Composing Analysis Patterns to Build Complex Models: Flight Reservation

Zhen Jiang
Computer Science Department
25 University Ave.
West Chester University
West Chester, PA 19383
zjiang@wcupa.edu

Eduardo B. Fernandez
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
ed@cse.fau.edu

ABSTRACT

In previous work we developed the concept of Semantic Analysis Patterns (SAPs). SAPs are mini-applications realizing a few use cases selected so as to make them as generic as possible. One of the objectives of this approach is to make SAPs convenient for inexperienced modelers to build complex object-oriented concept models. In this paper we show the use of SAPs to build complex analysis patterns from the combination of simpler patterns. We also claim that this approach provides models that are also extensible and reusable. We present here a case study (a flight reservation system) that illustrates how SAPs can be composed to build complex models in a convenient way. In developing our set of patterns we created one pattern and specialized some existing patterns in the context of flight reservations. These patterns provide a common structure that has to be present in any flight reservation system, but they can also be of independent value.

Keywords

Analysis patterns, Composite patterns, Flight reservation systems, Flight routing, Object-oriented analysis and design

1. INTRODUCTION

When dealing with the specification, design, or implementation of a number of similar applications [1], common parts can be found. These parts can be specified as patterns that are independent from a particular specification, implementation details, or implementation languages. Sometimes such modules are not so simple: a general module that can satisfy different purposes is not trivial. Moreover, the more complicated modules often vary slightly from application to application. As the number of applications increases, their management becomes increasingly difficult and unwieldy. In previous work [6], we developed the concept of *Semantic Analysis Pattern* (SAP), which emphasizes functional aspects of the application model as opposed to improving flexibility as in most design and analysis patterns. The main purpose of this type of pattern is to serve as a starting point when translating requirements into a conceptual model. A SAP represents a minimum application (a set of basic use cases) so that it can be applied to a variety of situations and it can be combined with other related patterns to describe more complex applications. A SAP is built by combining the classes and is required to realize two or three basic use cases, which are determined in the usual object-oriented way [14]. SAPs use the POSA template [4].

A possible way to build complex analysis models using SAPs was described in [6]. However, regardless of the methodology used to build the models, having a complex pattern as a building block makes this work easier. We show here how to build a complex analysis pattern [15] using a travel application as an example. In this example, the composite pattern is built from some existing patterns. The application requirements are described in a common context for all the patterns. Our approach is intended for analysts, architects, and developers, i.e. all those involved in building complex systems.

Section 2 describes the requirements for this system while Section 3 presents the atomic patterns that satisfy these requirements. To describe each pattern we will loosely follow the templates of [4]. Section 4 shows how we compose these patterns to develop a flight reservation system. The effectiveness and flexibility of the complete application, in effect, a new SAP, is shown by domain analysis using some examples. We end with our conclusions in Section 5.

2. REQUIREMENTS AS COMMON CONTEXT

A flight reservation system is a commonly used system. Typically a customer places an order for seats in a combination of connected flights from an origin to a destination airport. The customer and the system need to check the feasibility of flight connections and their schedule [2]. Because of the complexity of this application, it is difficult to build it as one unit. We show here that by composing some simple patterns, we can build this application as a SAP in a systematic way. The atomic or component patterns correspond to specific functions of the system and can be either new patterns or existing patterns, perhaps specialized for the application. Because of the way the whole application is built, the resulting model is also flexible and reusable [12]. The final composite system could also be the basis for a framework.

It is possible to visualize the structure of a large system as a set of component units, where each one is based on a different set of functional aspects. The list of requirements for the reservation system is given below. These requirements can be expressed as a set of use cases, which we do not describe in detail for conciseness. The most important requirements for this system are:

- A *flight* is defined by a number and a date, and it defines a route from an origin airport to a destination airport. A plane is assigned to a flight and it contains a set of numbered seats.
- Customers make reservations for specific seats in specific flights.
- A *route* is the path followed by a flight from its origin airport to its destination airport. There may be several flights that share the same origin and destination. A route includes one or more spans.
- A *span* is a part of a route to get from a start airport to a termination airport as part of a specific flight. The start (or termination) airport is called the origin (or destination) airport of this span.
- For each flight there are several *connecting flights* (i.e., different flights that leave from an intermediate stop closely after its arrival).
- A *ticket* includes a one-ticket route (one-way ticket) or a two-ticket route (round-trip ticket). Using a round-trip ticket, a passenger can go to an airport and come back using the same route. If the passenger returns using a different route, he/she needs a set of one-way tickets. Such a round-trip is special in that its source and destination are the same. Stops are not indicated in the ticket unless the flight number changes, which indicates a plane change.
- An *airlink* is any direct flight between two airports. All airlinks are spans of some flight. A basic airlink has no stops. A route is a set of connected basic airlinks connecting all the airports through which it passes.
- For the convenience of customers we may keep information on relevant facilities; for origins we keep aspects such as parking, for destinations we list hotels near the airport, for intermediate stops we provide lists of hotels close to the airport, restaurants, etc. This information may also include details of the cities nearby.

Figure 1 clarifies these concepts. A passenger intends to go from airport A to airport B. He has four routes available for this trip:

- Route 1: Using flights 12 and 13.
- Route 2: Using flights 12 and 14.
- Route 3: Using flights 15 and 13.
- Route 4: Using flights 15 and 14.

