INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Prof. Dr. Dr. h.c. M. Broy
Dr. H. Ehler, S. Wagner

Lösungsblatt 3
9. November 2006

Übungen zu Softwaretechnik

Aufgabe 5 – Datenmodellierung

Für das in Aufgabe 3 beschriebene Informationssystem werden auch Stammdaten der gehandelten Produkte benötigt. Dafür liegen folgende Anforderungen vor: Produkte haben jeweils eine Produktnummer, einen Namen, einen Einkaufs- und einen Verkaufspreis. Die gehandelten Produkte werden von einem Hersteller bestellt, nach Lieferung gelagert und, sofern im Lager vorhanden, weiterverkauft. Da Produkte auch Bauteile anderer Produkte sein können, muss diese Information in den Produktstammdaten enthalten sein.

a) Modellieren Sie geeignete Datenstrukturen mit folgenden Beschreibungsmitteln:

- *Algebraische Spezifikation*
- *Entity/Relationship-Modell*
- *UML-Klassenmodell*
- *Prozedurale Sprache C*
- *Objekt-orientierte Sprache Java*

1. Algebraische Spezifikation

Sorten:

```
sort Product,
sort State,
sort Int,
sort String,
sort Bool
```

Funktionen:

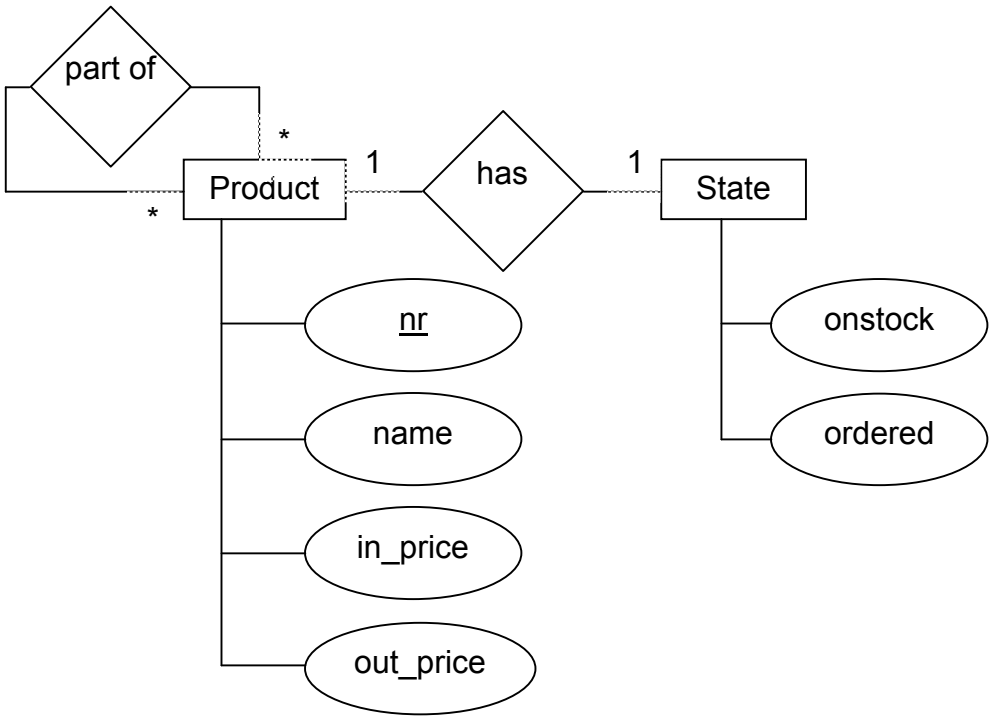
```
prod_nr, prod_in_price, prod_out_price: Product → Int,
prod_name: Product → String,
prod_state: Product → State,
prod_order, prod_decrease_order, prod_incoming, prod_outgoing:
      Product, Int → Product,

onstock: State → Int,
ordered: State → Int,

prod_new: Int, String, Int, Int → Product,
```

```
ispartof: Product, Product → Bool,
new_part: Product, Product → Product
```

