

Part I

1 Summary of the requirements

The web holiday system provides an online booking system for packaged family holidays. A main booking system interfaces with the Web Holiday System. The latter is capable of allowing prospective customers browse through the catalogue, check availability of holiday packages and read reviews of holidays. After booking a holiday package the (now) booked customer is able to extend her holiday. The main booking system or the administrator also allows extending holidays, maintaining the holiday catalogue including the reviews and the availability of packages. All these tasks are possible via the web holiday system that partly acts as interface to the main booking system.

Assumptions: To have a comparable result with the result from the Entity-Relationship-Modelling (ERM) approach in coursework 1 where the task was to design and implement a *simple prototype* of a data-engine for a web-based holiday-booking system for *packaged family holidays*, similar assumptions are being made: The family is treated uniformly as an entity, i.e. for instance the number of children and their age are not taken into account. Also, the prototype booking system is not meant to take care of (various means of) transportation or different types of hotels since holiday trips are supposed to be entire packages, i.e. entities. Finally, the client is expected to pay per credit card to complete the booking-process.

2 Scenario descriptions of the three use cases

For the following scenarios the actor (me, first hand experience) browses the website i.e. the e-commerce system of a holiday company such as Classic Ski Ltd Hampshire or Thomson Holidays, a division of TUI UK Ltd.

2.1 Browse through catalogue and check availability

I open the website and wait until it shows up completely offering a form to enter holiday details and a 'read reviews' button. Since I plan to go from Manchester to Athens for Xmas I enter Manchester as departure and Athens as destination. As departure date I type in 22/12/02 and holiday duration of ten days. Clicking on 'check availability' opens a page showing one available holiday package with holiday details confirming the selected locations and dates and the overall price as well as a 'book' button and 'read reviews' button.

2.2 Read reviews of holiday

I open the website and click on the 'read reviews' button which opens another page listing 23 holiday packages from the catalogue having reviews by previous customers. I click on the third one from the top that happens to be the holiday package to Athens. Another page gets loaded both showing the holiday details including a 'book' button and presenting a list of ten reviews, showing both the ratings (number of stars out of five) and the actual comments as text.

2.3 Book a holiday

I continue on the last page of the 'browse catalogue and check availability' scenario which shows the holiday details for my trip to Athens. After pressing the 'book' button I am asked to enter my fullname, email address and my creditcard number, which I enter truthfully and press 'submit' which the website confirms thankfully after some moments.

3 The analysing modelling technique

For this model partly John Hunt's analysis modelling technique is used that consists of the following eight steps that are to be applied iterative.

1. Identify objects and classes
2. Generate use case realisations
3. Prepare a data dictionary
4. Identify associations between objects
5. Identify attributes of objects
6. Organise and simplify classes (using inheritance)
7. Iterate and refine model
8. Group classes into packages

4 Concept-level class diagram based on scenario descriptions

To identify objects and classes for the concept class diagram one looks at the nouns used in the scenarios. Some of the nouns have to be adjusted to serve as concept-classes:

- 'I' → 'WebSiteVisitor'
- 'CheckAvailabilityButton' → 'Availability'
- 'ReviewsButton' → 'Reviews'
- 'BookButton' → 'Booking'
- 'SubmitButton' → 'Registration'

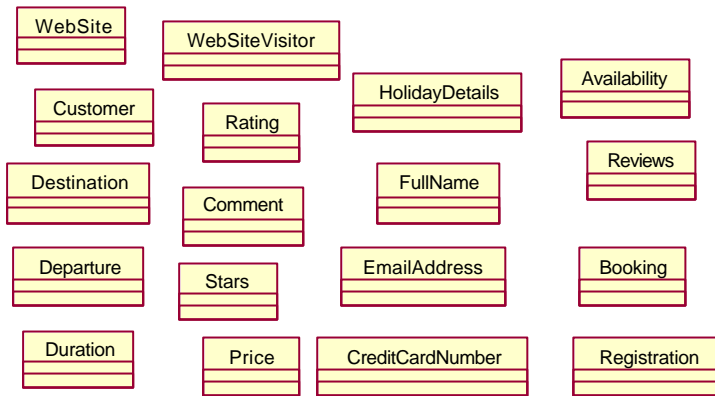


Figure 1: Identified objects and classes within scenario descriptions

These classes are being connected via preliminary associations to design the concept-level class diagram. Note that ‘WebSite’ changed from Class to System.

