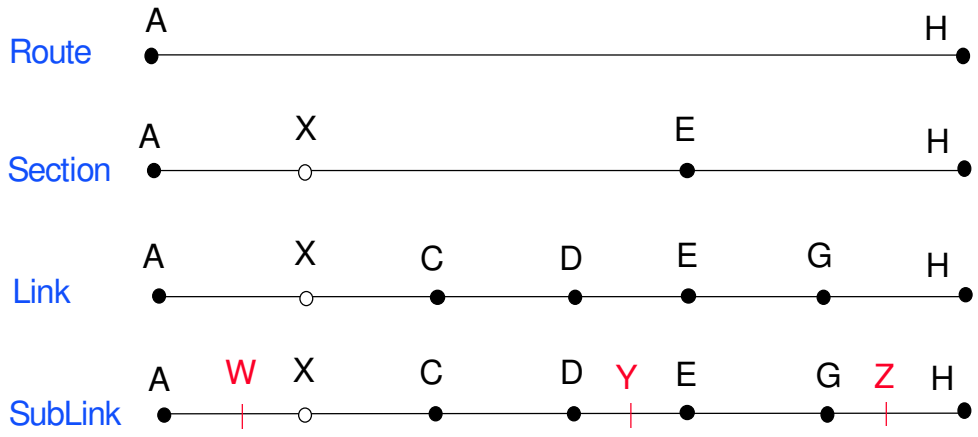


Figure 10. Sections defined using a point other than a stopPoint.

Precise definition of these routeComponents is essential. In addition, the rules for layering them must be clearly specified. Perfect tree structures are recommended for this layering. In other words, every higher level link must only be expressed in terms of the links at the next level down in the hierarchy. In a layered structure made up of sections, links and sublinks, we have to enforce these two rules; a sublink can not stretch across the ends of a link, and a link can not stretch across the ends of a section.

If we want to work with sections like XE in figure 9, then we have to accept that some links (role: interstopLink), can not be components of this section (e.g., AC and BC). Both of these links stretch across the end of section XE. We would have to work with more than one tree structure to layer these interstopLinks.

If we use the definition of section inherent in Figure 9, and consider other points within links (e.g., W in AX, Y in DE, and Z in GH), then we are in effect introducing a 3-tier layering made up of sections, links and sublinks (see figure 11).



Figire 11. A layered structure for a route.

In our first transport object model (TOM) for the scheduling domain [5,9], we defined any point used for route definition as a routeDefinitionPoint. This has become one of the roles in our Point pattern. All the other points in figure 11 could play the role of routeDefinitionPoints as well as their own more specific roles.

We wanted to create a perfect tree structure within which the links traversed by a journey could always be expressed in terms of the links and sublinks used for defining routes. To meet this requirement, some routeDefinitionPoints also play a journeyDefinitionPoint role. All journeyDefinitionPoints must also play the routeDefinitionPoint role. Nearly all the journeyDefinitionPoints play the role of stopPoints, but this is not mandatory.

So how can we cope with all the complexity introduced by multiple concurrent roles for the points making up a transport network?

Solution

– Structure

The Route pattern is presented in Figure 12. A route is composed of routeComponents (sections, links or sublinks), sectionComponents (links or sublinks) and linkComponents (sublinks). At each level, an abstract component class provides an interface to a leaf or

composite class. The composite classes are ordered aggregates of their parent component class.

