Composing analysis patterns to build complex models: Flight reservation

Zhen Jiang and 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 a convenient way for inexperienced modelers to build complex object-oriented 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) showing 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 also can be of independent value.

---

1. Introduction

When dealing with the specification, design, or implementation of a number of similar applications, common parts can be found. These parts can be specified as patterns that are independent from a particular specification, implementation details, or implementation language. 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 [Fer00a], 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

1

and it can be combined with other related patterns to describe more complex applications.

A possible way to build complex analysis models using SAPs was described in [Fer00a]. However, regardless of the methodology used to build the models, having a complex pattern as a building block makes this work simpler. We show here how to build a complex analysis pattern using a travel application as example. In this example, the composite pattern is built from some existing patterns and one new pattern. The application requirements are described in a common context for all the patterns.

Section 2 describes the requirement for this system while Section 3 presents the atomic patterns that satisfy these requirements. To describe each pattern we follow loosely the templates of [Bus96]. 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 some 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. 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. 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 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.

2

- *A route* is the way followed by a flight from its origin airport to its destination airport. There may be several flights that share the same origin and destination airports. 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 origin (or destination) airport of this span.

- For each flight there are several *connecting flights* (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, she needs a set of one-way tickets. Such a round-trip is a special case of one-way trip where 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 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.
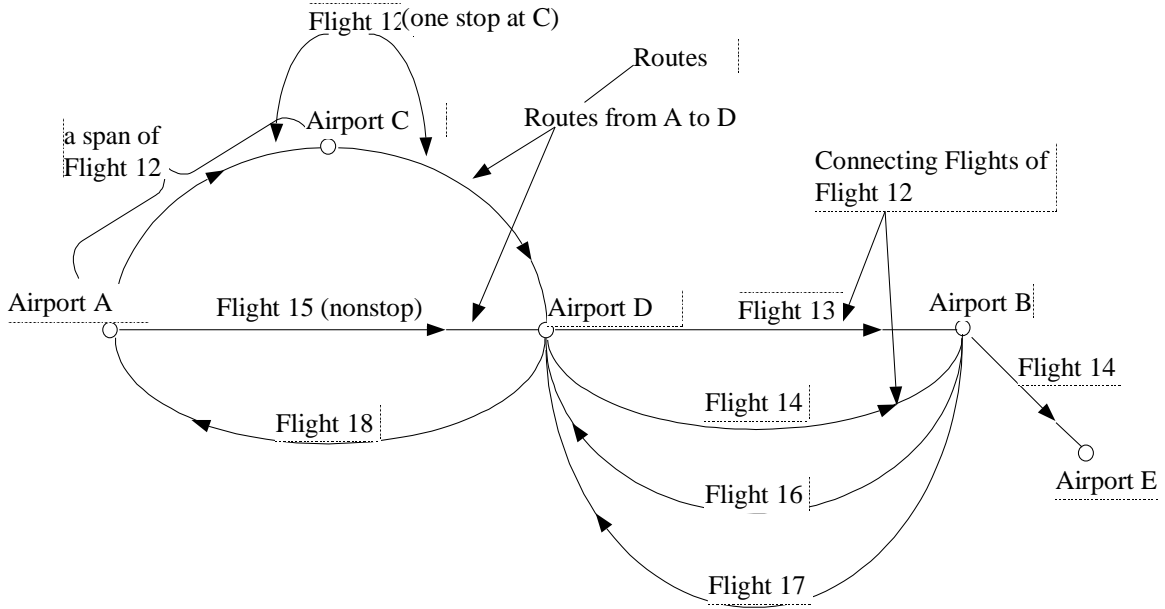
Figure 1. Definitions for the requirements.

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 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 [Fer00], where each flight corresponds to an order line, and the Seat Assignment pattern is a special case of the Assignment pattern [Fer05]. Section 3 presents all the component patterns.
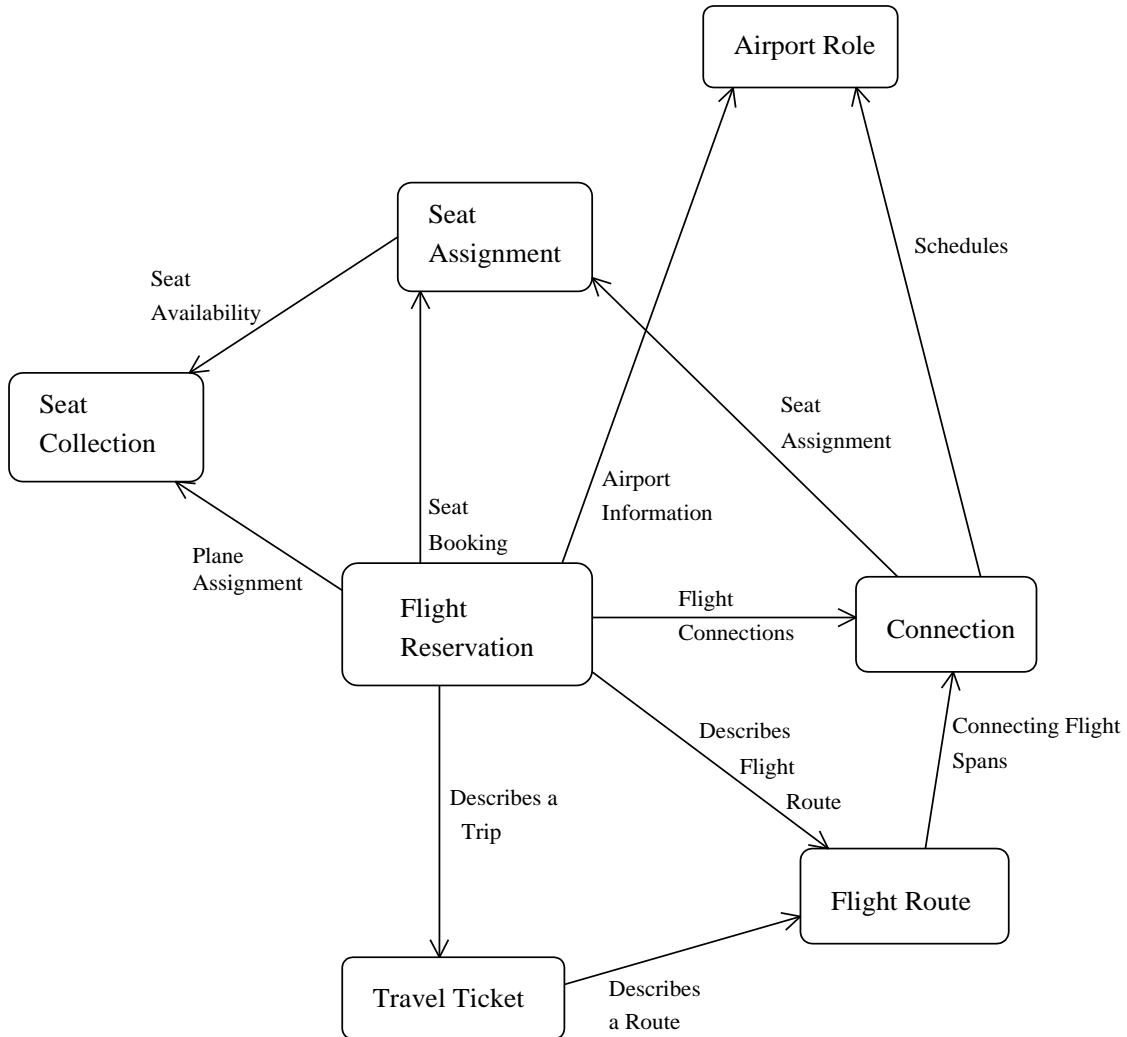