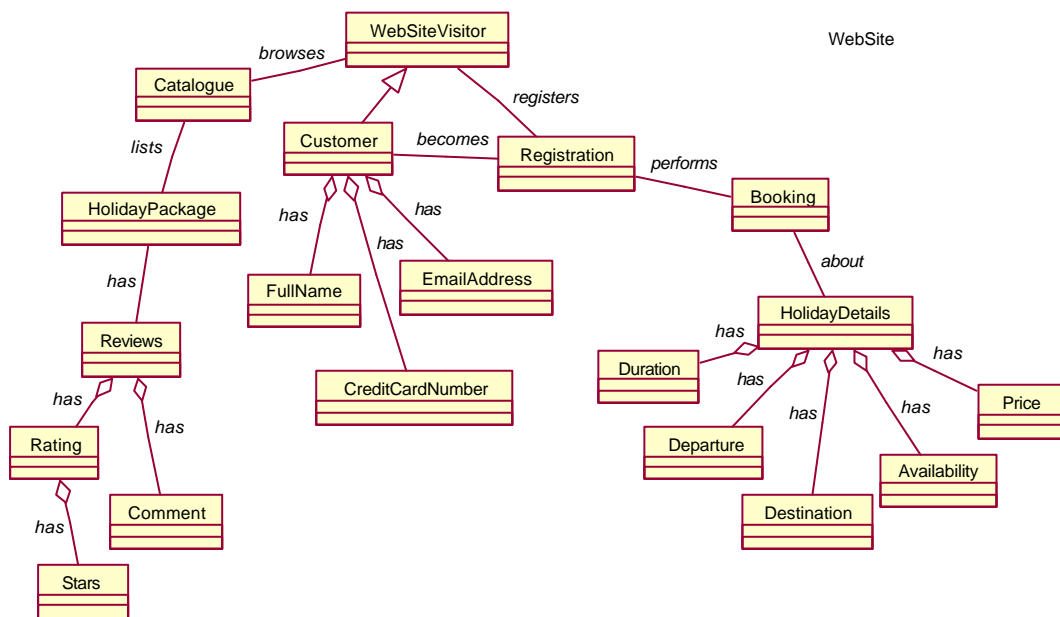


Figure 2: Concept class diagram

This concept class diagram is the result of the first step in John Hunt’s analysis modelling technique. In the next step use case realisations are generated for each use case. These are first described in more detail using the following forms.

5 Use case descriptions for the three use cases

Use Case Name:	[Browse] Browse through catalogue and check availability
Primary Actor:	Prospective customer
Other Actors:	Booked customer
Value Proposal to Actor(s)	Find a holiday package that is worth booking.
Basic Course of Events: Path [Browse 1]	<p>This use case begins when ...</p> <ul style="list-style-type: none"> • Prospective Customer (Actor) opens website. Actor enters departure location and date, duration and destination, and clicks 'check availability'. • One holiday package is presented with details such as destination, duration and cost.
Alternative Path: Path [Browse 2]	<ul style="list-style-type: none"> • Prospective Customer (Actor) opens website. Actor enters departure location and date, and clicks 'check availability'. • A list of ten holiday packages is presented with details such as destination, duration and cost.
Exception Path: Path [Browse 3]	<ul style="list-style-type: none"> • Prospective Customer (Actor) opens website. Actor enters departure location and date, duration and destination, and clicks 'check availability'. • A message appears stating that no packages are available. • Actor closes website.
Assumptions:	Actor uses a web-browser capable to interpret and display the required content and functionality ¹ .
Pre-conditions:	There are available holiday-packages.
Post-conditions:	One or more holiday-packages are being presented.