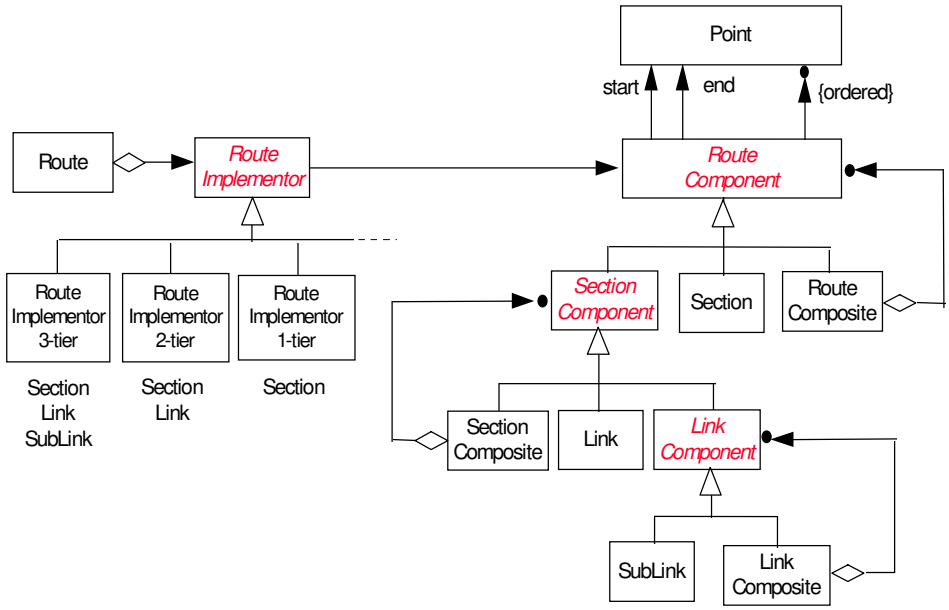


Figure 12. The Route pattern

A component in a particular layer does not have to be instantiated. For example, a route can be expressed as an ordered sequence of the following components.

1) sublinks only
2) sublinks and links
3) sublinks and sections
4) links only
5) links and sections
6) sections only
7) sublinks, links and section

At every level, a component can be expressed as an ordered sequence of points rather than as an ordered sequence of lower level components.

The rules and algorithms for implementing a particular route representation are contained in the concrete subclasses of the abstract class RouteImplementor. The implementation can be varied simply by delegating to a different routeImplementor. This can even be done at run time.

– Example

To illustrate the Route pattern in Figure 12, we use the route structure in Figure 11 as an example. The three layers of route AH (A-W-X-C-D-Y-E-G-Z-H) are as follows.

1) routeComponent which is an ordered sequence of sections (e.g., AH –> AX, XE and EH) or sectionComponents.
2) sectionComponent which is an ordered sequence of links (e.g., XE –> XC, CD and DE) or linkComponents.
3) linkComponent which is an ordered sequence of sublinks (e.g., DE –> DY and YE).

The option to express each lower level component (i.e., sections, links and sublinks) in terms of points is inherited from the parent class RouteComponent.

A typical route structure involving sublinks (e.g., route AH) is shown in Figure 13. RouteImplementor3-tier would be responsible for constructing this route structure.

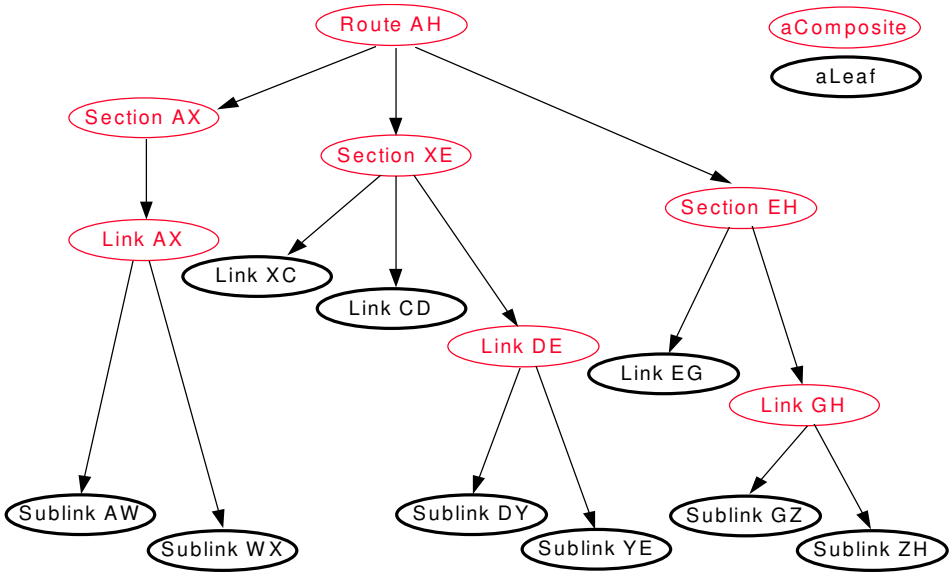


Figure 13. Typical 3-tier route structure.