He could use any of these routes in a one-way ticket from A to B. If he wants a round-trip ticket he has more combinations, any of the four routes above to get to Airport B can be combined with routes (16,18) or (17,18) to return to

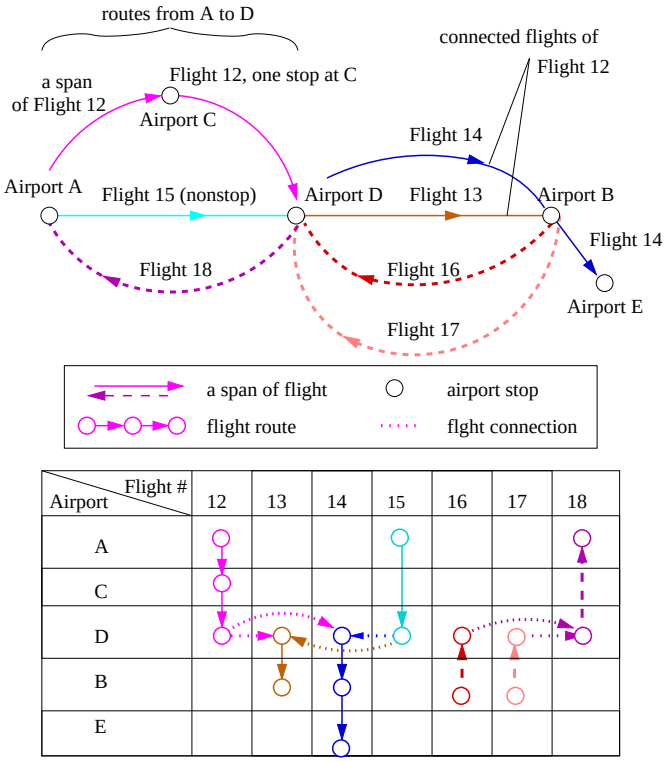

Figure 1: Definitions for the requirements.

A. Flight 12 (from A to D) has two spans because it stops at Airport C, while Flight 15 has only one span (a nonstop flight). The set of Flights 12, 14 is an airlink from A to B. Flights 13 and 14 are basic airlinks connecting Airports D and B. However, Flight 14 has another span to get to Airport E, while Flight 13 has only one span.

From these requirements, we derive some atomic patterns that describe specific aspects. The Connection pattern appears to be new, while Flight Route, Seat Collection, Airport Role, Travel Ticket, and Seat Assignment are instantiations of known patterns in a specific context. For each functional aspect, we use a corresponding pattern to implement it and make some adjustments to satisfy the context constraints. Figure 2 describes these patterns as a pattern language for travel. A **Travel Ticket** describes an air trip as a series of tickets that correspond to connecting flights (**Connection** pattern). For each specific flight, a seat assignment must be obtained (**Seat Assignment** pattern). Seat availability is determined by using the **Seat Collection** pattern (a flight implies a collection of seats). The connecting flights may use information from the airports through the **Airport Role** pattern. The common contexts for all these patterns are passenger transportation systems, including airlines, railways, water navigation, or bus systems, although for concreteness we use the notation of air transportation. Figure 2 can be seen as a metamodel that describes how these patterns should be connected to define a complete model.

Patterns can be expressed at different levels of abstraction. For example, a pattern for general customer orders can be specialized to order tickets. The specialized version can still be considered a pattern (and not a specific model) because

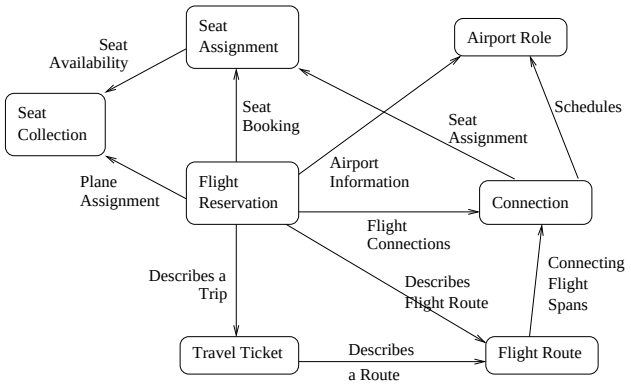

Figure 2: A pattern language for flight reservations.

there are many situations that require ordering a series of tickets, airline reservations being one of the most important and probably the most complex. In particular, the Travel Ticket pattern is a specialization of the Order pattern [5, 7], where each flight corresponds to an order line, and the Seat Assignment pattern is a special case of the Assignment pattern [10]. Section 3 presents all the component patterns.

3. COMPONENT PATTERNS

We present here all the patterns described in Section 2. These are the building blocks for the complex pattern.

3.1 Travel Ticket

Intent
A series of tickets for a certain type of trip (one-way / round-trip) is booked for a passenger. Each ticket describes a series of connecting flights from an origin to a destination.
Problem
How should we describe a request for a series of tickets?
Forces

- Going from an origin to a destination often implies a series of tickets, not just a single ticket.
- The information to model a ticket must include origin, destination, flight information, and seat information.
- A passenger is responsible for one or more tickets.

Solution
A specialized version of the Order pattern [7]) satisfies the forces. The class model of Figure 3 shows the required information, including classes to describe the series of tickets (**TicketSeries**), the passenger who is responsible for that order (Passenger), and two sets of tickets (**TicketRoute** and **TicketUnit**). TicketRoute is used for arranging the schedule of Flight and TicketUnit is used for price-checking or possibly even for later check-in. Each TicketRoute object consists of several TicketUnit objects. The corresponding sequence diagram is given in Figure 4, which shows how to place an order for a series of tickets. The two aggregations correspond to two views of the tickets, a set of paper/electronic tickets that describe the costs and are used for check in, and a set of specific routes describing an itinerary.
Consequences

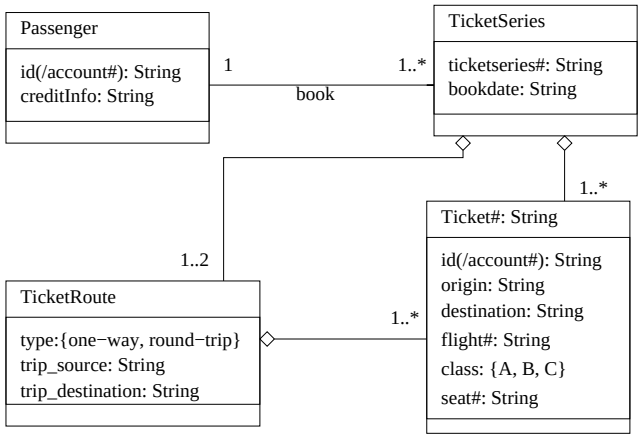

Figure 3: Class diagram for Specialized Order (Ticket) pattern.