¹ Most holiday booking websites need a browser with high JavaScript capabilities since data in forms is updated without reloading the site from the server. Thus, an overall exception path could also be the usage of a text-based or alternative browser such as Lynx or Opera.

Use Case Name:	[Read] Read reviews of holidays
Primary Actor:	Prospective customer
Other Actors:	Booked customer
Value Proposal to Actor(s)	Read other people's experience/opinion of holiday packages to take into account in deciding for a holiday.
Basic Course of Events: Path [Read 1]	<p>This use case begins when ...</p> <ul style="list-style-type: none"> • Prospective Customer (Actor) opens website. Actor clicks on 'read reviews'. • A list of packages with available reviews appears. Actor picks third package. • A list of detailed reviews by prior customers shows up.
Alternative Path: Path [Read 2]	<ul style="list-style-type: none"> • Prospective Customer (Actor) opens website. Actor enters departure location and date, duration and destination, and clicks 'check availability'. • One holiday package is presented as in path [Browse 1]. Actor clicks on 'read reviews'. • A list of detailed reviews by prior customers shows up.
Exception Path: Path [Read 3]	<ul style="list-style-type: none"> • Prospective Customer (Actor) opens website. Actor enters departure location and date, duration and destination, and clicks 'check availability'. • One holiday package is presented with details of flight departure, cost. Actor chooses this package. • Actor clicks on 'read reviews'. • A message appears stating that no reviews to this package are available. • Actor closes website.
Assumptions:	Actor uses a web-browser capable to interpret and display the required content and functionality.
Pre-conditions:	There are available holiday-packages; there are reviews to these packages.
Post-conditions:	One or more reviews are being presented.

Use Case Name:	[Book] Book a holiday
Primary Actor:	Prospective customer
Other Actors:	Booked customer
Value Prop. to Actor	Have a holiday booked to look forward to.
Basic Course of Events: Path [Book 1]	<p>This use case begins continues path [Browse 1].</p> <ul style="list-style-type: none"> • Prospective customer (Actor) chooses package. • Website asks to enter name, address and credit card details. Actor types in data. • After submission, website responds with ‘thank you booking’-message.
Alternative Path 1: Path [Book 2]	<ul style="list-style-type: none"> • Path [Browse 2] continued. • Prospective customer (Actor) chooses third package. • Website asks to enter name, address and credit card details. Actor types in data. • After submission, website responds with ‘thank you booking’-message.
Alternative Path 2: Path [Book 3]	<ul style="list-style-type: none"> • Path [Read 2] continued. • Prospective customer (Actor) clicks on ‘book holiday’. • Website asks to enter name, address and credit card details. Actor types in data. • After submission, website responds with ‘thank you booking’-message.
Exception Path: Path [Book 4]	<ul style="list-style-type: none"> • Path [Browse 1] continued. • Prospective customer (Actor) chooses package. • Website asks to enter name, address and credit card details. Actor types in data. • After submission, website responds with ‘sorry, package booked out’ - message. (While typing in data other prospective customers may have booked this package.) • Actor closes website.
Assumptions:	Actor uses a web-browser capable to interpret and display the required content and functionality.
Pre-conditions:	One or more packages are being displayed as choice.
Post-conditions:	A holiday package is booked.

Part II

6 Worked example demonstrating John Hunt's analysis modelling technique

6.1 Use case realisations for the three use cases

Since use case 1 and 2 can start from the same boundary class these are drawn within one diagram as follows.