Figure 2. A pattern language for flight reservations

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 to 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.
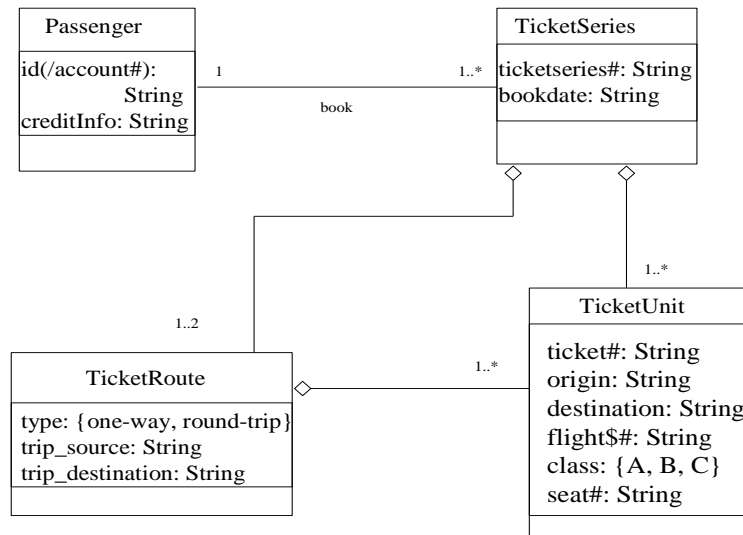
- A passenger is responsible for one or more tickets.

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

Solution
A specialized version of the Order pattern [Fer00]) 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.
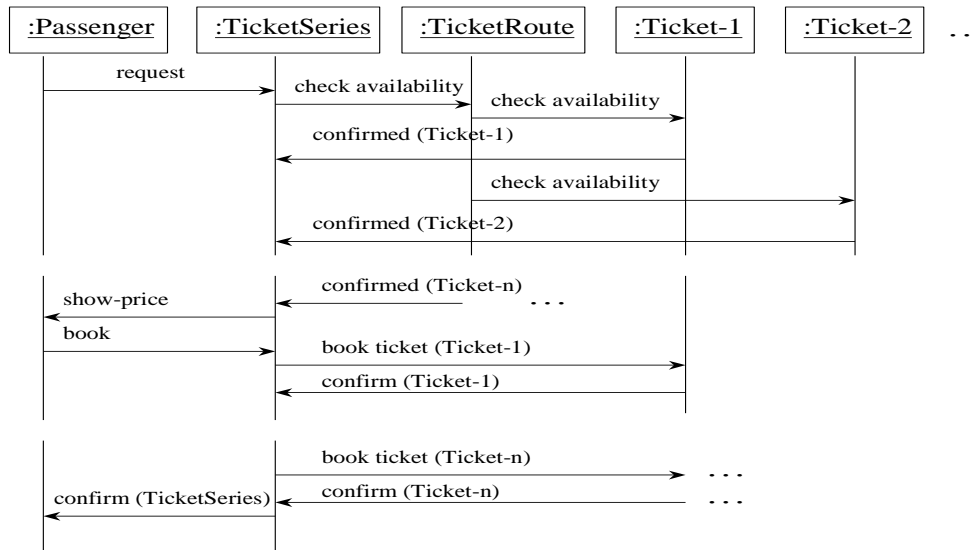
Figure 4: Sequence diagram to order a set of tickets

Consequences

- 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 passenger who will get the ticket(s) and use it (them). We could use a customer with two roles for this purpose.

- 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 lef), concerts (subscription to a series of concerts).

Related patterns

This is a specialization of the Order pattern [Fer00].

3.2 Seat Assignment

Intent

A TicketUnit is assigned to a Seat, Flight, and Span.