Other route structures at a higher level of abstraction (i.e., not down to sublink level) would be the responsibility of other concrete subclasses of RouteImplementor. Route would hold a reference to the appropriate concrete routeImplementor object which could be changed even at run time. For example, routeImplementor2-tier would represent AH as shown in Figure 14.

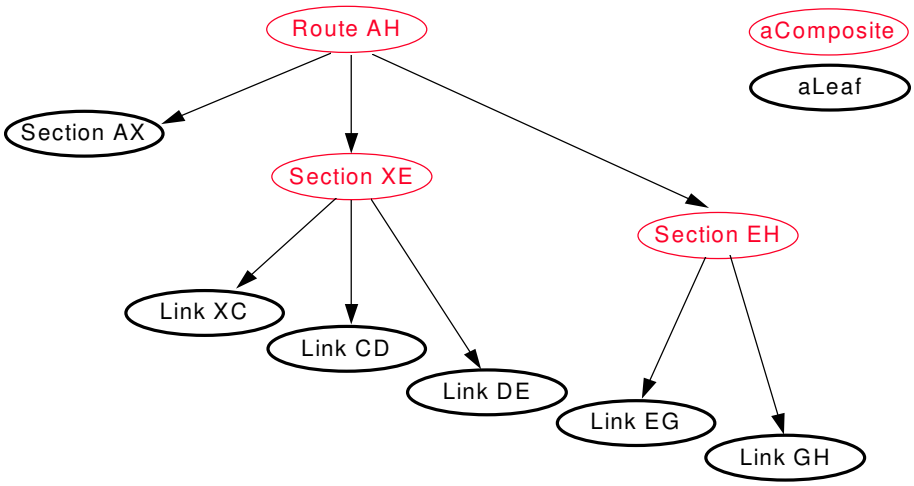


Figure 14. Typical 2-tier route structure.

– Participants

Components (RouteComponent, SectionComponent, LinkComponent)

RouteComponent = RouteComposite, Section or SectionComponent.

– Declares the interface for routeComposite, section and sectionComponent objects.

- Implements default behaviour for the interface common to these routeComponent objects as appropriate.
- Declares an interface for managing the ordered sequence of these routeComponent objects.

SectionComponent = SectionComposite, Link or LinkComponent.

- Declares the interface for sectionComposite, link and linkComponent objects.
- Implements default behaviour for the interface common to these sectionComponent objects as appropriate.
- Declares an interface for managing the ordered sequence of these sectionComponent objects.

LinkComponent = LinkComposite or Sublink

- Declares the interface for linkComposite and sublink objects.
- Implements default behaviour for the interface common to these linkComponent objects as appropriate.
- Declares an interface for managing the ordered sequence of these linkComponents objects.

- **Leaf (Section, Link or Sublink)**
  - Defines behaviour for these leaf objects.

- **Composite (RouteComposite, SectionComposite, LinkComposite)**

  RouteComposite
  - Defines behaviour for routes with one or more routeComponent objects.
  - Stores an ordered sequence of routeComponent objects.
  - Implements operations for managing this ordered sequence of routeComponent objects.

  SectionComposite
  - Defines behaviour for sections with one or more linkComponent objects.
  - Stores ordered sequence of sectionComponent objects.
  - Implements operations for managing this ordered sequence of sectionComponent objects.

  LinkComposite
  - Defines behaviour for links with one or more sublinks.
  - Stores ordered sequence of sublinks.
  - Implements operations for managing this ordered sequence of sublinks.

