Hausarbeit zum Thema

# Ligaverwaltung

im Rahmen der Lehrveranstaltung

## Entwurf von Informationssystemen

bei Prof. Dr. Martin Gogolla

Sommersemester 2007


| | | |
|---|---|---|
| Edvin Pehlivanović | edvin@tzi.de | 19 17 886 |
| Roman Asendorf | romanase@tzi.de | 19 18 099 |


27. September 2007

# Inhaltsverzeichnis

# 1 Einleitung

Diese Hausarbeit ist im Rahmen der Veranstaltung 'Entwurf von Informationssystemen' im Sommersemester 2007 bei Prof. Dr. Martin Gogolla entstanden.

Ziel und Zweck ist die Anwendung von UML (Unified Modelling Language) und OCL (Object Constraint Language) zur Spezifikation eines Informationssystems.

Die Struktur des Systems wird durch UML festgelegt und ist durch Klassen modelliert. Die Klassen besitzen zur genaueren Definition Attribute und Operationen.

Durch OCL sind weitere Einschränkungen in Form von Invarianten auf Systemzuständen sowie Vor- und Nachbedingungen zu den Operationen definiert.

Zu den Einschränkungen durch Invarianten existiert ein gültiges Beispiel. Dieses ist in einem Objektdiagramm (Abb. 2) auf Seite 19 dargestellt. Die Testfälle beschränken sich auf die Einschränkungen in OCL, basieren jedoch grundsätzlich auf einer gültigen Objektstruktur.

Die Einschränkungen durch Vor- und Nachbedingungen bei Operationen werden anhand eines validen Ablaufs in einem Sequenzdiagramm (Abb. 16) auf Seite 37 präsentiert. Auch hier werden alle Vor- und Nachbedingungen anschließend getestet und ebenfalls durch kleine Sequenzdiagramme begleitet (Abschnitt 5.2).

Die Ausführung der Beispiele und Tests[1] erfolgten hierbei mit dem in der Arbeitsgruppe Datenbanksysteme der Universität Bremen[2] entwickelten Werkzeug USE - UML Based Specification Environment [5].

Alle Abbildungen in diesem Dokument wurden Screenshots aus USE entnommen.

[1] Quelltexte zu den Beispielen und Tests befinden sich im Anhang.
[2] Homepage: http://db.informatik.uni-bremen.de/

# 2 Systembeschreibung

Bei der Ligaverwaltung handelt es sich um ein System, mit dem sich verschiedene Ligen in einer oder mehreren Sportarten verwalten lassen.

Daneben lassen sich Benutzer anlegen, die einzelnen Ligen zugeordnet werden können. Um die Ligen zu verwalten, ist diese Zuordnung jedoch nicht obligatorisch.

Die Verwaltung der Ligen unterliegt einigen Abhängigkeiten. So wird vorausgesetzt, dass einer Liga die Existenz einer Sportart und einer Saison vorausgeht. Eine Liga ist also einer Sportart und einer Saison zugehörig und wird durch einen eindeutigen Namen gekennzeichnet.

Da es innerhalb einer Sportart unterschiedliche Gewinnsätze abhängig von der Liga geben kann, sind die Gewinnsätze in der Liga definiert. Auch die Punkteverteilung für Sieg, Niederlage und Unentschieden werden in der Liga festgelegt.

Der Liga selbst sind die Teams und die Spieltage zugehörig. Die Teams (im weiteren auch als Mannschaften bezeichnet) lassen sich genau einer Liga zuordnen. Um die Teams unterscheiden zu können, werden sie durch einen eindeutigen Namen innerhalb der Liga gekennzeichnet. Wenn zwei Teams in verschiedenen Ligen den gleichen Namen erhalten, handelt es sich um verschiedene Objekte und sie sind dadurch eindeutig festgelegt.

Die Begegnungen der Teams sind in Spieltagen organisiert. Die Anzahl der Spieltage ergibt sich aus $(n - 1) * 2 \mid n = \textit{Anzahl der Teams einer Liga}$, da jedes Team gegen jedes andere Team spielt und es eine Hin- und eine Rückrunde gibt. An jedem Spieltag spielt jedes Team der Liga genau ein Spiel gegen ein anderes Team. Jeder Spieltag erhält dabei eine eindeutige Ordnungsnummer. (Anmerkung: Die Spieltage sind in der Realität nicht zwingend einzelne Kalendertage.)