Problem

How to 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 Ticket Unit 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 of a general assignment pattern, which can be used to assign resources [Fer05]. 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 a **TicketUnit**. Ticket units are collected in a **TicketSeries,** which collects all the ticket units for a trip.
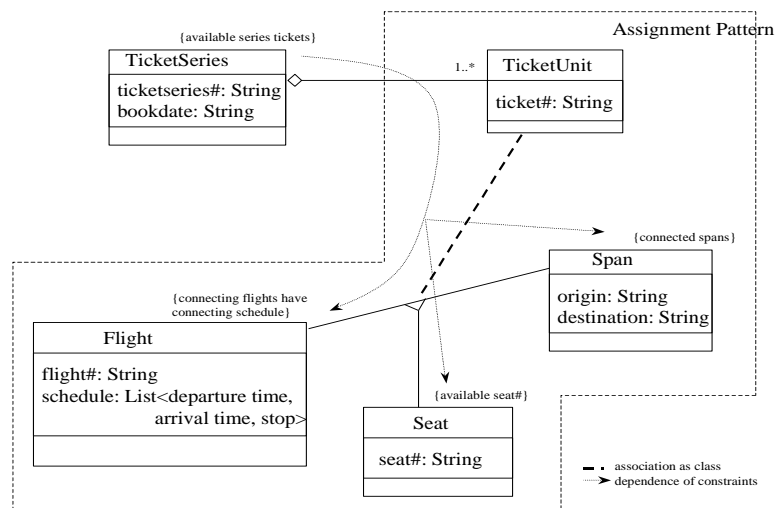
Figure 5: Class diagram for the Seat Assignment pattern

Consequences

- TicketUnit can provide the right to go in a specific span of a flight.

8

- Including a flight seat combination in a ticket indicates that this passenger has a reserved seat in this flight.
- 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 of a general resource assignment pattern [Fer05b], 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 theatre.

Context
Any physical domain where seats must be kept together as a group.

Problem
How to 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 buiding that has seats.

- 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). Alternatively, 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, theatres, stadiums, have collections of seats, usually numbered.

Related patterns

This is a special case of the Whole-Part pattern of [Bus96]. The Whole-Part pattern describes the aggregation of components that together define a semantic unit.
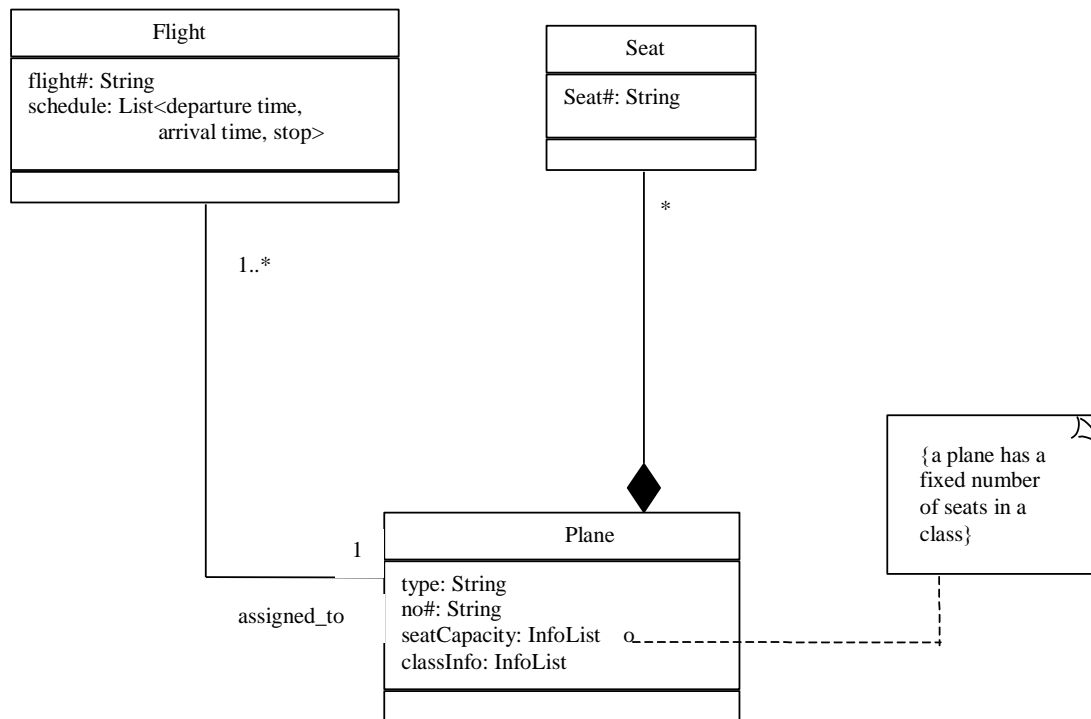

Figure 6: Class diagram for Collection pattern (Seat and Plane)

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.
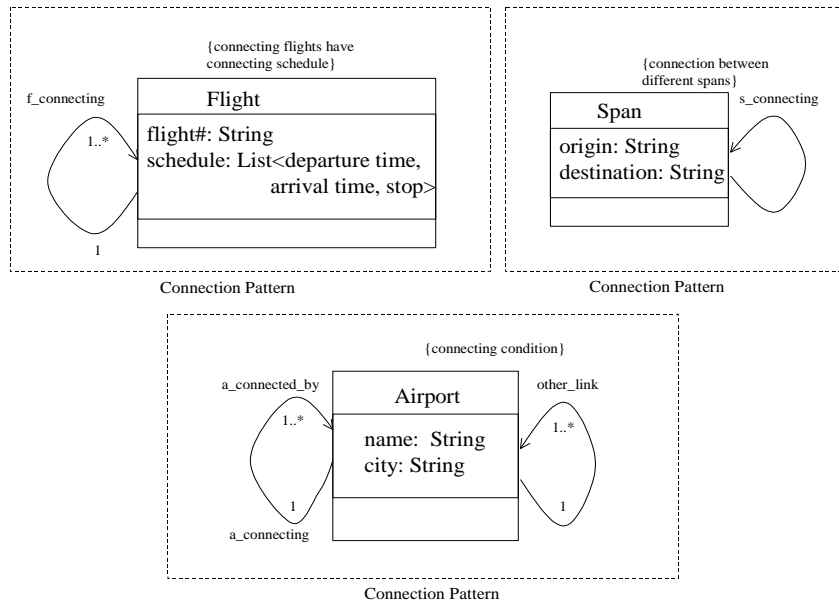

Figure 7: Class diagrams for three instances of the Connection pattern

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 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 in the connecting association, defining attributes of the 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 [Fer05].