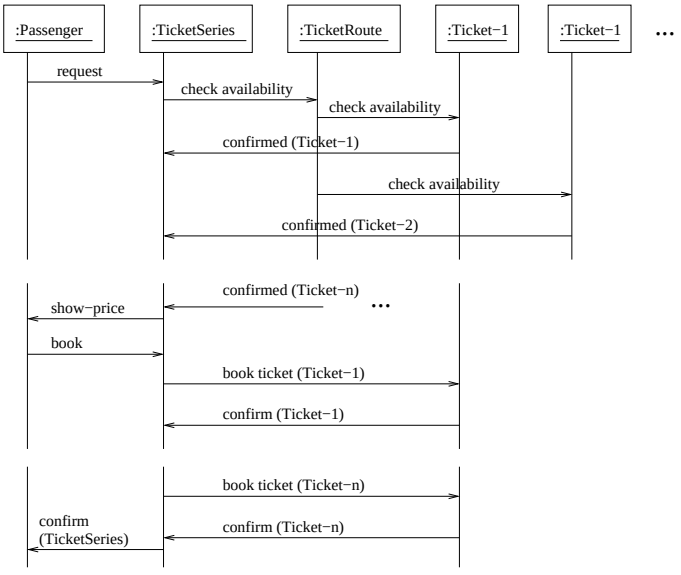

Figure 4: Sequence diagram to order a set of tickets.

- This pattern describes only the ordered tickets, it must be complemented with other aspects e.g., seat and flight information (although this information may appear in them too).
- TicketSeries in this pattern may be any other product with multiple units.
- The customer who places an order may be a person or a corporation. Here we only focus on the availability for the passenger who will get the ticket(s) and use it (them). We could separate this customer role from our purpose of ticket purchase [13].
- The requested tickets refer to a series of products. Their relationship and detail processing are discussed later in other patterns.
- Delivery, payment, and identification details are not included.

Known uses
In several situations we need to issue a set of related tickets, e.g. air travel (a ticket for each flight), railway tickets (a ticket for each train), concerts (subscription to a series of concerts).
Related patterns
This is a specialization of the Order pattern [7].

3.2 Seat Assignment

Intent
A TicketUnit is assigned to a Seat, Flight, and Span.
Problem
How should we indicate the assignment of a seat to a span of a flight and correlate it to a ticket unit for as given passenger?
Forces

- For each right to go from one place to another in a particular flight we need to have a description. A TicketUnit is a representation of that right.
- A TicketUnit defines a seat reservation for a flight corresponding to a particular span. The seat may not be defined when the ticket is issued.
- To produce the TicketUnit the availability of a seat and the feasibility of a connecting span and its connecting flight must be confirmed.

Solution
This is a special case in a general assignment pattern, which can be used to assign resources [10]. Flight, span, and seat correspond to a ternary association and the ticket unit that describes the assignment is an association class. Figure 5 shows classes **Seat**, **Span**, and **Flight** and their assignment to the **TicketUnit**. Ticket units are collected in the **TicketSeries**, which collects all the tickets for a trip.
Consequences

- TicketUnit can provide the right to go in a specific span of a flight.
- Including a flight seat combination in a ticket indicates that this passenger has a reserved seat on this flight.

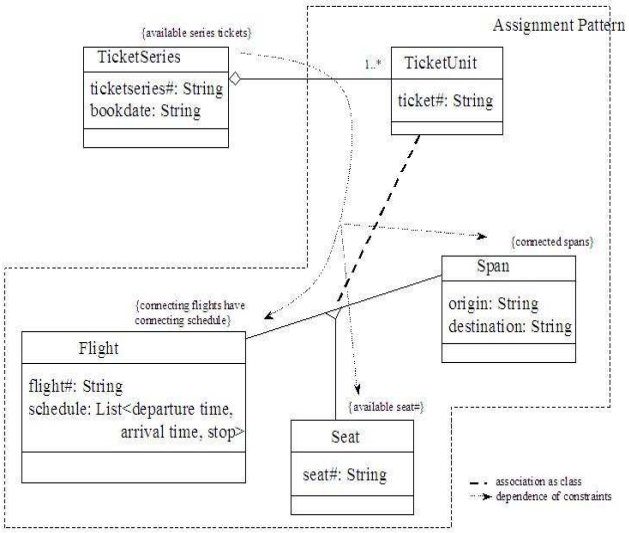

Figure 5: Class diagram for the Seat Assignment pattern.

- Constraints on TicketSeries help to define the connecting ticketUnit based on available seats, connecting flights, and connecting spans.

Known uses
Assignments of seats in flights, in a theater, in a classroom.
Related patterns
This is a special case in a general resource assignment pattern [10], which can be used to assign, for example, faculty to sections in specific courses.

3.3 Collection of Seats

Intent
Keep information about a collection or aggregation of seats in a plane, vehicle, or theater.
Context
Any physical domain where seats must be kept together as a group.
Problem
How should we describe a collection of seats that must be assigned individually?
Forces

- An entity has a certain number of similar units. For example, each seat is a part of a plane.
- There is a *whole* class and a *part* class. For example, a plane has a whole/part (WP) relationship with the seats it contains.

Solution
Based on that WP relationship, one can check or modify the availability of units or other information about the components. The class model for this pattern is shown in Figure 6.
Consequences

- This pattern may be used to keep track of the seats of any vehicle or building that has seats.

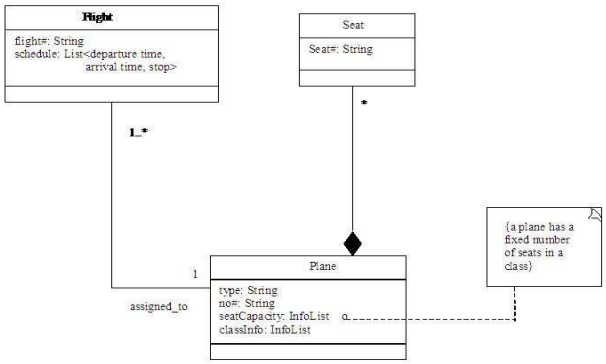

Figure 6: Class diagram for the Collection pattern (Seat and Plane).