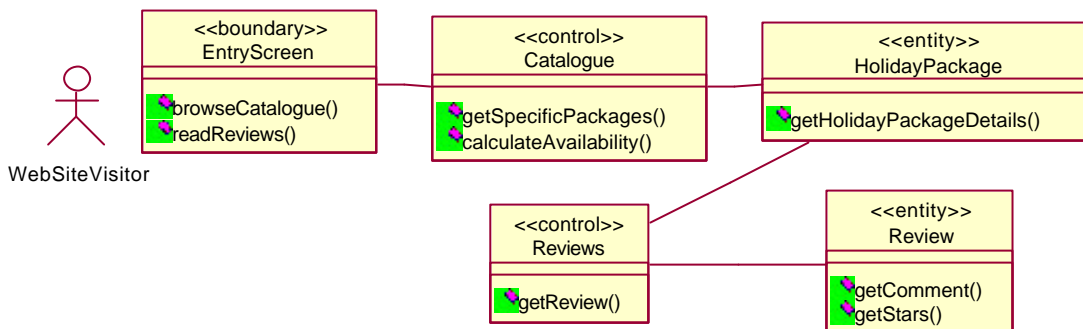


Figure 3: Browse through catalogue and check availability, read reviews of holidays

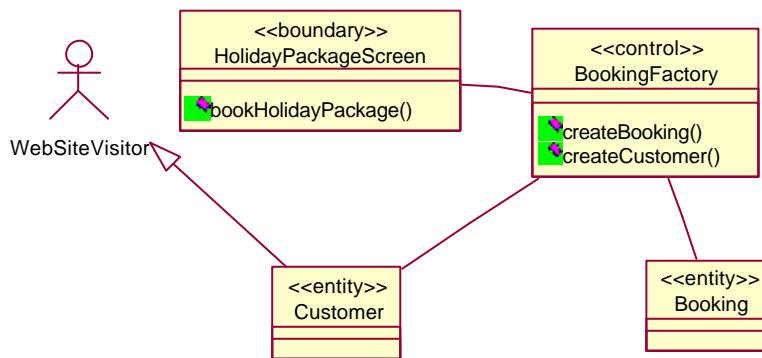


Figure 4: Book a holiday

6.2 Data dictionary

The next step is to create a data dictionary describing each class with name, attributes, methods and other comments. Note that this state of the dictionary describes the final design.

<i>Name</i>	<i>Attributes</i>	<i>Methods</i>
WebVisitor	—	—
	The WebVisitor is the most generalized person in this scenario. Since browsing the web is done anonymously ² , there is no attribute and no operation. The class can be seen as abstract.	
Registered-WebVisitor	fullname, password, email	
	This class is specialised into Customer and Administrator and hence can be seen as abstract, too.	
Customer	status	getStatus, setStatus
	For the Booking of a HolidayPackage the WebVisitor has to enter the data needed to fill the attributes and therefore a new Customer is created. The attribute status could be used to distinguish between ‘more valuable’ and normal customers for example.	
Administrator	—	—
	Administrator may book HolidayPackages (via generalization) for evaluation purposes. One could think of a rights-system to allow certain actions to different levels of administrators/users.	
Catalogue	holidayPackage	getPackages, createPackage
	The Catalogue contains all HolidayPackages. Catalogue might inherit from a Collection class, especially a Set class such as java.util.Vector in Java to easily offer the possibility to add and get items.	
Holiday-Package	destination, contingent, costperday, review	calculateAvailability, getReview, getDestination, getContingent, getCostPerDay, addReview, setDestination, setContingent, setCostPerDay, deletePackage
	HolidayPackage contains all the detailed information about the specific packages. The contingent is maintained by the main booking system. Its value is used to calculate the availability of one package between certain days: if on every day within the given period the sum of instances of Bookings over this instance of HolidayPackage is smaller than the contingent, there are packages available (contingent – booked packages). Similar to Catalogue, HolidayPackage also acts as a collection/list for reviews.	

² Excluding server-logged data such as IP-address, browser and operating system and maybe screen resolution and referrer URL. One could imagine to instantiate the object with this data, though.