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 to 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, 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 if and only if these two airlinks are connected at one airport or the succeeding airport of the first airlink is connected with the preceding airport of the second airlink by other means of transportation. A route is a set of connected basic airlinks connecting all the airports through which it passes.

- All the 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 [Sch91] 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 8b shows a flight   including one route, which in turn, includes a subset of basic airlinks. This subset is described in association class Span.
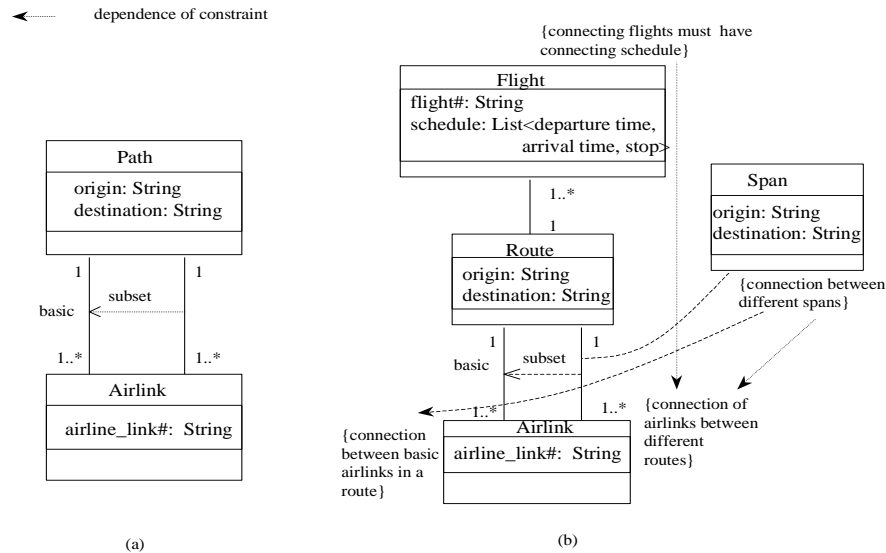

Figure 8: Class diagram for Path pattern and its application in the reservation system

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

13

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 6(b).

Known uses

Routing of electric networks, transportation systems, water distribution systems.

Related patterns

[Sch91] 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 to model 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 [Bau00] can be used here. A core object usually plays several roles and the same 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.
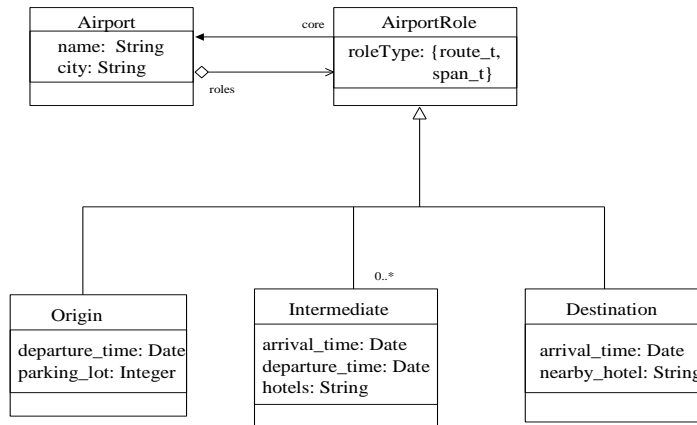

Figure 9: Class diagram for Airport Role pattern

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 of the Role Object pattern [Bau00].

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, 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 (NY). From NY, 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 Hongkong (HK) asks customer *B* to make a reservation

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

Context

Section 2 shows the context for this pattern.

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.
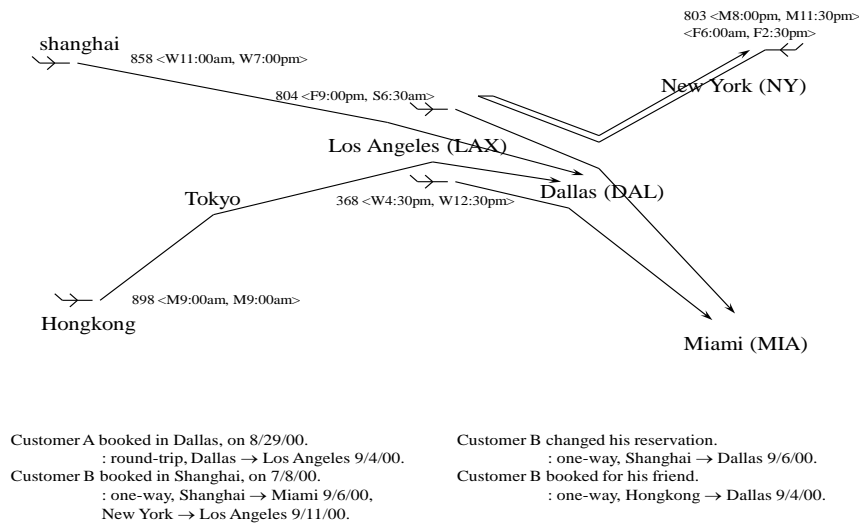

Customer A booked in Dallas, on 8/29/00.
    : round-trip, Dallas → Los Angeles 9/4/00.
Customer B booked in Shanghai, on 7/8/00.
    : one-way, Shanghai → Miami 9/6/00,
    New York → Los Angeles 9/11/00.

Customer B changed his reservation.
    : one-way, Shanghai → Dallas 9/6/00.
Customer B booked for his friend.
    : one-way, Hongkong → Dallas 9/4/00.

Figure 10: Some examples using the reservation system

Solution

Figure 11 combines all the patterns seen earlier and includes all their requirements. A **Passenger** books a **Ticket Series** consisting of some **Ticket Units**. The series defines some **TicketRoutes**, where each route is described by a Ticket Unit. A TicketRoute consists of one or more **Flights.** Each flight has Route defined by one or more **Airlinks.** A flight is assigned to a **Plane,** which has a set of seats to be used by

passengers. In each **Span** of a flight, some seats are available for use. A quaternary association relates ticket units to flights, spans, and seats. Airlinks connect **Airports**, which can have the roles of **Origin, Intermediate**, or **Destination** for a given airlink.