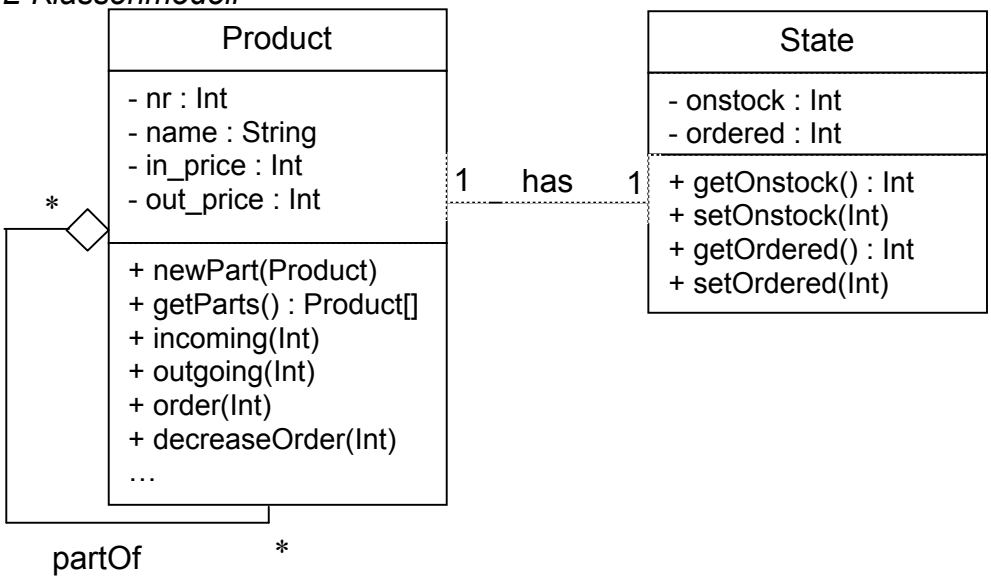
Axiome:

```
prod_nr(prod_new(nr, name, in, out)) = nr,
prod_name(prod_new(nr, name, in, out)) = name,
prod_in_price(prod_new(nr, name, in, out)) = in,
prod_out_price(prod_new(nr, name, in, out)) = out,

onstock(prod_state(prod_new(nr, name, in, out))) = 0,
ordered(prod_state(prod_new(nr, name, in, out))) = 0,

ispartof(prod, prod_new(nr, name, in, out)) = false,
ispartof(prod_new(nr, name, in, out), prod) = false,

(i ≤ ordered(prod_state(prod)) =>
  ordered(prod_state(prod_incoming(prod, i))) =
      ordered(prod_state(prod)) – i,
  onstock(prod_state(prod_incoming(prod, i))) =
      onstock(prod_state(prod)) + i,
  prod_nr(prod_incoming(prod, i)) = prod_nr(prod),
  prod_name(prod_incoming(prod, i)) = prod_name(prod),
  prod_in_price(prod_incoming(prod, i)) = prod_in_price(prod),
  prod_out_price(prod_incoming(prod, i)) = prod_out_price(prod)),

(i ≤ ordered(prod_state(prod)) =>
  ordered(prod_state(prod_outgoing(prod, i))) =
      ordered(prod_state(prod)),
  onstock(prod_state(prod_outgoing(prod, i))) =
      onstock(prod_state(prod)) – i,
  prod_nr(prod_outgoing(prod, i)) = prod_nr(prod),
  prod_name(prod_outgoing(prod, i)) = prod_name(prod),
  prod_in_price(prod_outgoing(prod, i)) = prod_in_price(prod),
  prod_out_price(prod_outgoing(prod, i)) = prod_out_price(prod)),
…
```

2. Entity/Relationship-Modell



3. UML-Klassenmodell



4. Prozedurale Sprache C

Die eigentliche Beschreibung der Signatur ist ohne genaue Beschreibung der Datentyprealisierung möglich, also ohne eine genaue Beschreibung der Definition von `product` durch ein entsprechendes `typedef`-Konstrukt.

```
/* Einbinden der Realisierung der Datentypen */
#include "product.h"

/* Definition des Datentyps state */
int state_onstock(state s);
int state_ordered(state s);

/* Definition des Datentyps product */
product prod_new(int prod_nr, string prod_name,
  int prod_in_price, int prod_out_price);

/* Selektoren */
int prod_nr(product p);
string prod_name(product p);
int prod_in_price(product p);
int prod_out_price(product p);
state prod_state(product p);

/* Weitere Zugriffsfunktionen */
product prod_order(product p, int o);
product prod_incoming(product p, int i);
prodcut prod_outgoing(product p, int o);

/* Kombination von Produkten */
bool is_part_of(product p1, product p2);
product new_part(product p1, product p2);
```

