
Errata & Addendum to the 1st print of the book

(the 2nd print – November 2001 – incorporated the content of this document)
# Changes – to the 1st print of the book

<table>
<thead>
<tr>
<th>DATE OF CHANGE</th>
<th>TYPE OF CHANGE</th>
<th>CHANGE WITH REGARD TO:</th>
</tr>
</thead>
</table>
| 9-April-2001   | Correction     | Chapter 4, p. 119 - Section 4.2.1.2.3 – Figure 4.2  
Chapter 4, p. 119 - Section 4.2.1.2.3 – paragraph before Figure 4.2  
Chapter 4, p. 151 – Question R5 |
| 24-April-2001  | Minor correction | Chapter 4, p.153 – Exercise Questions – Point 7, 1st sentence  
Chapter 4, p.153 – Exercise Questions – Point 10, 2nd sentence |
| 30-April-2001  | Minor improvement | Chapter 2, p. 44 - Section 2.1.5.2.3  
Chapter 2, p. 57 - Section 2.2.3.2 – 1st paragraph, last sentence  
Chapter 2, p. 70 - Section 2.2.6.1 – Analysis Tutorial: Step 16, first paragraph, last sentence |
| 30-April-2001  | Minor correction | Chapter 2, p. 72 - Section 2.2.6.2 – Figure 2.38  
Chapter 2, p. 110 - Section 4.2.1.1.3 – 1st paragraph, 1st sentence |
| 30-April-2001  | Correction     | Chapter 4, p. 132 - Section 4.2.4.3 – Figure 4.7  
Chapter 4, p. 142 - Section 4.3.2.3 – Figure 4.13 (corrected in the 2nd print but a new error introduced – ref. the next Errata & Addendum document for details) |
| 30-April-2001  | Minor extension | Chapter 2, p. 79 – Question E10  
Chapter 5, p. 163 – Section 5.1.4.2 – last paragraph |
| 17-May-2001    | Correction     | Chapter 8, pp. 278-290 – changes to the text and to the example models to reflect the fact that the ODB/ISA relationships inherit behavior only. |
| 25-May-2001    | Minor correction | Chapter 8, pp.279-280 |
Chapter 2  
Underpinnings of Requirements Analysis

p. 44 - Section 2.1.5.2.3

Add a new paragraph at the end of the section:

Regrettably, UML does not support modeling of dynamic or multiple classification. This aligns it with the similar lack of support in programming environments. Consequently, our explanations and examples are not enhanced with graphical models.

p. 57 - Section 2.2.3.2 – 1st paragraph, last sentence

Replace:

(the final state)

With:

(that leads to a final state)

p. 70 - Section 2.2.6.1 – Analysis Tutorial: Step 16, first paragraph, last sentence

Replace:

However, the use case model has not specified when the payment is actually received with relation to the invoice that is to be prepared for the order.

With:

This results in the generation of an order and subsequently in the preparation of an invoice. However, the use case diagram has not clarified when the payment is actually received with relation to the invoice. We can assume, for example, that the payment can be made before or after the invoice is issued and that partial payments are allowed.

p. 72 - Section 2.2.6.2 – Figure 2.38

Replace:

Cancelled (the state name)
**With:**

Canceled (as below)

---

**p. 79 – Question E10**

*Add the following paragraph at the end:*

Provide two solutions. One for the situation in which partial payments are a priori designated as partial. The second for the situation in which the system has to calculate if a payment is partial or in full.
Chapter 4
Requirements Specification

p. 110 - Section 4.2.1.1.3 – 1st paragraph, 1st sentence
Replace:
(by the UML Unified Process, to be precise)

With:
(by the Rational Unified Process (RUP), to be precise)

p. 119 - Section 4.2.1.2.3 – Figure 4.2
Replace:
$number_currently_available in VideoMedium

With:
$percentage_excellent_condition in VideoMedium (as below)

<table>
<thead>
<tr>
<th>MovieTitle</th>
<th>VideoMedium</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;PK&gt;&gt; movie_code : String</td>
<td>video_condition : Byte</td>
</tr>
<tr>
<td>movie_title : String</td>
<td>$ percentage_excellent_condition : Decimal</td>
</tr>
<tr>
<td>director : String</td>
<td></td>
</tr>
<tr>
<td>/ is_in_stock : Boolean</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RentalConditions</th>
<th>VideoTape</th>
</tr>
</thead>
<tbody>
<tr>
<td>rental_period_in_days : Integer</td>
<td></td>
</tr>
<tr>
<td>rental_charge_per_period : Currency</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VideoTape</th>
<th>VideoDisk</th>
</tr>
</thead>
<tbody>
<tr>
<td>BetaTape</td>
<td>VHSTape</td>
</tr>
<tr>
<td>DVDDisk</td>
<td></td>
</tr>
</tbody>
</table>

p. 119 - Section 4.2.1.2.3 – paragraph before Figure 4.2
Replace:
VideoMedium.number_currently_available is a class-scope (static) attribute (Section 2.1.6). MovieTitle.is_in_stock is a derived attribute, i.e. it can be computed by looking at the current value of VideoMedium.number_currently_available.