- **Client (RouteImplementor)**
  - Manipulates sections, links and sublinks through a routeComponent interface on behalf of route.
  - Each concrete routeImplementor is responsible for the structuring of the routeComponents as required by route.

Pattern Interactions

– Point

Point is used to build RouteComponent (Figure 15). Every routeComponent starts and ends at a point and can be expressed as an ordered sequence of two or more points. For a description of the Point pattern, see section 3.

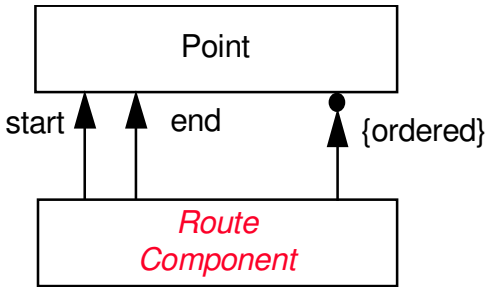


Figure 15. Pattern interaction: RouteComponent builds upon Point

– Composite

The underlying pattern of Route is Composite [6]. Composite is a very powerful and generic construct for Route, which has two particularly elegant features.

- It can represent layered tree structures consistently.

  Most of the many variations in representing routes that we have encountered are covered by Composite. For example, if we have a consistent 3-tier structure and then have to extend the system to cater for a rogue sublink (i.e., one that is not a subdivision of one of the existing links), then this can be introduced as shown in Figure 16 without disrupting the existing structure. Identifying such inconsistencies may prompt a redefinition of requirements. However, we often have to live with these inconsistencies because of the need to provide continuing support for legacy systems and find other means of integrating the applications concerned. Composite helps us to deal with these mismatches.

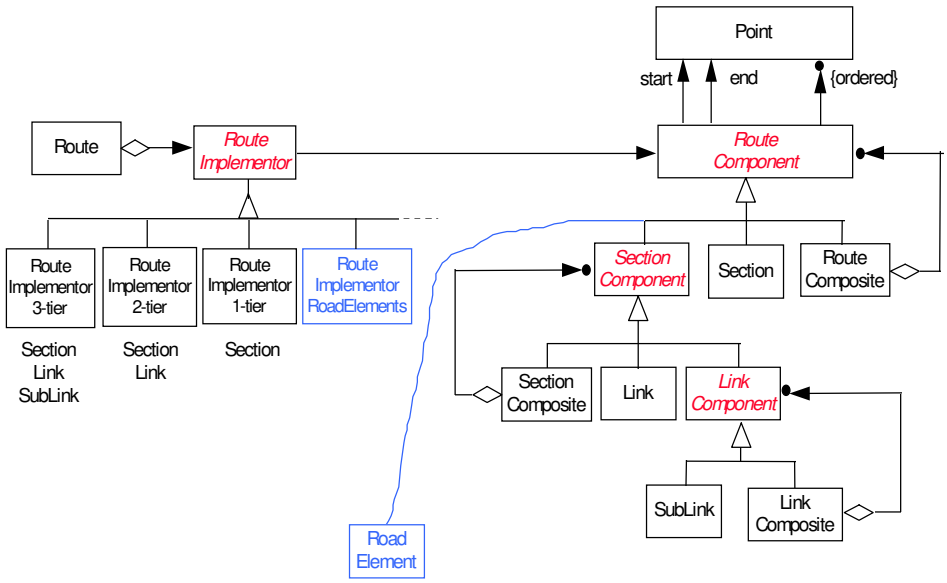


Figure 16. Inserting a component that breaks the layering rules.

- It can express the cascade of roles in a clear order.

  The role pattern is in our opinion particularly suitable for modelling independent objects like point. Objects that are defined upon other objects like route, section and

link imply a layered tree structure and can not simply be modelled as roles. Composite has been used to cope with the cascade of roles in Route.