Demzufolge finden an einem Spieltag $\frac{n}{2} \mid n = \textit{Anzahl der Teams in einer Liga}$ Spiele statt. Jedes Team spielt genau zweimal gegen jedes andere Team in einer Saison. Die Spiele eines Spieltages sind ebenfalls mit eindeutigen Ordnungsnummern versehen.

Ein Spiel selbst setzt sich aus einem Heim- und einem Gastteam zusammen. Sofern das Spiel noch keine Ergebnisse erzielt hat, sind dem Spiel noch keine Sätze zugeordnet. Ist das Spiel beendet, so müssen diesem Spiel $\geq n \leq (n * 2) - 1 \mid n = \textit{einer Liga definierte Gewinnsätze}$ Sätze zugeordnet sein. Auch die Sätze sind mit eindeutigen Ordnungsnummern versehen. Die Sätze enthalten schließlich die Punkte (Tore), die Heim- bzw. Gastteam erzielt haben.

# 3 Klassendiagramm

Das Klassendiagramm in Abbildung 1 zeigt die Struktur einer Ligaverwaltung. Die Klasse `Liga` kann als eine Hauptklasse angesehen werden. Sie beschreibt die wichtigste Einheit in diesem System.

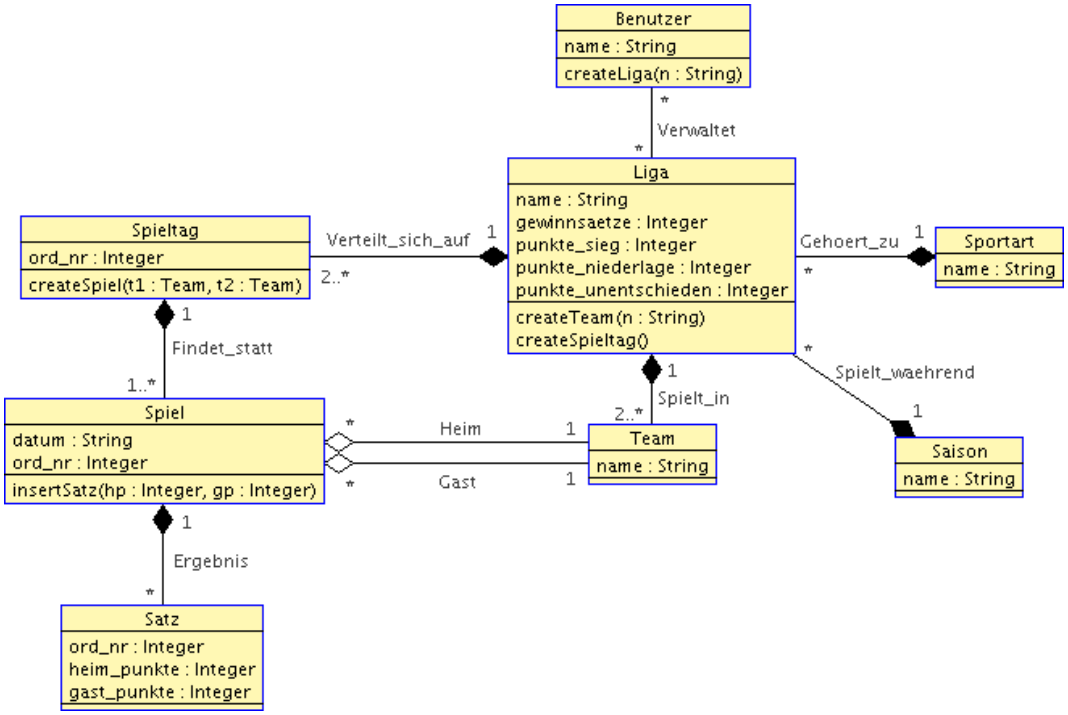


Abbildung 1: Klassendiagramm

Jede Liga gehört zu genau einer Sportart und spielt während genau einer Saison. Die Ligen können von Benutzern verwaltet werden. In dieser Hinsicht gibt es keine Einschränkungen, was bedeutet, dass ein Benutzer keine oder auch mehrere Ligen verwalten kann. Im Gegenzug bedeutet das auch, dass eine Liga von keinem oder beliebig vielen Benutzern verwaltet werden darf.

In jeder Liga spielen mindestens zwei Teams. Die Begegnungen der Teams in Spielen finden an bestimmten Spieltagen statt. Jedes Spiel gehört dabei zu genau einem Spieltag und die Begegnung in einem Spiel ist durch genau zwei Teams definiert, einem Heim- und einem Gastteam. Zu jedem Spiel gehören Ergebnisse, die in Form von Sätzen beschrieben werden.