With:

MovieTitle.is_in_stock is a derived attribute.
VideoMedium.percentage_excellent_condition is a class-scope (static) attribute (Section 2.1.6). This attribute will contain the percentage of VideoMedium objects with the value of the attribute video_condition = “excellent”. Although not shown in the diagram, a class-scope operation (named, for example, $computePercentageExcellentCondition) would need to be associated with that attribute to compute its current value on demand.

Explanation of the above correction:

The class-scope attribute VideoMedium.number_currently_available is not able to serve the purpose stipulated in the textbook. As shown later in Figure 4.7, VideoMedium is an abstract class at the root of the inheritance tree. The concrete classes at the bottom of the inheritance tree will each have its own inherited copy of $number_currently_available. The three copies will share the same value of that attribute and the meaning of that value will be the number of currently available VideoMedium objects (i.e. the combined number of Beta tapes, VHS tapes and DVD disks). Therefore, the derived attribute is_in_stock cannot be computed by looking at the current value of $number_currently_available.

p. 132 - Section 4.2.4.3 – Figure 4.7

Replace:

$number_currently_available in VideoMedium

With:

$percentage_excellent_condition in VideoMedium (as below)
p. 142 - Section 4.3.2.3 – Figure 4.13

Replace:

Figure 4.13

With:
p. 147 – Section 4.3.4.3 – Figure 4.15

Add:
The operation addStudent(stdOID) to the operations compartment in the class Course.
**p. 151 – Question R5**

*Replace:*

Figure 4.2 (Example 4.5)

*With:*

Figure 4.7 (Example 4.10)

*Replace:*

and VideoMedium.number_currently_available as a *static* attribute.

*With:*

and VideoMedium.percentage_excellent_condition as a *static* attribute.

**p. 153 – Exercise Questions – Point 7, 1st sentence**

*Replace:*

In a single rental transaction, a customer can borrow many tapes or disks.

*With:*

A customer can borrow many tapes or disks but each borrowed video medium constitutes separate rental record.

**p. 153 – Exercise Questions – Point 10, 2nd sentence**

*Replace:*

Once two overdue notices on a single tape have been sent…

*With:*

Once two overdue notices on a single tape/disk have been sent…
p. 154 – Exercise Questions – Question E6

Add:

Add the following sentence after the sentence ending with “…(Example 4.14).”

Ignore the last paragraph in the Main Flow section of the table (this paragraph refers rather to the use case Maintain Customer).
p. 163 - Section 5.1.4.2 – Last paragraph

Add:

To be precise, the class in Figures 5.7 and 5.8 should also contain the attribute `date_start` and the operation `computeDuration()` inherited from `Campaign`. These properties should be modeled as `private` in `BonusCampaign`. (Note that `private` properties of the base class remain `private` to the base and are inaccessible to operations of the derived class. However, the definition of a `private` property is inherited in the usual way.)
p. 242 – Exercise Questions – Additional requirements (Video Store) – Requirement 10 – 2nd sentence

Replace:

Once two Overdue Notices on a single tape…

With:

Once two Overdue Notices on a single tape or disk…
p. 278 – Section 8.2.1 – 2nd paragraph from the bottom

Replace:

For consistency with UML, we will keep using the notion of the class in its all-encompassing meaning as both the type and the class (unless stated to the contrary).

With:

The semantics of the type enables separating the specification from its various implementations. The specification of the abstract behavior of a type is called an interface. An interface cannot be directly instantiated.

p. 279 – Section 8.2.1.1 – the bottom of the page

Replace:

A collection literal is a template on literals, i.e. a parameterized type (Section 6.2.2.5) where parameter values are literal values. A collection literal can be:

- set<t> (i.e. a set where all the elements are of the same literal type t; e.g. set<dept_name>, where dept_name is a string)

With:

A collection literal is a parameterized type (Section 6.2.2.5) where formal parameters are usually literal types, but they can be object types. Either way, a collection literal does not have an object identifier. A collection literal can be:

- set<t> (i.e. a set where all the elements are of the same literal type or object type t; e.g. set<dept_name>, where dept_name is a string)

p. 280 – Section 8.2.1.1 – bullet point in the middle of the page

Replace:

collection (the same set of possibilities as for the collection literal but with parameters taking on object values; e.g. set<Dept>, where Dept is a class)

With:
collection (the same set of possibilities as for the collection literal but a collection object has an object identifier; e.g. set<Dept>, where Dept is a class and instances of set<Dept> are objects)

Explanation of the corrections below that relate to ODB examples:

Recent ODMG standard documents clarify that the interface can contain properties and operations but the ISA relationships enforce only inheritance of behavior (i.e. operations). This has necessitated changes in examples in the book in which the ISA relationships were used for the inheritance of both state and behavior.