In effect, Composite is so generic that we have also used it to structure our Duty pattern [10] and a generic layered tree pattern which we are calling Cascade[11]. The Cascade pattern considers this hierarchical layering of Composite patterns stripped of its transport domain specific content.

Applicability

Use the Route pattern to order and layer the components that define a path through a road network. Sections, links and sublinks are in a part-whole relationship with route and all rely for their definition on the roles of the points that they pass through.

Clients of a route can ignore the difference between compositions of objects and individual objects. Clients will treat all objects used for route definition uniformly.

Forces

All routeComponents – even the route itself – are in effect basic links playing particular roles determined by the roles of their begin and end points. The sections, links and sublinks described in this paper differ only in that they have been assigned to a particular layer in a perfectly layered tree structure. Many other basic link roles can be defined which do not fit neatly into this tree structure.

An alternative to the explicit layering of links would be to work only with basic links. The rules for implementing many different tree structures made up of basic links would also be encapsulated within appropriate concrete routeImplementors. This solution is a special case of the Cascade pattern (i.e., the one tier case). This solution may be preferred when there is no real world need to describe basic links in terms of their position within one particular tree structure and/or where the behaviour and rules to be assigned to a RouteComponent in each layer do not differ significantly.

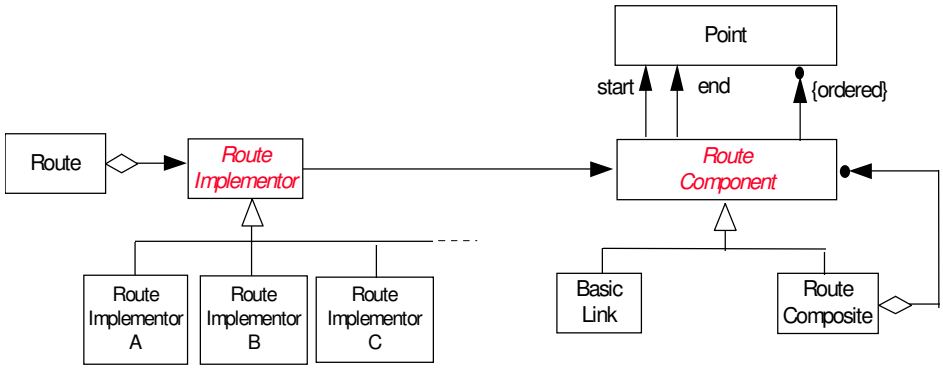


Figure 17. Using basic links

Some of the responsibility for implementing and maintaining the internal structure of the cascade of composites can be assigned to concrete subclasses of the RouteComposite, SectionComposite and LinkComposite classes rather than RouteImplementor. For example, in our Duty pattern [10], we have applied the Gang-of-Four composition-compositor solution (pp. 41-42, [6]) to DutyComposite, DutyPartComposite, StretchComposite and SpellComposite which are also cascaded like Route. The algorithms and rules for composing duties, dutyParts, stretches and spells are complex,

and vary with the type of DutyComponent being constructed, which makes them particularly suitable for this treatment.