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 < M 8: 00pm, M 11: 30pm; F 6: 00am, F 2: 30pm >, and select another to come back, i.e., 368 < W 4: 30pm, W 12: 30pm >. Although the route of flight 803 covers the span from LAX to DAL and
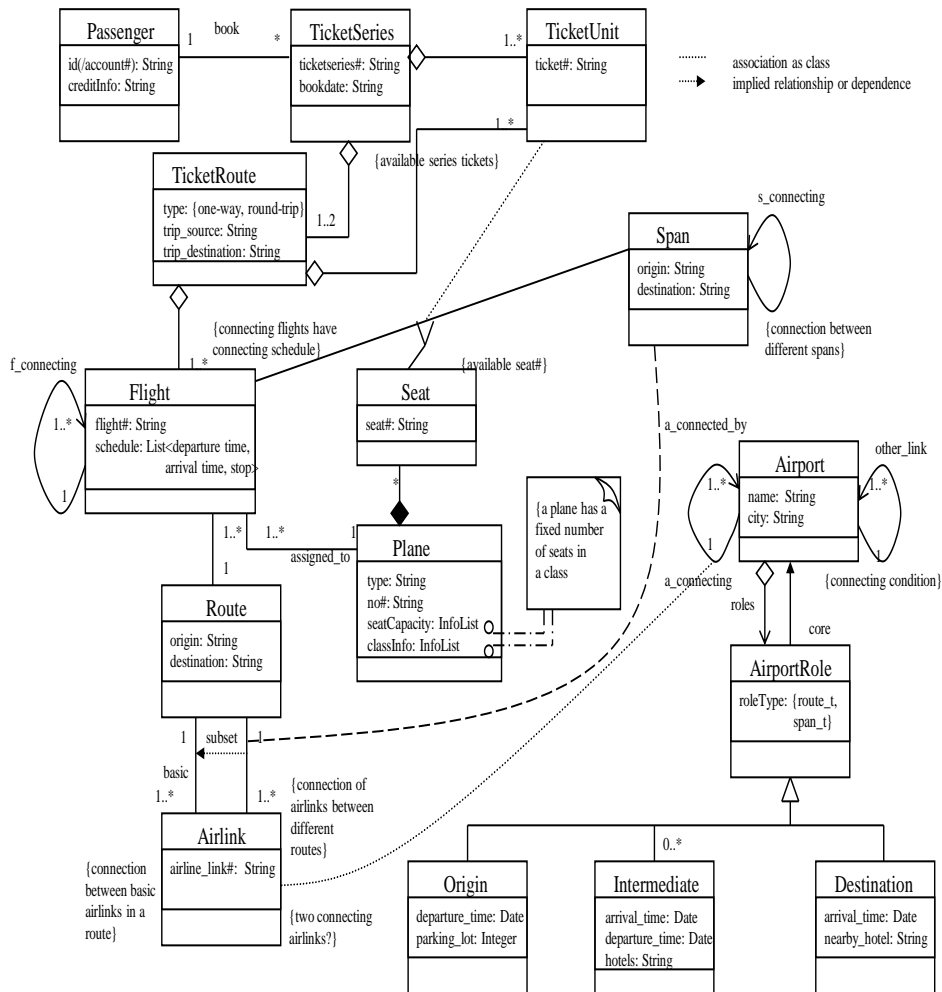

Figure 11. Class diagram for complete flight reservation system.

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 a set represents a span using a 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 seats in the plane, he may use another flight. Finally, customer *A* will be satisfied by a seat in the plane. As shown in Figure10, *J17* in $< B767, 131 >$ and *F12* in $< B737, 189 >$ are such 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 $< Shanghai,LAX>$ 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 NY. As the arrival time of NY is earlier than the departure time of flight 803, the passenger may select the connecting span of flight 803 from NY 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 the 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).
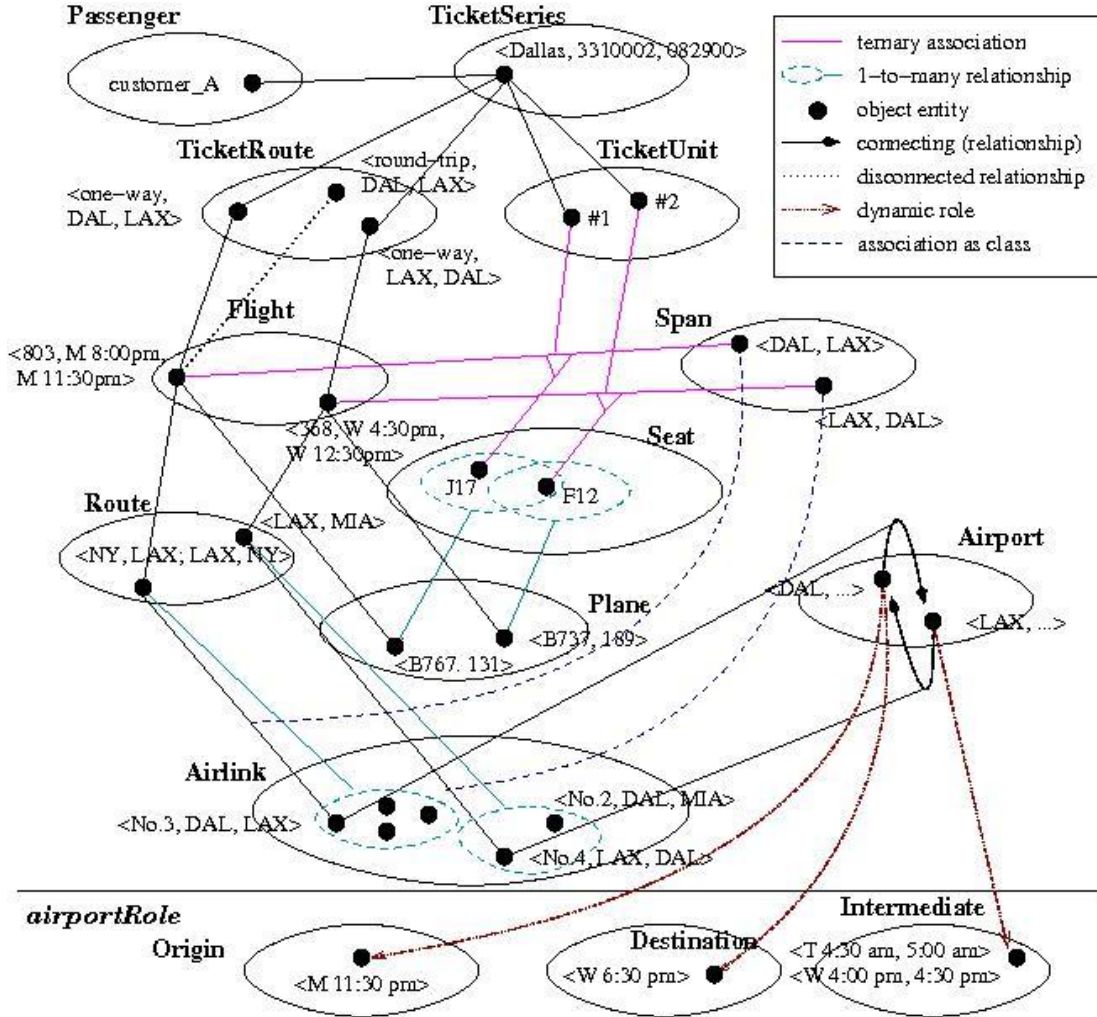
Figure 12: Domain analysis of a reservation system for customer A (round-trip)