**p. 282 – Section 8.2.1.3 – Figure 8.4**

Replace:

existing Figure 8.4

With:

an improved version of Figure 8.4 below

![Diagram](image)

**p. 282 – Section 8.2.1.3 – two paragraphs under Figure 8.4**
Replace:

Figure 8.4 shows an extended UML diagram with ISA and EXTENDS relationships. `EmployeeClass` inherits the interface definition from `PersonInterface`. `PersonInterface` is an abstract class. `ManagerClass` inherits the implementation, including the declaration of the method `age`, from `EmployeeClass`.

The ODMG standard uses the keyword `interface` when defining an abstract class, such as `PersonInterface`. The keyword `class` is used only for the definition of concrete classes that can be instantiated, such as `ManagerClass` and `EmployeeClass`.

With:

`Employee` inherits only the behavior from `PersonIN`. The properties of `PersonIN` are not inherited - a copy of `dob` must be explicitly included in `Employee`. `Manager` inherits both the state (declarations of properties) and the behavior (code of operations) from `Employee`.

The ODMG standard permits multiple ISA inheritance of interfaces (i.e. multiple inheritance of behavior). Both interfaces and classes can inherit from interfaces but interfaces cannot inherit from classes. Also, the ISA inheritance is not allowed between classes. Finally, the standard permits only single EXTENDS inheritance of state and behavior.

**p. 286 – Section 8.2.2.1 – Figure 8.5**

Replace:

existing Figure 8.5

With:

a corrected version of Figure 8.5 below
p. 285 – Section 8.2.2.1 – two paragraphs underneath the Example 8.1 frame

Replace:

Figure 8.5 demonstrates the mapping. We introduced two ODB interfaces: PersonShortName and PersonLongName. The first interface defines the data type for contact_name. We specified the type for contact_name explicitly within the class Contact and additionally via the ISA relationship. Strictly speaking, the ISA relationship is redundant here. The class Contact does not want to inherit the two separate attributes from PersonShortName. It only uses PersonShortName as the type for its own attribute contact_name.

PersonLongName inherits the two attributes of PersonShortName via the ISA relationship and together with its own attribute provides the data type for employee_name. As before, the ISA relationship between Employee and PersonLongName provides only for a visual connection, but it is redundant otherwise.

With:

Figure 8.5 demonstrates the mapping. We introduced two ODB abstract classes to represent the types PersonShortName and PersonLongName. The first class defines the type for contact_name within the class Contact. The second class inherits the two attributes of PersonShortName via the EXTENDS relationship and together with its own attribute provides the data type for employee_name within the class Employee.
Replace:

existing Figure 8.6

With:

a corrected version of Figure 8.6 below
p. 286 – Section 8.2.2.2 – 2nd sentence in the last paragraph

Replace:

Apart from the previously introduced ODB interfaces (PersonShortName and PersonLongName), we created an inheritance hierarchy for the ODB interface Address.

With:

Apart from the previously introduced ODB classes (PersonShortName and PersonLongName), we created an EXTENDS inheritance hierarchy for the ODB class Address.

p. 289 – Section 8.2.2.3 – Figure 8.7

Replace:

existing Figure 8.7

With:

a corrected version of Figure 8.7 below
p. 288 – Section 8.2.2.3 – 2nd sentence in the paragraph underneath the Example 8.3 frame

Replace:

Two ODB interfaces are defined: YearSemester and AcademicRecord.

With:

Two abstract ODB classes are defined: YearSemester and AcademicRecord.

p. 288 – Section 8.2.2.4 – 1st sentence in the section

Replace:

Mapping UML generalizations to ODB ISA and EXTENDS relationships is essentially one-to-one. Inheriting from an ODB interface results in an ISA relationship. Inheriting from an ODB class is modeled with an EXTENDS relationship.
Mapping UML generalization relationships is likely to rely predominantly on the ODB EXTENDS relationship. UML has neither a notion of interface (in the ODMG sense), nor a separate category of generalization relationship that would correspond to the ODB ISA relationship. Inheritance of behavior is supported in UML through property visibility (attributes with private visibility are not inherited).

**p. 290 – Section 8.2.2.4 – Figure 8.8**

Replace:

existing Figure 8.8

With:

a corrected version of Figure 8.8 below

![Diagram showing ODB classes and relationships]

**p. 289 – Section 8.2.2.3 – the last two paragraphs**

Replace:

The transformation from the UML to the ODB generalization model is shown in Figure 8.8. Since the generalization hierarchy in Figure 4.7 represented the interface inheritance, the OMG ISA relationships are used.
The UML class RentalConditions is transformed to an ODB interface. This ODB interface is used as a structured object type for the attribute rental_cond embedded in the ODB interface VideoMedium. We decided that RentalConditions is a property of VideoMedium and, as such, it should be embedded in the concrete classes inheriting from VideoMedium (i.e. BetaTape, VHSTape and DVDDisk).

With:

The transformation from the UML to the ODB generalization model is shown in Figure 8.8. The UML class RentalConditions is transformed to an abstract ODB class. The ODB class VideoMedium contains the attribute rental_cond of the type RentalConditions. That attribute, together with other properties of VideoMedium, is inherited down to the concrete classes at the bottom of the inheritance tree (i.e. BetaTape, VHSTape and DVDDisk).