Review	comment, stars	getStars, setComment, setStars, deleteReview, getComment
	A Review consists of the textual comment and the numerical number of stars. Reviews are 'added' to instances of HolidayPackage.	
Booking	from, to, holiday, customer	extend, createCustomer, setDateFrom, setDateTo, calculateTotalCost, cancelBooking
	Instances of Booking store the dates of specific holidays of specific customers. To extend the stay, only Date 'to' has to adjusted which can be done easily via myBooking.extend(byNumberOfDays).	

6.3 Associations, attributes, organisation and simplification

In the first iteration the following associations between the revised classes are found. See Figure 5:

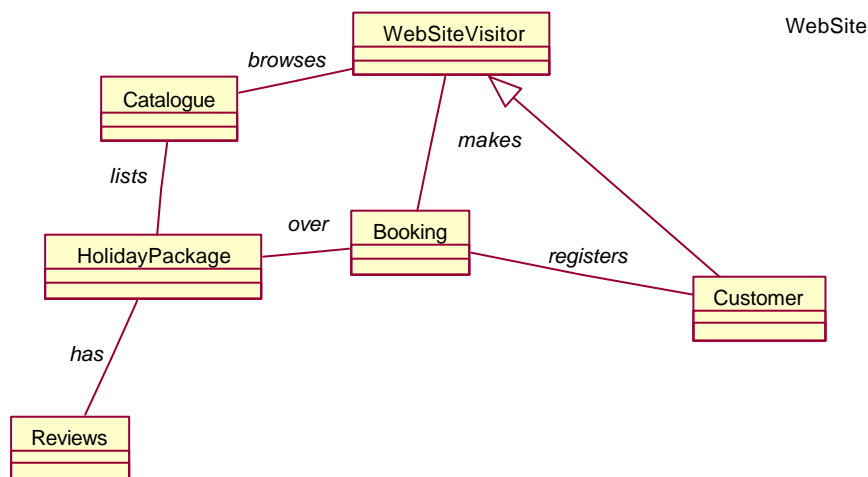


Figure 5: Class diagram with associations, not showing attributes/operations

As seen in Figure 5, many classes have been removed since these concepts now serve as attributes or are superfluous meaning the same thing.

- Registration plus Booking is replaced by Booking alone.
- HolidayPackage serves itself as Collection for individual Reviews, thus the class Reviews is abandoned, associating class Review directly to HolidayPackage.
- Review now has the attributes comments and stars.
- HolidayPackage now has the attributes destination and costperday formerly seen as individual classes associated to HolidayPackage.

Moreover, all methods from control classes have been (slightly renamed and) moved into entity classes and empty control classes have been removed, such as BookingFactory for example.

An administrator class as specialisation of RegisteredWebVisitor is being introduced to maintain the HolidayPackages via the Catalogue.

The system is renamed from ‘Website’ to ‘Web Holiday System’.

For the result of changes in classes, associations and attributes in the second iteration please see Figure 6: Design level diagram.

7 Final design level class diagram

The final design level class diagram of the second iteration shows all details of attributes and methods such as visibility and parameters, the structures and associations with names and multiplicity. Note that these are all entity-classes; control classes have been removed successfully; some boundary classes such as an anticipated LoginScreen are missing³. One could also think of introducing more detailed information about HolidayPackages and classify them into different types of activities, such as Skiing, Beaching or Culture for example. Doing that, also methods to get (and set) HolidayPackages according to specific activity types have to be designed.

The aim in designing this model was to keep it simple but scaleable.

- One could think of support for family-size and reduced prices for children. The agency might offer bonuses for clients with bookings or many reviews; the price of the prolonged stay might be lower. What about a correlation between availability and price? If availability goes down, price might go up by some factor...
- Timestamps in various objects might be introduced to keep track of the booking-date or date of review writing. Especially the latter might be checked not to take place before the actual (end of) holiday.
- The contingent for availability merely is a variable, i.e. the travel agency has ‘pre-booked’ a more or less fixed quantity of places, which has the advantage that this data does not need to be changed if a trip is booked.
- Building a complete model for airplane traffic/transportation is not considered to be part of the holiday booking system since this type of information is expected to be incorporated on the travel agencies’ website via the air travel-companies’ interface.