- To check if a seat in a requested class is available, the seat (*part* object) should know the capacity and class information of a plane (its *whole* object). Alternately, a plane should know if all seats are booked from this WP relationship.

- Constraints defined on Seat help to confirm the available seats of a plane which is assigned to a flight.

Known uses
Airplanes, trains, theaters, stadiums, have collections of seats, usually numbered.
Related patterns
This is a special case in the Whole-Part pattern of [4]. The Whole-Part pattern describes the aggregation of components that together define a semantic unit.

3.4 Self-Connection pattern

Intent
Describe relationships between objects in the same class.
Problem
Objects in a set may have relationships to some others of the same set. An airport is connected with another by an airlink. A span is connected to another if the destination of the preceding span is also the origin of the succeeding one. A span is also connected to another if the destination of the preceding span is connected with the origin of the succeeding one by other means of transportation. If there is another kind of connection between these two airports we use other-link (see Figure 7). A flight is connected to another if and only if airports between two connected spans have a connecting schedule such that the arrival time of a flight is before the departure time of the other flight. We need a convenient way to describe these connections.
Forces

- An object may be connected to another object of the same type by a semantic relationship.

- The two ends of the relationship have different meanings, e.g., origin and destination of a flight.

Solution
The Connection Pattern describes self-associations in a class

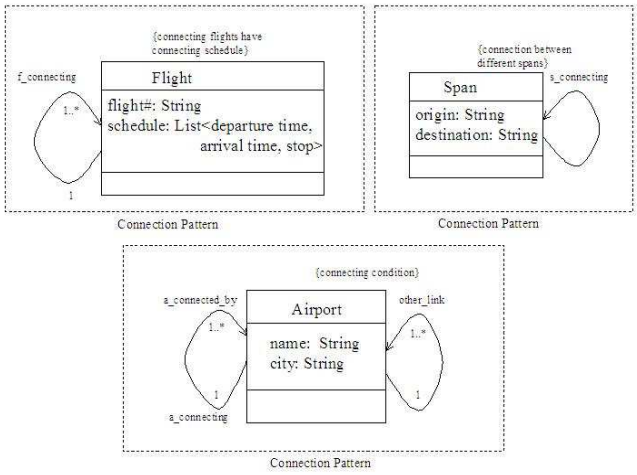

Figure 7: Class diagram for three instances of the Connection pattern.

and satisfies these forces. Figure 7 shows three instances of this pattern for connecting flights, connecting spans, and connecting airports.
Consequences

- A connecting association is used to describe the connection between two related objects of the same type.

- Constraints on a class help to define more precisely the connection between related objects

- There can be association classes defined on the connecting association, describing the attributes of such a connecting association.

- Role names may be needed to clarify the meaning of the ends of the relationship; e.g., origin.

Known uses
A manager is in charge of several employees, a flight has several connecting flights, people are related to several other people in their families.
Related patterns
This pattern is an important special case of the Assignment pattern [10].

3.5 Flight Route pattern

Intent
A flight route represents a collection of connecting airlinks that can be used as spans for travel from start to termination using one flight.
Problem
How should we describe a choice among a set of possible routes?
Forces

- Each Flight object defines a route from an origin airport to a destination airport. Between the origin and the destination, there may be several intermediate stops. An airlink which links an airport (called preceding airport) to another (called succeeding airport) without intermediate stops is called a *basic airlink*. In a route,

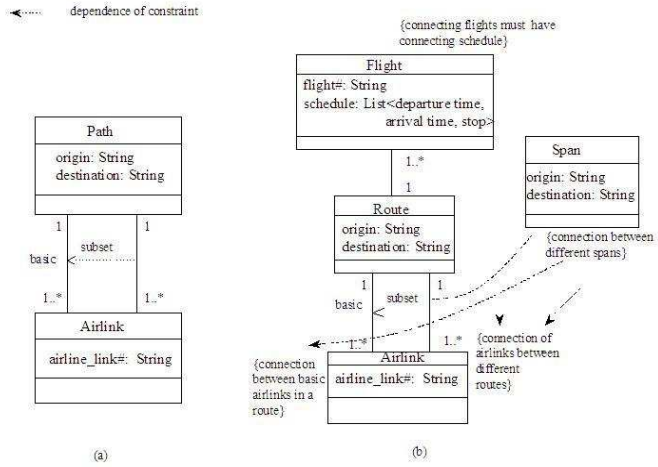

Figure 8: Class diagram for Flight Route pattern and its application in the reservation system.

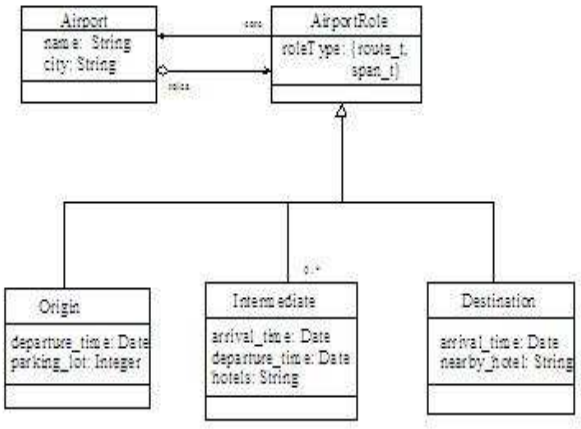

Figure 9: Class diagram for Airport Role pattern.

two basic airlinks are called 'connected' if and only if the first airlink ends at the preceding airport of the second airlink. Two basic airlinks in different routes are called connected when

1. these two airlinks are connected at one airport, or
2. the succeeding airport (i.e., the destination) of the first airlink is connected with the preceding airport (i.e., the source) of the second airlink by an other-link (other means of transportation).