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 selects the simplest span for him to go to DAL. Instead of using the connecting spans of flight 858 and flight 368, he selects the span directly to DAL from the basic airlinks in the route of flight 858. After two days, he continues his trip to MIA using a connecting span of connecting flight 804. It is advantageous here to keep most of the feasible parts of the trip and change to a more convenient plan dynamically and easily. In Figure 14, the customer only changes the connecting spans at DAL and chooses the simplest span to reach DAL. The related connecting flights and the connecting spans based on the basic airlinks of the routes which are already available in the system provide 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 the flight 898 (from HK to DAL), a seat of 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 all in the family. It also facilitates the management of the information for airports and flights.
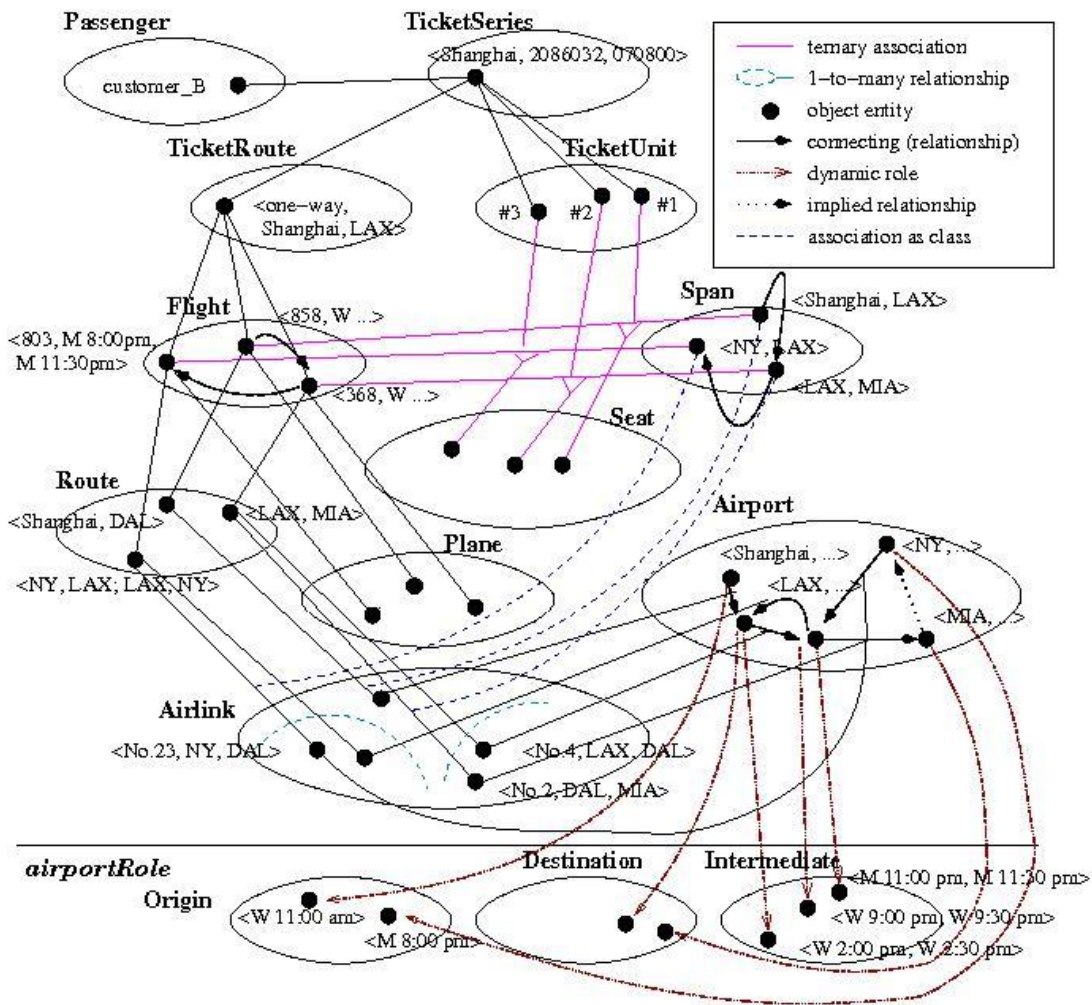
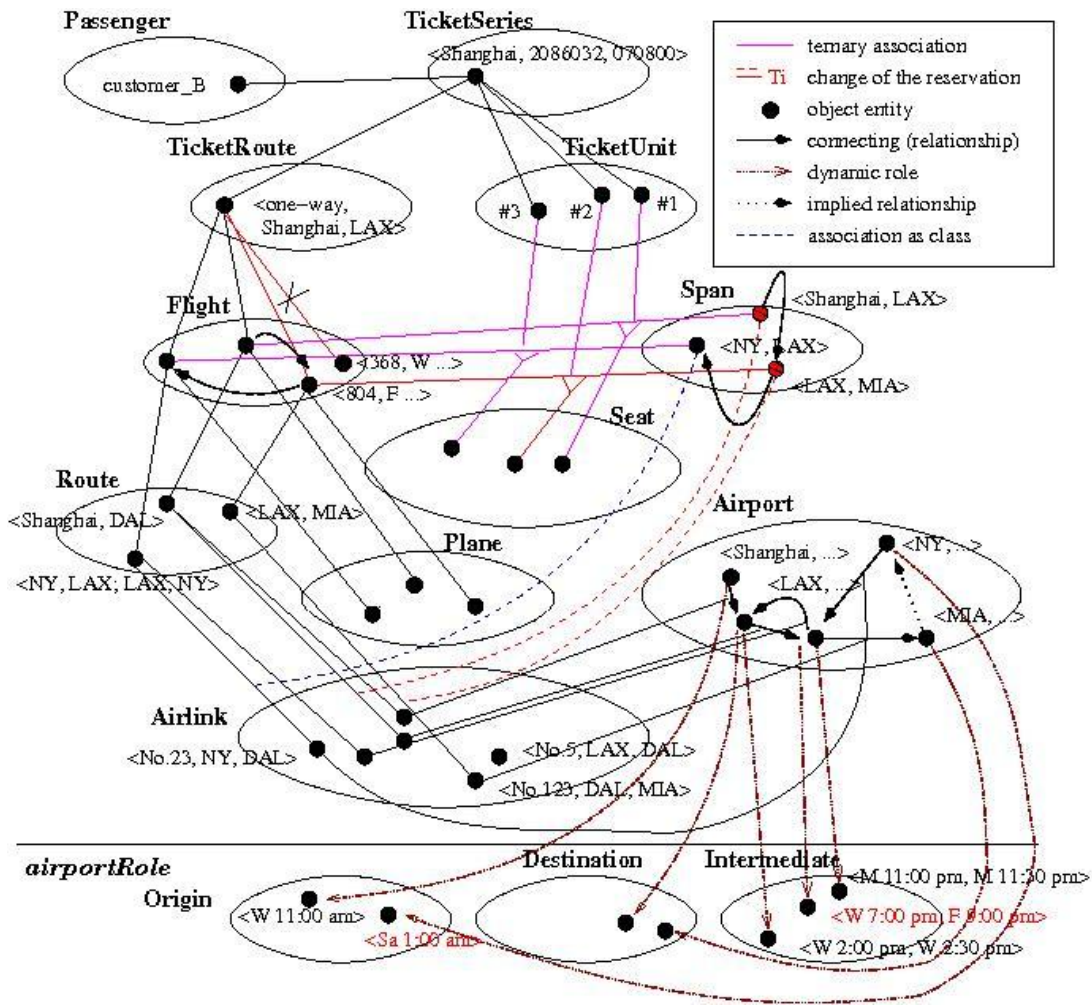Figure 13: Domain analysis of a reservation from Shanghai to LAX for customer B