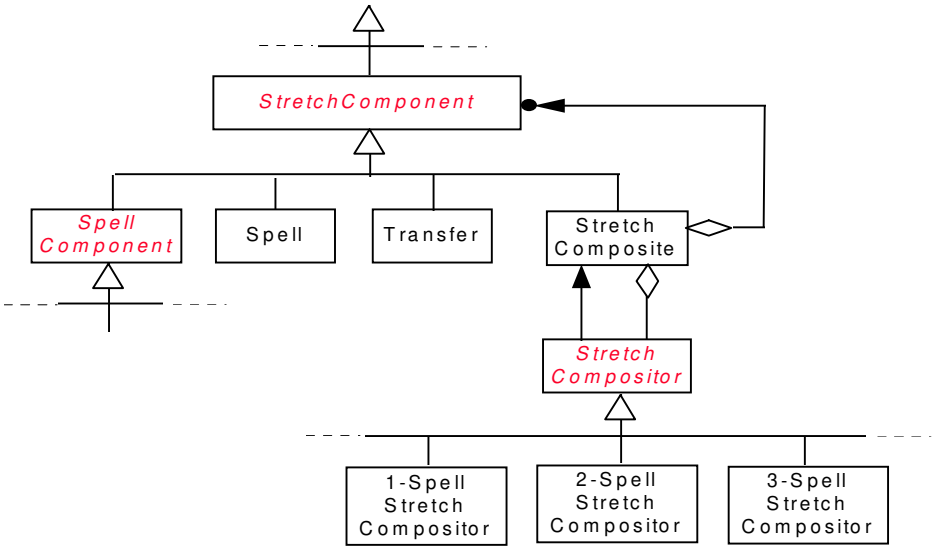


Figure 18. Using Composite-Compositor.

Should points themselves be layered? For example, a point playing the role of a station may be viewed as several points playing the role of a platform. Cascade can also be used for layering or grouping points. The implications of two parallel and interacting Cascade patterns, one for links and one for such point groups, will be considered in a separate paper [11].

Design Rationale

The Route pattern caters for the requirements of those transport operators and designers who use link or edge based algorithms as well as point or node based algorithms; the representation is not hard coded into a route object; responsibility for managing the structure is delegated to concrete route implementors and/or compositor objects. Different types of routeComponent, defined by the roles of the points making up these components, can be layered into one or more perfect tree structures. The Route pattern helps to structure the complex movements of vehicles, drivers and passengers within a transport network.

Similar Patterns and Applications

The Route pattern is a domain specific application of a Cascade [11] of Composite [6] patterns for layering routeComponents playing multiple concurrent Roles. The same cascading of roles is also suitable for representing other transport domain objects (e.g., journeys and duties). Other patterns interact with this cascade of composites. For example, the Gang-of-Four Composite-Compositor solution has been applied to each of the layers of our Duty pattern. Riehle [8] proposes an Observer pattern within his larger Bureaucracy pattern to perform a similar function. The application of these patterns help to bring order and understanding to complex dynamic systems such as the operation of a transport system.

5. Acknowledgments

We wish to thank our EuroPLoP96 and PLoP96 "Shepherd", Bob Hanmer, for his encouragement in writing up our transport domain patterns; for his continuing enthusiasm and providing of many invaluable comments on our pattern papers. We would also like to thank colleagues in various Science and Engineering Research Council (UK) and European Union (EU) funded projects (especially within Cassiope and Titan), with whom we have discussed the problems of modelling public transport operations.

References

[1] Buschmann F. and R. Meunier (1995). A System of Patterns. In *Pattern Languages of Program Design.* J.O. Coplien and D.C. Schmidt (Ed.), Addison-Wesley.

[2] Coad, P. (1992). Object-oriented patterns. *Communications of the ACM, 35(9): 152-159.*

[3] Coplien J.O. (1996). *Software Patterns: A White Paper.* SIGS Publications, New York.

[4] Foster, E and L. Zhao (1996). TOPs: transport object patterns. In the Proceedings of EuroPLoP96.

[5] Foster, E, A.J.Barnard and L.Zhao (1994). PoeT: Object-Engineering in Public Transport. An Object Model for Timetabling and Vehicle Scheduling. Report, University of Leeds, School of Computer Studies.

[6] Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley.

[7] Martin, J and J.J.Odell (1992). *Object-Oriented Analysis & Design*. Prentice-Hall.

[8] Riehle, D. (1996). Bureaucracy – A Composite Pattern. To appear in the Proceedings of EuroPLoP96, Addison-Wesley.

[9] Zhao, L and E.Foster (1994). ROO: rules and object-orientation. In C.Mingins and B.Meyer (Eds.), *Technology of Object-Oriented Language and Systems:* TOOLS 15, pp. 31-44. Prentice-Hall.

[10] Zhao, L. and E. Foster (1996). PLOTS: Duty pattern. In preparation.

[11] Zhao, L and E. Foster (1996). Cascade: a generic tree structure. In preparation.