A route is a set of connected basic airlinks connecting all the airports through which it passes.

- All basic connection airlinks are available for a passenger to choose as part of a span in a path from source to destination of his trip.

Solution
A specialization of the Path Pattern [17] satisfies the forces. The class model for that pattern is shown in Figure 8 (a) and its application in our reservation system is shown in Figure 8 (b). Figure 8 (b) shows a flight including one route, which in turn includes a subset of basic airlinks. This subset is described in association class Span.
Consequences

- A span is a part of route of a flight, and the choice of a span is based on the available connected airlinks in a flight route. For the convenience of customers, we should list the airlinks of all the airlines. The customer's request would be satisfied if there is any available connecting span, even when it belongs to a different airline.

- Based on the pattern, connected basic airlinks in a route of a flight provide a set of available spans. A passenger may select a subset of airlinks in a route to form a span from its origin to destination. The trip can be extended by other connecting spans from the destination of the preceding span. With all the connected spans in different flight routes, a passenger may fulfill a series of trips from source to destination.

- Constraints defined on connecting flights and connecting spans require that class Airlink provide *connections* for different airlinks. The constraints on connecting flights are based on the *connection* between two airlinks in different routes. The constraints on connecting spans are based on the connection between different spans. The dependence of constraints is shown explicitly in Figure 8 (b).

Known uses
Routing of electric networks, transportation systems, water distribution systems.
Related patterns
[17] describes pipes to fill vats with juice.

3.6 Airport Role pattern

Intent
To support the descriptions for different airlinks, the airports are classified as Origin, Intermediate, and Destination. An airport usually plays several roles.
Problem
How should we model the role aspects of an object as separate role objects that are dynamically attached to and removed from that object (*core* object)?
Forces

- An airport may have different roles for routes and spans at the same time. In a route, an airport and its connected airlinks indicate the available connections for a flight. In a span, an airport and its connected airlinks indicate the selection of reservations by the passenger.

- An airport may change its roles dynamically.

- Relationships between an airport and its roles are independent from each other so that changes to a role do not affect airlinks that are not involved in that role.

Solution
A specialization of the Role Object pattern [3] can be used here. A core object usually plays several roles and the same

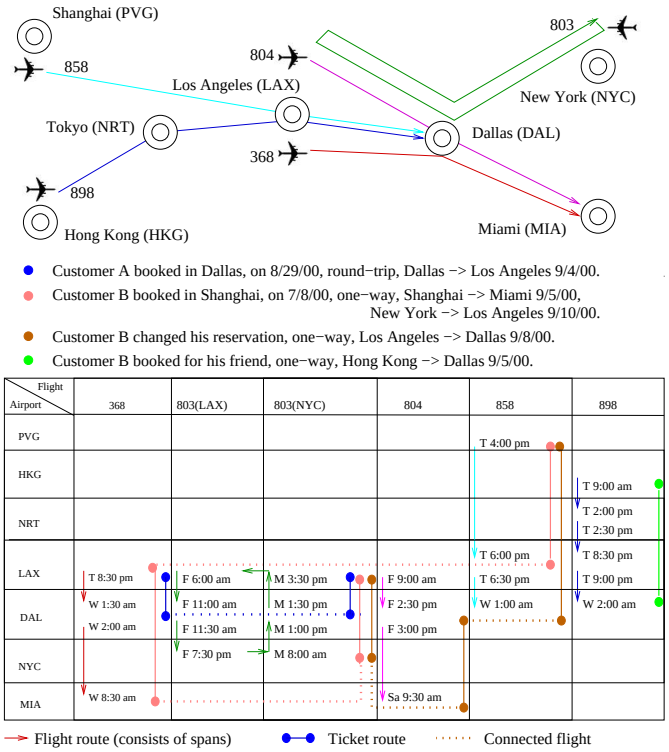

Figure 10: Some examples using the reservation system.

role is likely to be played by different core objects. When a core object is currently playing the requested role, it is in a role specific way and meets any request for that role. Figure 9 shows the class model for such a pattern specialized for a flight reservation system.

Known uses

A faculty member in a university may take the roles of instructor, thesis advisor, Principal Investigator in a research project.

Related patterns

This pattern is a special case in the Role Object pattern [3].

4. FLIGHT RESERVATION PATTERN

Intent

This pattern describes the placement of an order for a series of tickets.

Example

Figure 10 shows a specific example of the way of using such a system. Customer A wants to make a reservation in Dallas (DAL) on 8/29/00 (MM/DD/YY) for a business class round-trip to Los Angeles (LAX) next week. Customer B from Shanghai needs to attend a conference in Los Angeles on 9/12/00. Before getting there, he wants to travel to several cities in the United States. Starting from Shanghai (PVG), he plans to go first to Miami (MIA). Then, he will go along the east coast by car. It will take him several days until he arrives at New York. From New York (NYC), he continues the trip by plane and arrives at LAX before 9/12/00. He wants to make a reservation for such a trip. But a friend visiting in Hong Kong (HKG) asks customer B to make a reservation for him and his family (wife and

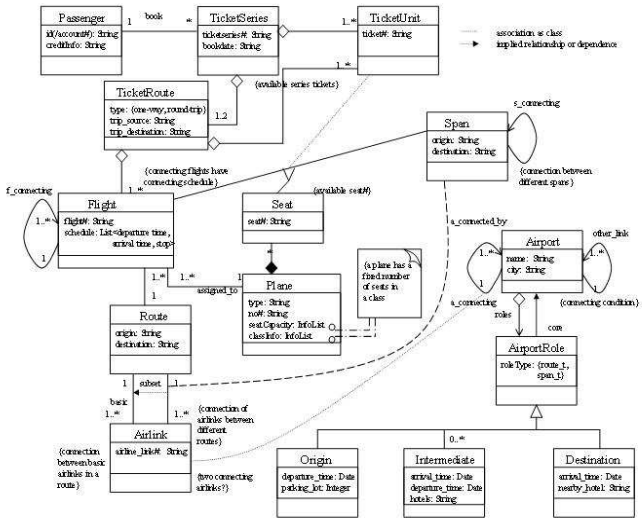

Figure 11: Class diagram for complete flight reservation system.

daughter) to Dallas (from HKG). Now B wants to change his reservation and make a reservation for the friend so that he can stay with his friend at Dallas for two days.