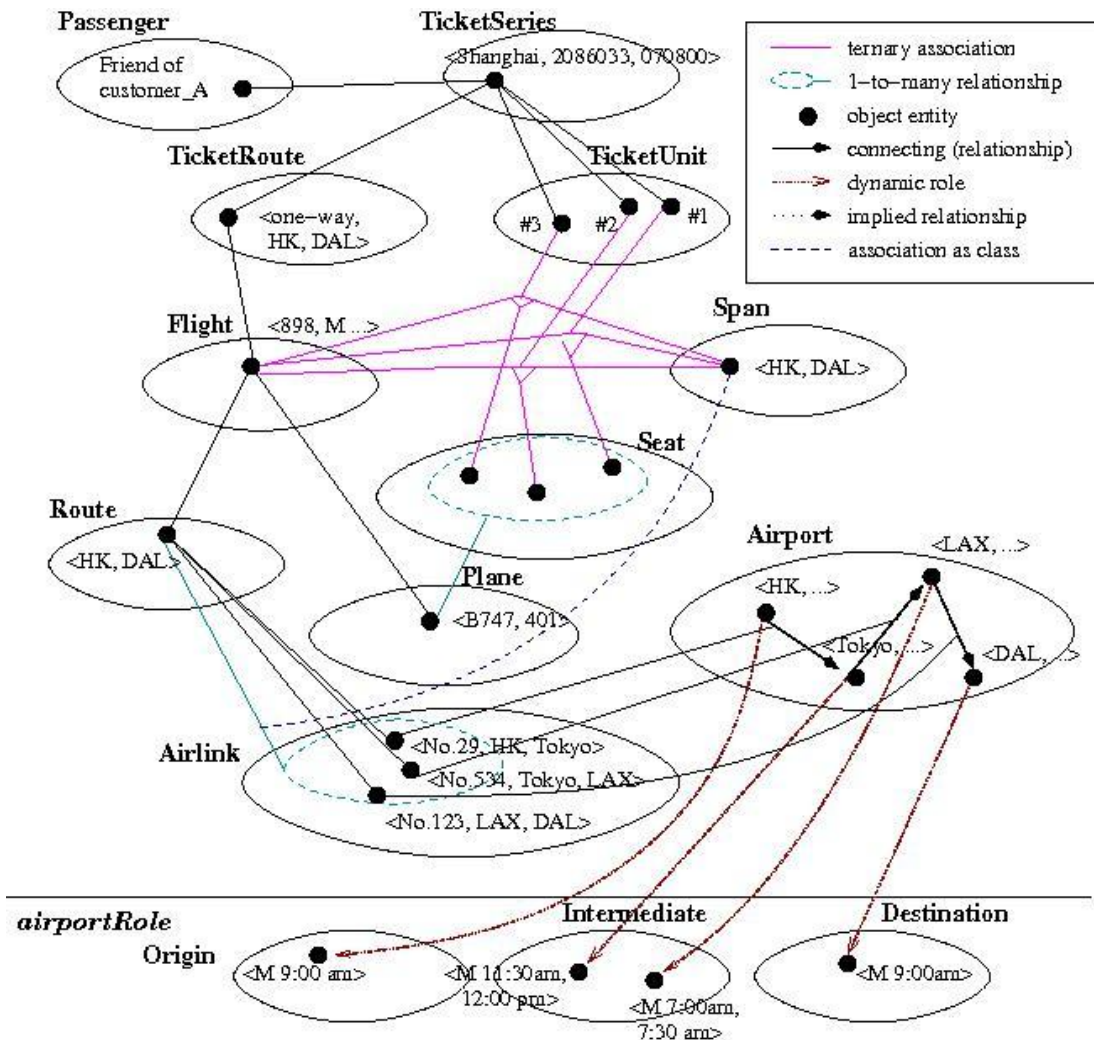Figure 14: Domain analysis of a changed reservation for customer B