Alle Spieltage gehören zu genau einer Liga. Da eine Liga mindestens zwei Teams haben muss (weniger Teams in einer Liga hätten keinen Sinn), gibt es auch mindestens zwei Spieltage in jeder Liga. Die obere Schranke der Kardinalität kann in diesem Klassendiagramm nur mit 'beliebig viele' angegeben werden. Eine genauere Einschränkung kann nur mit zusätzlichen Sprachmitteln gegeben werden. Diese

Einschränkungen werden im Abschnitt 3.3 beschrieben.

## 3.1 Attribute und Operationen

Dieser Abschnitt beschreibt die Attribute und Operationen der Klassen.

- Klasse `Benutzer`

  **name** Der Name des Benutzers. Dieses Attribut muss für alle Instanzen der Klasse eindeutig sein. Datentyp: `String`.

  **createLiga(...)** Operation zum Anlegen einer Liga, die von dem Benutzer verwaltet wird.
  Parameter:

  **n : String** Der Name der neuen Liga.

  **gs : Integer** Anzahl der Gewinnsätze in der neuen Liga.

  **ps : Integer** Punkte für ein gewonnenes Spiel.

  **pn : Integer** Punkte für ein verlorenes Spiel.

  **pu : Integer** Punkte für ein unentschiedenes Spiel.

  **sa : Sportart** Sportart, zu der die Liga gehört.

  **s : Saison** Saison, während der die Liga spielt.

- Klasse `Sportart`

  **name** Der Name der Sportart. Dieses Attribut muss für alle Instanzen der Klasse eindeutig sein. Datentyp: `String`.

- Klasse `Saison`

  **name** Der Name der Saison. Dieses Attribut muss für alle Instanzen der Klasse eindeutig sein. Datentyp: `String`.

- Klasse `Liga`

  **name** Der Name der Liga. Dieses Attribut muss für alle Instanzen der Klasse, die zur selben Sportart und selben Saison gehören, eindeutig sein. Datentyp: `String`.

  **gewinnsaetze** Anzahl der Gewinnsätze, die in dieser Liga gespielt werden, z.B. in der Fußball-Bundesliga wird auf einen Gewinnsatz gespielt, im Volleyball i.d.R. auf drei Gewinnsätze. Datentyp: `Integer`.

  **punkte_sieg** Beschreibt die Anzahl der Punkte, die es für einen Sieg im Spiel gibt. Datentyp: `Integer`.

  **punkte_niederlage** Beschreibt die Anzahl der Punkte, die es für eine Niederlage im Spiel gibt. Datentyp: `Integer`.

**punkte_unentschieden** Beschreibt die Anzahl der Punkte, die es für ein Unentschieden im Spiel gibt. Datentyp: `Integer`.

**createTeam(n : String)** Mit dieser Operation können Mannschaften für eine Liga erstellt werden. Der Parameter ist der Name der neuen Mannschaft.

**createSpieltag()** Diese Operation erstellt einen neuen Spieltag für die Liga.

- Klasse `Team`

  **name** Der Name des Teams. Dieses Attribut muss für alle Instanzen der Klasse in derselben Liga eindeutig sein. Datentyp: `String`.

- Klasse `Spieltag`

  **ord_nr** Ordnungsnummer des Spieltages. Die Spieltage einer Liga sind von 1 bis $n$ geordnet. Datentyp: `Integer`.

  **createSpiel(h : Team, g : Team)** Für den Spieltag wird ein neues Spiel erstellt, in welchem die beiden übergebenen Teams gegeneinander spielen.

- Klasse `Spiel`

  **datum** Datum, an welchem das Spiel stattfindet. Datentyp: `String`.

  **ord_nr** Ordnungsnummer des Spiels. Die Spiele an einem Spieltag sind von 1 bis $n$ geordnet. Datentyp: `Integer`.

  **insertSatz(hp : Integer, gp : Integer)** Diese Operation erstellt einen Satz für die Ergebnisse des Spiels. Die Parameter beschreiben die Punkte der Heim- bzw. Gastmannschaft.