Context

Section 2 shows the context for this pattern.

Problem

Building a complicated itinerary for flight reservations is a difficult problem. Most systems build these itineraries using a travel specialist who may miss useful possibilities. We need an automatic way to compose available tickets.

Forces

- The requested tickets and the relationships between them must be captured in a precise way. Requests may be individual or group requests.
- A customer's reservations may change over time and it should be easy to make these changes.
- The pattern must describe a fundamental semantic connection. This means the pattern must be simple enough to apply to a variety of related situations.

Solution

Figure 11 combines all the patterns seen earlier and includes all their requirements.

Example resolved

Using such a system, customer A can select an available flight among all those passing through the span from DAL to LAX, i.e., $803 < M8:00am, M3:30am; F6:00am, F7:30pm >$, and select another to come back, i.e., $368 < T8:30pm, W8:30am >$. Although the route of flight 803 covers the span from LAX to DAL and he may select flight 803 for his ticket, he does not use such a flight because he does not want to wait until Friday. He removes the relationship of the ticket for the back trip and resets it to an earlier flight (flight 368). He will get a series of tickets with two sets of round-trip ticket units. The first one is for the trip from DAL to LAX and the second one is for the trip back to DAL. Each ticket unit in the set represents a span using a

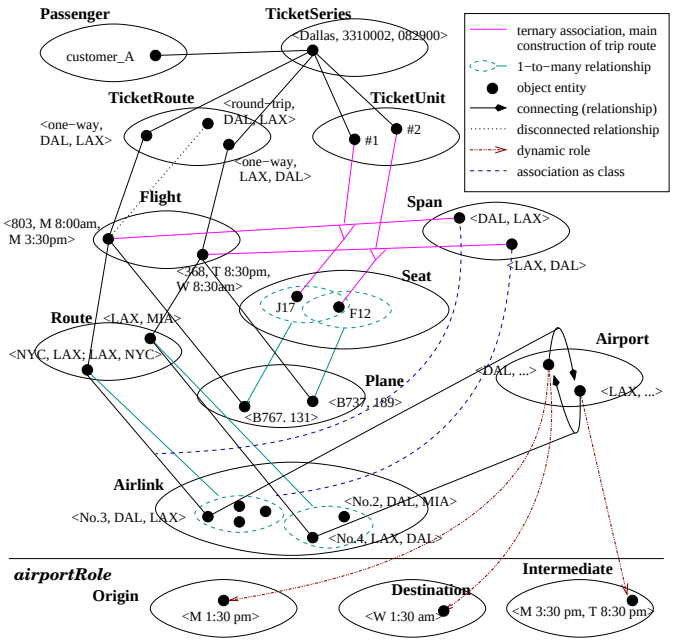

Figure 12: Domain analysis of a reservation system for customer A (round-trip).

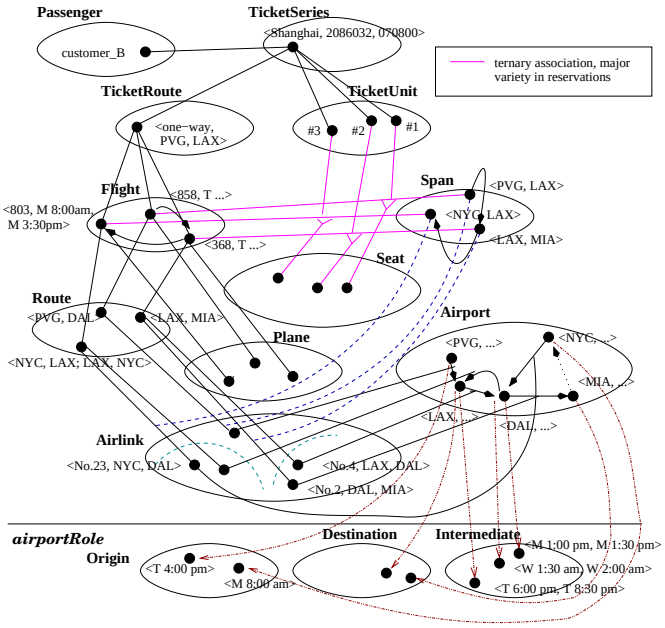

Figure 13: Domain analysis of a reservation from Shanghai to LAX for customer B.

part of a route of the flight. For example, customer A uses the part of route of flight 803 from DAL to LAX and that of flight 368 back to DAL. A trip in any set can be extended by a ticket with a connecting span. The extension will be discussed in the next example for customer B. Any airport in the trip may have different roles for route and span. For example, DAL is an intermediate stop for the route of flight 803 but it is also an origin airport for the span from DAL to LAX. To check if there is an available seat in the requested class, the detail capacity and class information of planes $< B767, 131 >$ and $< B737, 189 >$, which are assigned to flights 803 and 368, can be accessed by the collection of requested seats. If the requested seat is not available and there is no more available seat in the plane, he may use another flight. Finally, customer A will be satisfied by a seat in the plane.