³ Unfortunately, the desired changes cannot be drawn since all computers with Rational Rose are in use this last day.

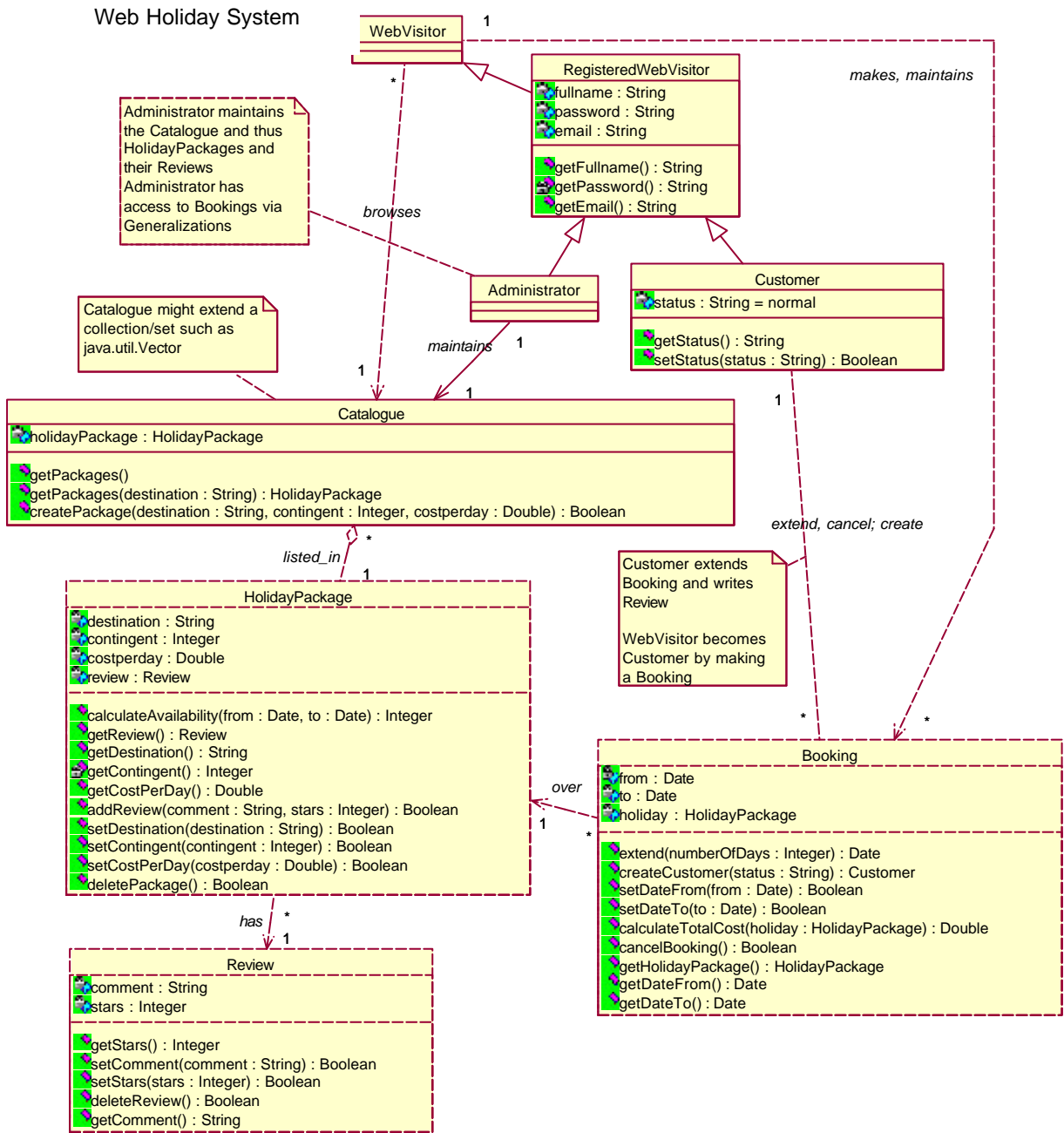


Figure 6: Design level diagram

Class Booking: attribute *customer* : *Customer* missing in diagram.