- Klasse `Satz`

  **ord_nr** Ordnungsnummer des Satzes. Die Sätze eines Spiels sind von 1 bis $n$ geordnet. Datentyp: `Integer`.

  **heim_punkte** Anzahl der Punkte der Heimmannschaft. Datentyp: `Integer`.

  **gast_punkte** Anzahl der Punkte der Gastmannschaft. Datentyp: `Integer`.

## 3.2 Assoziationen

Im Folgenden werden die Assoziationen beschrieben und erläutert.

Die Assoziation `Verwaltet` zwischen den Klassen `Benutzer` und `Liga` legt fest, dass ein Benutzer keine bis beliebig viele Ligen verwalten kann und eine Liga von keinem oder beliebig vielen Benutzern verwaltet werden kann. Zwischen den beiden Klassen existiert keine Teil-von-Relation, so dass hier eine Assoziation ausreicht.

```
association Verwaltet between
  Benutzer[0..*]
  Liga[0..*]
end
```

Die Komposition `Gehoert_zu` zwischen den Klassen `Sportart` und `Liga` legt fest, dass eine Sportart keine bis beliebig viele Ligen enthalten kann und eine Liga genau einer Sportart zugehörig sein muss. Ligen existieren solange wie auch die dazugehörige Sportart.

```
composition Gehoert_zu between
  Sportart[1]
  Liga[0..*]
end
```

Die Komposition `Spielt_waehrend` zwischen den Klassen `Saison` und `Liga` legt fest, dass eine Liga solange existiert wie auch die dazugehörige Saison. Außerdem kann jede Saison keine bis beliebig viele Ligen enthalten und eine Liga muss während einer Saison spielen.

```
composition Spielt_waehrend between
  Saison[1]
  Liga[0..*]
end
```

Die Komposition `Spielt_in` zwischen den Klassen `Liga` und `Team` legt fest, dass in einer Liga zwischen zwei und beliebig vielen Teams spielen. Ein Team kann jedoch nur in einer Liga spielen. Teams existieren solange wie auch die dazugehörige Liga existiert.

```
composition Spielt_in between
  Liga[1]
  Team[2..*]
end
```

Die Komposition `Verteilt_sich_auf` zwischen den Klassen `Liga` und `Spieltag` legt fest, dass sich eine Liga auf zwei bis beliebig viele Spieltage verteilt. Ein Spieltag kann jedoch nur einer Liga zugehörig sein und existiert solange wie auch die dazugehörige Liga existiert.

```
composition Verteilt_sich_auf between
  Liga[1]
  Spieltag[2..*]
end
```

Die Komposition `Findet_statt` zwischen den Klassen `Spieltag` und `Spiel` legt fest, dass an einem Spieltag zwischen einem und beliebig vielen Spielen stattfinden. Ein Spiel muss genau einem Spieltag zugehörig sein und existiert solange wie auch der dazugehörige Spieltag existiert.

```
composition Findet_statt between
  Spieltag[1]
  Spiel[1..*]
end
```

Die Komposition `Ergebnis` zwischen den Klassen `Spiel` und `Satz` legt fest, dass ein Spiel keinen bis beliebig viele Sätze enthalten darf. Ein Satz muss genau einem Spiel zugehörig sein und existiert solange wie auch das dazugehörige Spiel existiert.

```
composition Ergebnis between
  Spiel[1]
  Satz[0..*]
end
```

Die Aggregation `Heim` zwischen den Klassen `Spiel` und `Team` legt fest, dass ein Spiel in der Rolle `Heimspiel` genau ein Team in der Rolle `Heimteam` enthält. Ein Team in der Rolle `Heimteam` kann keinem oder beliebig vielen Spielen in der Rolle `Heimspiel` zugehörig sein.

```
aggregation Heim between
  Spiel[0..*] role heimspiel
  Team[1] role heimteam
end
```

Die Aggregation `Gast` zwischen den Klassen `Spiel` und `Team` legt fest, dass ein Spiel in der Rolle `Gastspiel` genau ein Team in der Rolle `Gastteam` enthält. Ein Team in der Rolle `Gastteam` kann keinem oder beliebig vielen Spielen in der Rolle `Gastspiel` zugehörig sein.

```
aggregation Gast between
  Spiel[0..*] role gastspiel
  Team[1] role gastteam
end
```

## 3.3 Invarianten und Vor- und Nachbedingungen

Die folgenden Invarianten (inkl. der Vor- und Nachbedingungen von Operationen) beschreiben weitere Einschränkungen, denen die Ligaverwaltung unterworfen ist und die nicht bereits durch das Klassendiagramm (Abb. 1) gegeben sind. Alle Invarianten sowie die Vor- und Nachbedingungen sind in einem bestimmten Kontext definiert.