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

Known uses

Orbitz is a good example of a system that uses a similar pattern. 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 trip

- 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 extending 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 ticket, delays, and flight cancellations.

- Alarms; fro example, when a flight is sold out.

- Historical information.

- Billing and payment policies, e.g., order cancellation and refunding.

- Personal identification.

These aspects should be completed with additional patterns.

Known uses
Orbitz provides possible routes between any two destinations, including flights of any airline. These can be converted into reservations and later into tickets.
American Airlines provides similar functions but including only their own flights and those of their alliance partners.

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 problems 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 [Fer00a] and can also be used to define Secure SAPs, where the functionality is complemented with authorization and authentication aspects [Fer07]. 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 [Fer03], which is another proof of its possible value. There are other object-oriented approaches based on patterns, e.g., several approaches are discussed in [Sia01], and we don't claim that our approach is better than any of these methods, 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 case study is of intrinsic interest because of its economic importance [Rie03]. It is clear that software for flight reservation 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. What is more important, the use of analysis patterns can help build good conceptual models to designers who have little experience.

Acknowledgements

References

[Bal91] C. Ball, "An object oriented analysis of air traffic control", MITRE Corp, August 1991, http://www.mitrecaasd.org/library/tech_docs/pre1999/wp90w542/

[Bau00] D. Baumer, D. Riehle, W. Siberski, and M. Wulf, ''Role Object'', Chapter 2 in *Pattern Languages of Program Design* 4, Addison- Wesley, 2000.
Also in: http://jerry.cs.uiuc.edu/~plop/plop97/Workshops.html#Q2

[Bla98] M Blaha and W. Premerlani, *Object-oriented modeling and design for database applications,* Prentice-Hall, 1998.

[Bus96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented software architecture,* Wiley 1996.

[Fer99] E.B. Fernandez and X. Yuan, ''An Analysis Pattern for Reservation and Use of Reusable Entities'', *Procs. of Pattern Languages of Programs Conference,* PLoP'99. http:/ /jerry.cs.uiuc.edu/~plop/plop99

[Fer00] E.B. Fernandez, X. Yuan, and S. Brey, '' Analysis Patterns for the Order and Shipment of a Product'', *Procs. of Pattern Languages of Programs Conference*, PLoP 2000, http://jerry.cs.uiuc.edu/~plop/plop2k

[Fer00a] E.B. Fernandez and X. Yuan, "Semantic Analysis patterns", *Procs. of   19th Int. Conf. on Conceptual Modeling, ER2000*, 183-195.

[Fer03] E.B.Fernandez, "Building complex object-oriented systems with patterns and XP", Chapter 47 in "*Extreme Programming Perspectives*" , M. Marchesi, G. Succi, D. Wells, and L. Williams, (Eds.) , Addison-Wesley, 2003, pp. 591-600.

[Fer05a] E. B. Fernandez, T. Anantvalee, J. Labush, and M. M. Larrondo-Petrie,."Analysis patterns for elections" in *Proceedings of the Nordic Conference on Pattern Languages of Programs*, Viking PLoP 2005, Otaniemi, Finland, 23-25 September 2005.

[Fer05b] E.B.Fernandez, T. Sorgente, and M. VanHilst, "Constrained Resource Assignment Description Pattern". *Proceedings of the Nordic Conference on Pattern Languages of Programs*, Viking PLoP 2005, Otaniemi, Finland, 23-25 September 2005.

[Fer07] E.B.Fernandez and X.Y. Yuan, " Securing analysis patterns", *Procs. of the 45th ACM Southeast Conference (ACMSE 2007)*, March 23-24, 2007, Winston-Salem, North Carolina, http://acmse2007.wfu.edu

[Fow97] M. Fowler, *Analysis Patterns-Reusable Object Models*, Addison-Wesley, 1997

[Ful07] Mei Fullerton and E. B.Fernandez, "An analysis pattern for Customer Relationship Management (CRM)", *Procs. of the 6th Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP'2007),* May 27-30, 2007, Porto de Galinhas, Pernambuco, Brazil, 80-90.

[Rie97] D. Riehle. "Composite Design Patterns." In *Proceedings of the 1997 Conference on Object-Oriented Programming Systems, Languages and Applications* (OOPSLA '97). ACM Press, 1997. 218-228. http://www.riehle.org/computer-science/research/1997/oopsla-1997.pdf

[Rie03] H. Riebeck, "The ticket chase", *IEEE Spectrum*, January 2003, 72-73.

[Sch91] S. Shlaer and S. J. MelIor, ''An Object-Oriented Approach to Domain Analysis'', in *Object Lifecycle: Modeling the World in States,* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[Sia01] K. Siau and T. Halpin, Eds., *Unified Modeling Language: Systems Analysis*, *Design and Development Issues*, IDEA Group Publishing, Hershey, PA, 2001.