As shown in Figure 12, $J17$ in $< B767, 131 >$ and $F12$ in $< B737, 189 >$ are seats he can book. Using the same system, customer B can select a series of connecting spans for the one-way trip from Shanghai to LAX based on all the basic airlinks supported by connecting flights. As shown in Figure 11, among all the basic airlinks, he selects span $< PVG, LAX >$ and its connecting span $< LAX, MIA >$, which are part of routes of flight 858 and its connecting flight 368 to go to MIA. Flight 858 is connected by flight 368 at LAX because the arrival of flight 858 at LAX is two hours earlier than the departure of flight 368 in the same day. The trip by flight 858 is extended by flight 368 from LAX to MIA. From MIA, the passenger will go to NY by car; that is, there is an other-link relationship between MIA and NYC. As the arrival time of NYC is earlier than the departure time of flight 803, the passenger may select the connecting span of flight 803 from NYC to LAX after a trip from Shanghai to New York. The connection condition and its satisfaction, which is requested by the customer and supported by the routes of flights, are used to check the feasibility of spans for the trip. The availability of seat(s) in the plane assigned to the route of the used span is checked for the reservation. If there is no available seat in the plane, the customer should select another set of connecting spans to reach the destination of his trip. After the check, customer B makes a reservation for the available series of tickets with three one-way ticket units (see Figure 13).

After a request from his friend in Hong Kong, customer B wants to change his reservation so that he can stay at Dallas with his friend for two days. He extends the first span $< PVG, LAX >$ of flight 858 to DAL. After that, he continues his trip from DAL to MIA by using a connecting span of flight 804. It is advantageous here to keep most of the feasible parts of the trip so that the reservation can be made easily and quickly in a very complex and dynamic situation. In Figure 14, the customer only changes the connecting spans at DAL. Note that the related connecting flights and the connecting spans based on the basic airlinks of the routes are already available in the system, providing an easy change (see Figure 14).

For the friend and his family, customer B selects flight 898 and makes a reservation for three seats in the same trip. As shown in Figure 15, there is a series of one-way tickets for the friend. Each ticket unit is assigned to flight 898 (from HKG to DAL), a seat on the plane, and the span based on the route of such a flight. Except for the seat, all the ticket units share the same information of this trip. Duplicate copies of the information for airports and flights are avoided in such a system. This simplifies the process of finding the feasible spans and available seats for everyone in the family. It also facilitates the management of the information for airports and flights.

Known uses

Orbitz provides possible routes between any two destinations, including flights of any airline. These can be con-

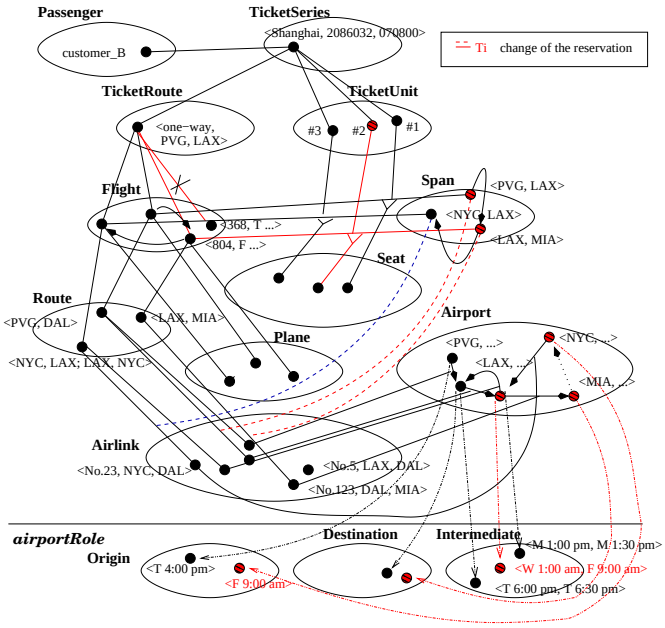

Figure 14: Domain analysis of a changed reservation for customer B.

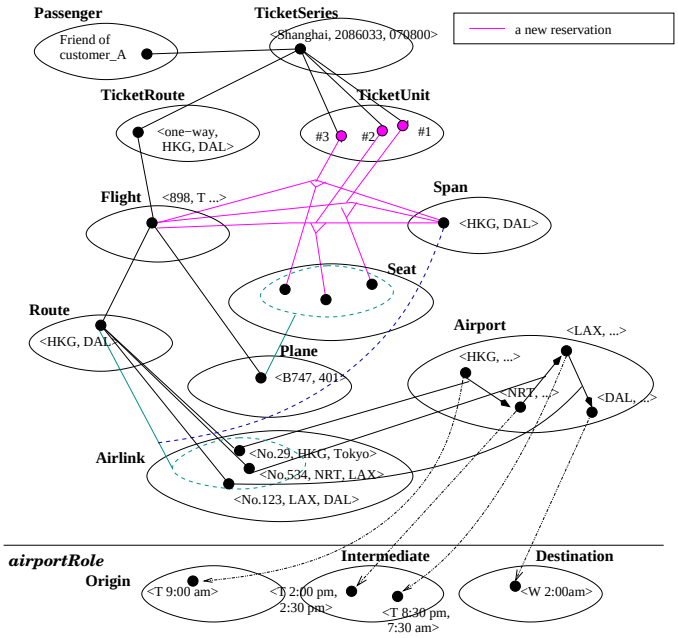

Figure 15: Domain analysis of reservation for the friend and his family.

verted into reservations and later into tickets. American Airlines provides similar functions, but includes only their own flights and those of their alliance partners.

It is a good example of a complex system that uses familiar, smaller patterns. Most airline web sites use similar models, although not necessarily object oriented.

Consequences

The model satisfies the forces in the following ways:

- The pattern describes the request and satisfaction of ticket(s) for different types of trips.
- The pattern can be used as a more abstract pattern; it can be applied, for example, to any reservation system for a series of products. The products may be different in different applications.
- Some of the component patterns could be replaced by a pattern with a different function. This would allow us to extend the model for other applications or with different functions. This and the previous consequence make this pattern reusable and extensible.
- The effect of other activities can be reflected through appropriate operations.
- It is easy to make changes in reservations or to add more functions for a ticket, e.g., descriptions of stops.

In order to make the pattern applicable to other cases, we have left out:

- Details of the items, such as operations for each service.
- Information about the airlines.
- Exceptions, e.g., unavailable tickets, delays, and flight cancelations.
- Alarms; for example, when a flight is sold out.
- Historical information.
- Billing and payment policies, e.g., order cancelation and refunding.
- Personal identification.