- Kontext `Benutzer`

  Der Name eines Benutzers muss eindeutig sein. Zwei unterschiedliche Instanzen der Klasse `Benutzer` müssen unterschiedliche Werte im Attribut `name` haben.

```
-- Name des Benutzers ist eindeutig
context Benutzer inv EindeutigerBenutzer:
  Benutzer.allInstances->forAll(b1, b2 |
    b1 <> b2 implies b1.name <> b2.name)
```

  Der Name eines Benutzers muss definiert sein. Das Attribut `name` muss einen Wert haben.

```
-- Name des Benutzers ist definiert
context Benutzer inv DefinierterBenutzer:
  self.name.isDefined()
```

  Die Vorbedingungen der Operation `createLiga` sorgen dafür, dass keine andere Liga mit demselben Namen existiert. Außerdem werden die Parameter für Gewinnsätze, Punkte für Sieg, Niederlage und Unentschieden auf gültige Werte überprüft. Die Nachbedingungen stellen sicher, dass genau ein neues Objekt vom Typ `Liga` erstellt wurde.

```
-- Vor-/Nachbedingungen beim Anlegen einer Liga: Es darf keine
-- Liga mit dem als Parameter uebergebenen Namen existieren; Die
-- Anzahl der Gewinnsaetze muss groesser 0 sein, sowie Punkte
-- fuer Sieg, Niederlage und Unentschieden muessen uebergeben
-- werden; Nach dem Anlegen muss genau ein neues Objekt
-- existieren;
context Benutzer::createLiga(n  : String,  gs : Integer,
                             ps : Integer, pn : Integer,
                             pu : Integer, sa : Sportart,
                             s  : Saison)
  pre  NameNichtVorhanden         : Liga.allInstances
                                      ->forAll(l | l.name <> n)
  pre  GewinnsaetzeGueltig        : gs > 0
  pre  SiegPunkteGueltig          : ps >= 0
  pre  NiederlagePunkteGueltig    : pn >= 0
  pre  UnentschiedenPunkteGueltig : pu >= 0
  post NurEinObjektMehr           :
       (Liga.allInstances - Liga.allInstances@pre)->size() = 1
  post NeuesObjekt                :
       (Liga.allInstances - Liga.allInstances@pre)
       ->forAll(l | l.oclIsNew())
```

- Kontext `Sportart`

  Die Bezeichnung einer Sportart muss eindeutig sein. Zwei unterschiedliche Instanzen der Klasse `Sportart` müssen unterschiedliche Werte im Attribut `name` haben.

```
-- Name der Sportart ist eindeutig
context Sportart inv EindeutigeSportart:
  Sportart.allInstances->forAll(sa1, sa2 |
    sa1 <> sa2 implies sa1.name <> sa2.name)
```

  Die Bezeichnung einer Sportart muss definiert sein. Das Attribut `name` muss einen Wert haben.

```
-- Name der Sportart ist definiert
context Sportart inv DefinierteSportart:
  self.name.isDefined()
```

  Der Name einer Liga muss in einer Saison, innerhalb einer Sportart, eindeutig sein. Somit müssen zwei unterschiedliche Instanzen der Klasse `Liga`, die in derselben Sportart und derselben Saison definiert sind, unterschiedliche Werte im Attribut `name` haben.

```
-- Name der Liga in einer Saison, innerhalb einer Sportart,
-- ist eindeutig
context Sportart inv EindeutigeLiga:
  self.liga->forAll(l1, l2 |
    l1 <> l2 and l1.saison=l2.saison implies l1.name <> l2.name)
```

- Kontext `Saison`

  Die Bezeichnung einer Saison muss eindeutig sein. Zwei unterschiedliche Instanzen der Klasse `Saison` müssen unterschiedliche Werte im Attribut `name` haben.

```
-- Name der Saison ist eindeutig
context Saison inv EindeutigeSaison:
  Saison.allInstances->forAll(s1, s2 |
    s1 <> s2 implies s1.name <> s2.name)
```

  Die Bezeichnung einer Saison muss definiert sein. Das Attribut `name` muss einen Wert haben.

```
-- Name der Saison ist definiert
context Saison inv DefinierteSaison:
  self.name.isDefined()
```