Part III

8 Assessment, comparing OO and ERM

8.1 *The programming paradigm*

Whatever programming paradigm a programmer chooses, the trend is pushing the ‘data logical’ towards the ‘information logical’. Abstractions become less and less – just imagine the days of Assembler. The more complex the technology the less involved the user has to be with it. The definition of classes does not resemble how data structures and operations are made interpretable to machines. The information model looks very much like a real world projection, yet it could/should result in executable code.

Using the object-oriented (OO) approach programmers/designers can think in terms of **real-world concepts**/objects like bank-accounts or holidays and customers. In the entity-relationship model (ERM) one has to be much more aware of the limited possibilities of data structures and **data types**. In ERM (and XML DTD) the designer cannot ‘invent’ new data types like in OO (and XML Schema) where objects can resemble new data types built out of known primitive data types or even out of other objects. For example, one could build an object ‘Address’ containing street-name, zip-code and country as attributes. This class could then be used as data type for an attribute ‘destination’ in ‘HolidayPackage’, for instance.

8.2 *Differences and similarities in the results*

Nevertheless, there are similarities in the result. **Objects similar to the entities** in the ERM can be found within the OO-structure. For example, the following entities, or objects respectively, share the same core set of attributes: Booking, HolidayPackage and Customer.

Thinking of the **three architectural layers**, namely *presentation*, *application* and *data*, the ERM just deals with the data; OO mainly with the application but interweaves with both the data and the presentational layer (mainly user interfaces). It is only for **persistent storage** where databases are the state-of-the-art technique using queries to efficiently receive the desired bit of data. The OO-approach can be seen as a **top-down** approach compared with the ERM that may be described as **bottom-up**.

Similar to the ERM it is necessary in OO to avoid **redundancy**, too. Whereas in ERM the process of normalisation is a well-defined technique to avoid redundancy – there are even different known stages of levels (**normal forms**) – for the OO designer

there does not seem to be such a mathematically based technique for that purpose. Design is less straight-forward offering more alternative routes.

Whereas in ERM the data is **queried** using SQL (connected via an O/JDBC-driver) this functionality is captured in methods that are part of the OO-structure. Thus '**messages**' can be passed across multiple objects – in SQL this only can be done via complex **joins** over multiple tables. In XML and ERM there are **IDs** used to refer to other pieces of the data-structure. In OO this can be done via references to or handling the actual **instance** of the object.

Both in OO and ERM it is not always easy to distinguish whether a concept should be implemented as entity/**object or attribute**. Even in designing an XML-structure (DTD, XML-Schema) this is not determined. Instead of attaching attributes to entities to specify particular types of an entity/object, one can build hierarchies using inheritance in OO.

Overall, OO offers much more alternatives. There is no *one best way* (F. W. Taylor) to do it. Machine independence, hierarchy allowing inheritance, actions between objects and re-use of packages are only some advantages.

8.3 Comment on John's hunt

I tend to argue that John Hunt's method is an excellent way to learn how to come up with a good OO model – for more or less experienced programmers/designers the method seems to be too strict in my eyes; especially the sequence of steps cannot always be followed intuitively. For example the three stages identifying associations, identifying attributes, and organising/simplifying classes using inheritance are interdependent with each other and thus should not be treated as individual steps. I consider the technique perfect for serving as a guideline but not as a set of rules.