Für die Übersetzung wird jedoch eine Realisierung benötigt (Datei “product.h”).

```
/* Definition von Basistypen */
typedef char *string;
typedef int bool;
```

```
/* Definition des Datentyps state */
typedef struct {
  int onstock; /* Anzahl der Produkte auf Lager */
  int ordered; /* Anzahl der bestellten Produkte */
} *state;

/* Definition des Datentyps product */
typedef struct product_struct {
  int number; /* Stueckzahl pro Produkt */
  string name; /*Name des Produkts */
  int in_price; /* Einkaufspreis des Produkts in Cent */
  int out_price; /* Verkaufspreis des Produkts in Cent */
  state actual_state; /* Aktueller Zustand des Produkts */
  struct product_list {
    struct product_struct *part; /* Produkt in der Liste */
    struct product_list *next; /* Fortsetzung der Liste */
  } parts; /* Liste der Teilprodukte */
} *product;
```

5. Objekt-orientierte Sprache Java

Die Definition ist in Java beispielsweise durch abstrakte Klassen möglich. Für die Implementierung wird dann eine konkrete Realisierung durch eine entsprechende Klasse benötigt.

```
// Definition des Typs Product
public abstract class Product {
  // Attribute
  private int nr;
  private String name;
  private int inPrice;
  private int outPrice;
  private State state;
  private Product[] parts;

  // Methoden
  public abstract Product[] getParts();
  public abstract void incoming(int amount);
```

```
  public abstract void outgoing(int amount);
  public abstract void order(int amount);
  public abstract void decreaseOrder(int amount);
  public abstract boolean isPartOf(Product part);
  // … weitere Getter und Setter
}

// Definition des Typs State
public abstract class State {
  // Attribute
  private int onstock;
  private int ordered;
  // moeglicherweise: private Product product;

  // Methoden
  public abstract int getOnstock();
  public abstract int getOrdered();
  public abstract void setOnstock();
  public abstract void setOrdered();
}
```

b) Bestimmen Sie die Unterschiede im Informationsgehalt dieser verschiedenen Beschreibungsmittel.

E/R-Diagramme: Weisen den geringsten Informationsgehalt auf, da sie nur die Entitäten, deren Bestandteile und die Zusammenhänge zwischen ihnen darstellen können.

UML-Klassenmodelle: Haben gegenüber E/R-Diagrammen zusätzlich explizite Datentypen und Methodensignaturen. Die Angabe von zusätzlichen Konsistenzbedingungen ist durch Benutzung der *Object Constraint Language (OCL)* möglich. Die Beziehungen zwischen den Klassen (Assoziation, Aggregation, Komposition, Vererbung) ist wesentlich differenzierter möglich.

C und **Java:** Können (in der hier dargestellten Form ohne ausprogrammierte Funktionen bzw. Methoden, keine Rümpfe) detailliert die Datentypen, die Zusammenhänge zwischen diesen und die Funktions- bzw. Methodensignaturen darstellen.

algebraische Spezifikationen: Haben den größten Informationsgehalt. Sie beschreiben neben den Datentypen und Funktionssignaturen auch Konsistenzbedingungen und Eigenschaften in abstrakter Form durch Axiome.

c) Diskutieren Sie Vor- und Nachteile der verschiedenen Beschreibungsmittel.

Beschreibungsmittel	Vorteile	Nachteile
Algebraische Spezifikation	- hoher Informationsgehalt - formale Darstellung (Semantik) - abstrakt	- sehr detailliert - schwer verständlich
E/R-Diagramm	- graphische Darstellung - übersichtlich	- keine Funktionen beschreibbar - wenig Konsistenzbedingungen
UML-Klassendiagramm	- graphische Darstellung - übersichtlich - nah an objekt-orientierten Programmiersprachen	- semi-formal
C	- compilierbar - formale Darstellung (Semantik)	- unübersichtlich - implementierungsnah
Java	- compilierbar - formale Darstellung (Semantik) - objekt-orientiert	- unübersichtlich - implementierungsnah