- Kontext `Liga`

  Die Anzahl der Spieltage in einer Liga ergibt sich aus der Anzahl der Teams, die in dieser Liga definiert sind. Bei $n$ Teams müssen $(n-1) * 2$ Spieltage definiert sein.

```
-- In der Liga sind bei n Teams genau (n-1) * 2 Spieltage
-- definiert.
context Liga inv AnzahlSpieltage:
  (self.team->size() - 1) * 2 = self.spieltag->size()
```

  Der Name einer Liga muss definiert sein. Das Attribut `name` muss einen Wert haben.

```
-- Name der Liga ist definiert
context Liga inv DefinierteLiga:
  self.name.isDefined()
```

  Der Name einer Mannschaft muss innerhalb einer Liga eindeutig sein. Zwei unterschiedliche Instanzen der Klasse `Team`, die in einer Liga definiert sind, müssen unterschiedliche Werte im Attribut `name` haben.

```
-- Name der Mannschaft innerhalb einer Liga ist eindeutig
context Liga inv EindeutigesTeam:
  self.team->forAll(t1, t2 |
    t1 <> t2 implies t1.name <> t2.name)
```

  Alle Spieltage einer Liga sind geordnet und tragen im Attribut `ord_nr` einen Wert zwischen 1 und $n$, wobei $n$ die Anzahl der Spieltage darstellt. Jede Ordnungsnummer ist genau einmal vergeben.

```
-- Ordnungsnummern der Spieltage gehen von 1..n
context Liga inv SpieltagsOrdnung:
  let st = self.spieltag->collect(ord_nr)->asSet()
           ->asSequence()->sortedBy(i|i)
  in
    st->size()  = self.spieltag->size() and
    st->first() = 1                     and
    st->last()  = self.spieltag->size()
```

  Die Mannschaften in einer Liga spielen genau zweimal gegeneinander. Einmal in der Hinrunde und einmal in der Rückrunde.

```
-- Jedes Team einer Liga hat genau zwei Begegnungen gegen jedes
-- andere Team
```

```
context Liga inv ZweiSpiele:
  self.team->forAll(t1 | self.team->forAll (t2 |
   t1 <> t2 implies
      self.spieltag.spiel->collect(Set{heimteam, gastteam})
                          ->count(Set{t1,t2}) = 2))
```

Die Operation `createTeam` erstellt ein neues Team in der Liga. Die Vorbedingung stellt sicher, dass kein Team mit dem übergebenen Namen bereits existiert. Die Nachbedingungen sorgen dafür, dass genau ein neues Team in der Liga erzeugt wurde.

```
-- Vor-/Nachbedingungen beim Anlegen eines Teams: Ein Team mit
-- dem als Parameter uebergebenen Namen darf noch nicht
-- existieren; Nach Anlegen des Teams muss genau ein neues
-- Objekt Team existieren
context Liga::createTeam(n : String)
  pre  NameNichtVorhanden : self.team->forAll(t | t.name <> n)
  post NurEinObjektMehr   : (self.team -
                              self.team@pre)->size() = 1
  post NeuesObjekt        : (self.team - self.team@pre)
                             ->forAll(t | t.oclIsNew())
```

Die Operation `createSpieltag` erzeugt einen neuen Spieltag in der Liga. Die Vorbedingung stellt sicher, dass noch nicht alle Spieltage erstellt worden sind. Dafür überprüft sie die Anzahl der bisherigen Spieltage. Die Nachbedingungen stellen sicher, dass genau ein neuer Spieltag in der Liga erstellt wurde. Außerdem muss die vergebene Ordnungsnummer des neuen Spieltags passen.

```
-- Vor-/Nachbedingungen beim Anlegen eines Spieltags: Es darf
-- noch nicht die maximale Anzahl an Spieltagen fuer die Liga
-- erreicht sein; Nach Anlegen des Spieltags muss genau ein
-- neues Objekt Spieltag existieren; Das neue Objekt Spieltag
-- muss die hoechste Ordnungsnummer als Attribut besitzen.
context Liga::createSpieltag()
  pre  MaxAnzahl        : self.spieltag->size() <
                          (self.team->size() - 1) * 2
  post NurEinObjektMehr : (self.spieltag -
                           self.spieltag@pre)->size() = 1
  post NeuesObjekt      : (self.spieltag - self.spieltag@pre)
                           ->forAll(st | st.oclIsNew())
  post OrdnungsnummerOK : self.spieltag->size() =
                     (self.spieltag - self.spieltag@pre)
                     ->collect(ord_nr)->asSequence()->first()
```