These aspects should be completed with additional patterns.

Related patterns This is a composite pattern using the six patterns described earlier.

5. CONCLUSIONS

Our approach involves the use of object-oriented methods and Semantic Analysis Patterns. By solving this type of problem using object-oriented methods we reap the general benefits of this approach, i.e., reusability, extensibility, and conceptual abstraction. It is recognized by researchers and practitioners that object-oriented methods are superior to procedural approaches for handling complex systems. This advantage extends to our approach. The general use of patterns is considered an advance in object-oriented methods because patterns distill the knowledge and experience of many developers and are highly reusable. Patterns also improve software quality because they have been scrutinized by many. Our Semantic Analysis Patterns have been shown to ease the task of building conceptual models by directly translating functional aspects of an application [6] and can also be used to define Secure SAPs, where the functionality is complemented with authorization and authentication aspects [11]. In this paper we have shown, through a case study, the ability of SAPs to compose patterns to build complex patterns or complex models in general. The component patterns realize the specifications of the system. While experiments with actual projects are necessary to prove the

practicality of this approach, we can say that this methodology is a better way to build complex systems than procedural programming or ad-hoc object-oriented methods. We have also shown our approach to be convenient to improve practical approaches such as XP [8], which is another proof of its possible value. There are other object-oriented approaches based on patterns, e.g., several approaches are discussed in [18], and we don't claim that our approach is better than any of these methods, because this would require a detailed and lengthy study. We do claim that our approach allows us to build complex models in a convenient and error-free way.

The specific problem that we used as a case study is of intrinsic interest because of its economic importance [16]. It is clear that software for flight reservations defined according to the requirements of Section 2 is used in many places. This software has been designed either by the procedural approach (most likely) or by object-oriented methods (in the most recent cases). However, our search did not yield any complete examples, only trivial portions in some textbooks. It is clear that software with this functionality is used in practice. We cannot then compare our solution to other solutions to this specific problem, but it was not our aim here to show a better solution to this problem; the example was selected because it was complex enough to show the value of our approach. Based on the discussion above, we would expect our solution to this specific problem to be easier to develop, more flexible, and more reusable than most solutions, at the same time without losing modeling precision. More importantly, the use of analysis patterns can help build good conceptual models to designers who have little experience.

6. ACKNOWLEDGMENTS

We thank all the participants of PLoP'09 workshop. We especially thank our shepherd, Sergio Soares, who provided valuable comments that considerably improved this paper.

7. REFERENCES

[1] M. Blaha and W. Premerlani. *Object-oriented Modeling and Design for Database Applications*. Prentice-Hall, 1998.

[2] C. Ball. An object oriented analysis of air traffic control.
MITRE Corp. August 1991. Document also available at http://www.mitrecaasd.org/library/tech_docs/pre1999/wp90w542/.

[3] D. Baumer, D. Riehle, W. Siberski, and M. Wulf. Role object. Chapter 2 in *Pattern Languages of Program Design 4*. Addison-Wesley, 2000. Document also available at http://jerry.cs.uiuc.edu/~plop/plop97/Workshops.html#Q2.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented Softwrae Architecture*. Wiley, 1996.

[5] E. Fernandez and X. Yuan. An analysis pattern for reservation and use of reusable entities. *Proc. of the Pattern Languages of Programs Conference* (PLoP'99). 1999. Document also available at http://jerry.cs.uiuc.edu/~plop/plop99.

[6] E. Fernandez and X. Yuan. Semantic analysis patterns. *Proc.of* 19^{th} *International Conference on Conceptual Modeling*, 2000, pages 183–195.

[7] E. Fernandez, X. Yuan, and S. Brey. Analysis pattern for the order and shipment of a product. *Proc. of the Pattern Languages of Programs Conference* (PLoP'00). 2000. Document also available at http://jerry.cs.uiuc.edu/~plop/plop2k.

[8] E. Fernandez. Building complex object-oriented systems with patterns and XP. *Extreme Programming Perspectives*. M. Marchesi, G. Succi, D. Wells, and L. Williams, (Eds.) Addison-Wesley, 2003, pages 591-600.

[9] E. Fernandez, T. Anantvalee, J. Labush, and M. Larrondo-Petrie. Analysis patterns for elections. *Proc. of the Nodic Conference on Pattern Languages of Programs* Viking PLoP'05. Otaniemi, Finland, September, 2005.

[10] E. Fernandez, T. Sorgente, and M. VanHilst. Constrained resource assignment description pattern. *Proc. of the Nodic Conference on Pattern Languages of Programs* Viking PLoP'05. Otaniemi, Finland, September, 2005.

[11] E. Fernandez and X. Yuan. Securing analysis patterns. *Proc. of the* 45^{th} *ACM Southeast Conference* (ACMSE'07). March, 2007. Document also available at http://acmse2007.wfu.edu.

[12] M. Fowler. *Analysis Patterns-Reusable Object Models*. Addison-Wesley, 1997.

[13] M. Fullerton and E. Fernandez. An analysis pattern for customer relationship management (CRM). *Proc. of the* 6^{th} *Latin American Conference on Pattern Languages of Programming* (SugarLoafPLoP'07). 2007, pages 80–90.

[14] C. Larman. Applying UML and patterns (3^{rd} Edition). Prentice-Hall 2006.

[15] D. Ridhle. Composite design patterns. Proc. of the 1997 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'97). ACM Press, 1997, pages 218–228. Document also available at http://www.riehle.org/computer-science/research/1997/oopsla-1997.pdf.

[16] H. Riebeck. The ticket chase. *IEEE Spectrum*, January 2003, pages 72–73.

[17] S. Shlaer and S. MelIor. An object-oriented approach to domain analysis. *Object Lifecycle: Modeling the World in States*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[18] K. Siau and T. Halpin. *Unified Modeling Language: Systems Analysis, Design and Development Issues*. IDEA Group Publishing, Hershey, PA, 2001.