- Kontext `Team`

  In einer Liga mit $n$ Teams, spielt jede Mannschaft genau $(n-1) * 2$ Mal.

```
-- Jede Mannschaft spielt, bei n Teams in der Liga,
-- genau (n-1) * 2 Mal
context Team inv AnzahlSpieleProTeam:
  (self.liga.team->size() - 1) * 2 =
  Set{self.heimspiel, self.gastspiel}->flatten()->size()
```

  Die Spiele einer Mannschaft finden an unterschiedlichen Spieltagen statt.

```
-- Jede Mannschaft spielt nur einmal an einem Spieltag
context Team inv TeamEinmalProSpieltag:
  Set{self.heimspiel, self.gastspiel}->flatten()
    ->forAll(s1, s2 | s1 <> s2 implies
             s1.spieltag <> s2.spieltag)
```

  Der Name einer Mannschaft muss definiert sein. Das Attribut `name` muss einen Wert haben.

```
-- Name der Mannschaft ist definiert
context Team inv DefiniertesTeam:
  self.name.isDefined()
```

- Kontext `Spieltag`

  An jedem Spieltag einer Liga spielen alle Mannschaften dieser Liga.

```
-- An jedem Spieltag spielt jede Mannschaft
context Spieltag inv JedesTeamSpielt:
  Set{self.spiel.heimteam, self.spiel.gastteam}->flatten()
  ->includesAll(self.liga.team)
```

  Bei $n$ Mannschaften in einer Liga gibt es genau $n/2$ Spiele an jedem Spieltag.

```
-- Bei n Teams in einer Liga gibt es genau n/2 Spiele
-- pro Spieltag
context Spieltag inv AnzahlSpieleProSpieltag:
  self.liga.team->size() / 2 = self.spiel->size()
```

  Alle Spiele eines Spieltags sind geordnet und tragen im Attribut `ord_nr` einen Wert zwischen 1 und $n$, wobei $n$ die Anzahl der Spiele am Spieltag darstellt. Jede Ordnungsnummer ist genau einmal vergeben.

```
-- Ordnungsnummern der Spiele gehen von 1..n
context Spieltag inv SpieleOrdnung:
  let st = self.spiel->collect(ord_nr)->asSet()
           ->asSequence()->sortedBy(i|i)
  in
    st->size()  = self.spiel->size() and
    st->first() = 1                  and
    st->last()  = self.spiel->size()
```

Die Operation `createSpiel` erzeugt eine neues Spiel innerhalb des Spieltags. Durch die Vorbedingungen wird sichergestellt, dass ein neues Spiel erzeugt werden darf und die Mannschaften aus derselben Liga wie der Spieltag kommen. Die Nachbedingungen stellen sicher, dass genau ein neues Spiel erstellt wurde und dass die Ordnungsnummer des Spiels stimmt.

```
-- Vor-/Nachbedingungen beim Anlegen eines Spiels: Es darf noch
-- nicht die maximale Anzahl an Spielen fuer den Spieltag
-- erreicht sein; Die uebergebenen Teams muessen aus der selben
-- Liga wie der Spieltag kommen; Nach Anlegen des Spiels muss
-- genau ein neues Objekt Spiel existieren; Das neue Objekt
-- Spiel muss die hoechste Ordnungsnummer als Attribut besitzen.
context Spieltag::createSpiel(h: Team, g : Team)
  pre  MaxAnzahl           : self.spiel->size() <
                             self.liga.team->size() / 2
  pre  HeimInSpieltagsLiga : self.liga = h.liga
  pre  GastInSpieltagsLiga : self.liga = g.liga
  post NurEinObjektMehr    : (self.spiel -
                              self.spiel@pre)->size() = 1
  post NeuesObjekt         : (self.spiel - self.spiel@pre)
                             ->forAll(s | s.oclIsNew())
  post OrdnungsnummerOK    : self.spiel->size() =
                        (self.spiel - self.spiel@pre)
                        ->collect(ord_nr)->asSequence()->first()
```

- Kontext `Spiel`

  In jedem Spiel dürfen nur Mannschaften gegeneinander spielen, die in der selben Liga definiert sind.

```
-- In einem Spiel koennen sich nur Teams der selben
-- Liga begegnen
context Spiel inv TeamsInSelberLiga:
  self.heimteam.liga = self.gastteam.liga
```

  In jedem Spiel spielen zwei Mannschaften gegeneinander. Diese Mannschaften müssen unterschiedlich sein, d.h. keine Mannschaft darf gegen sich selber spielen.

```
-- Ein Team darf nicht gegen sich selber spielen
context Spiel inv HeimUngleichGast:
  self.heimteam <> self.gastteam
```

In jeder Liga wird durch das Attribut `gewinnsaetze` die Anzahl der Sätze definiert, die jedes Spiel haben kann. Die Anzahl kann sich dabei zwischen `Liga.gewinnsaetze` und `Liga.gewinnsaetze * 2 - 1` bewegen. Wenn ein Spiel noch nicht gespielt wurde, dann existieren noch keine dazugehörigen Sätze. Somit kann die Anzahl von Sätzen auch Null sein.

```
-- Anzahl der Saetze in einem Spiel zwischen Liga.gewinnsaetze
-- und Liga.gewinnsaetze * 2 - 1 oder keine.
context Spiel inv AnzahlSaetze:
  let saetze : Integer = self.satz->size()
  in
    saetze >= self.spieltag.liga.gewinnsaetze and
    saetze <= self.spieltag.liga.gewinnsaetze * 2 - 1 or
    saetze = 0
```

Alle Sätze eines Spiels sind geordnet und tragen im Attribut `ord_nr` einen Wert zwischen 1 und $n$, wobei $n$ die Anzahl der Sätze im Spiel darstellt. Jede Ordnungsnummer ist genau einmal vergeben.

```
-- Ordnungsnummern der Saetze gehen von 1..n,
-- wenn es Saetze gibt.
context Spiel inv SaetzeOrdnung:
  let st = self.satz->collect(ord_nr)->asSet()
           ->asSequence()->sortedBy(i|i)
  in
    st->size()  = 0 or
    st->size()  = self.satz->size() and
    st->first() = 1                 and
    st->last()  = self.satz->size()
```

Die Operation `insertSatz` fügt das Ergebnis des Spiels ein. Die Vorbedingungen stellen sicher, dass die Anzahl an möglichen Sätzen noch nicht erschöpft wurde. Desweiteren werden die Werte der Parameter `hp` und `gp` überprüft. Die Nachbedingungen stellen sicher, dass genau ein neuer Satz erzeugt wurde und dass dieser Satz die richtige Ordnungsnummer trägt.

```
-- Vor-/Nachbedingungen beim Hinzufuegen eines Satzes: Es darf
-- noch nicht die maximale Anzahl Saetze fuer das Spiel erreicht
-- sein; Nach Anlegen des Satzes muss genau ein neues Objekt
-- Satz existieren; Das neue Objekt Satz muss die hoechste
```

```
-- Ordnungsnummer als Attribut besitzen.
context Spiel::insertSatz(hp : Integer, gp : Integer)
  pre  MaxAnzahl         : self.satz->size() <
                           self.spieltag.liga.gewinnsaetze *2 -1
  pre  HeimPunkteGueltig : hp >= 0
  pre  GastPunkteGueltig : gp >= 0
  post NurEinObjektMehr  : (self.satz -
                            self.satz@pre)->size() = 1
  post NeuesObjekt       : (self.satz - self.satz@pre)
                           ->forAll(s | s.oclIsNew())
  post OrdnungsnummerOK  : self.satz->size() =
                          (self.satz - self.satz@pre)
                          ->collect(ord_nr)->asSequence()->first()
```

- Kontext `Satz`

  Die Punkte in einem Satz müssen größer gleich Null sein.

```
-- Die Punkte/Tore muessen groesser gleich 0 sein.
context Satz inv GueltigePunkte:
  self.heim_punkte >= 0 and self.gast_punkte >= 0
```

# 4 Objektdiagramme

In diesem Abschnitt wird das System anhand von Objektdiagrammen dargestellt. Zunächst wird ein gültiger Systemzustand angegeben. Anschließend werden ungültige Systemzustände mit verschiedenen Szenarios durch Objektdiagramme beschrieben. Mit Hilfe dieser Szenarios werden die Invarianten aus Abschnitt 3.3 getestet.

## 4.1 Gültiges Objektdiagramm

Das Objektdiagramm in Abbildung 2 stellt einen gültigen Systemzustand dar. Hier wird ein kleines Beispiel einer Fußball-Bundesliga aufgezeigt, in welcher vier Mannschaften (Werder, Bayern, HSV und Schalke) gegeneinander antreten. Das entsprechende Skript ist im Anhang A.2 angegeben.

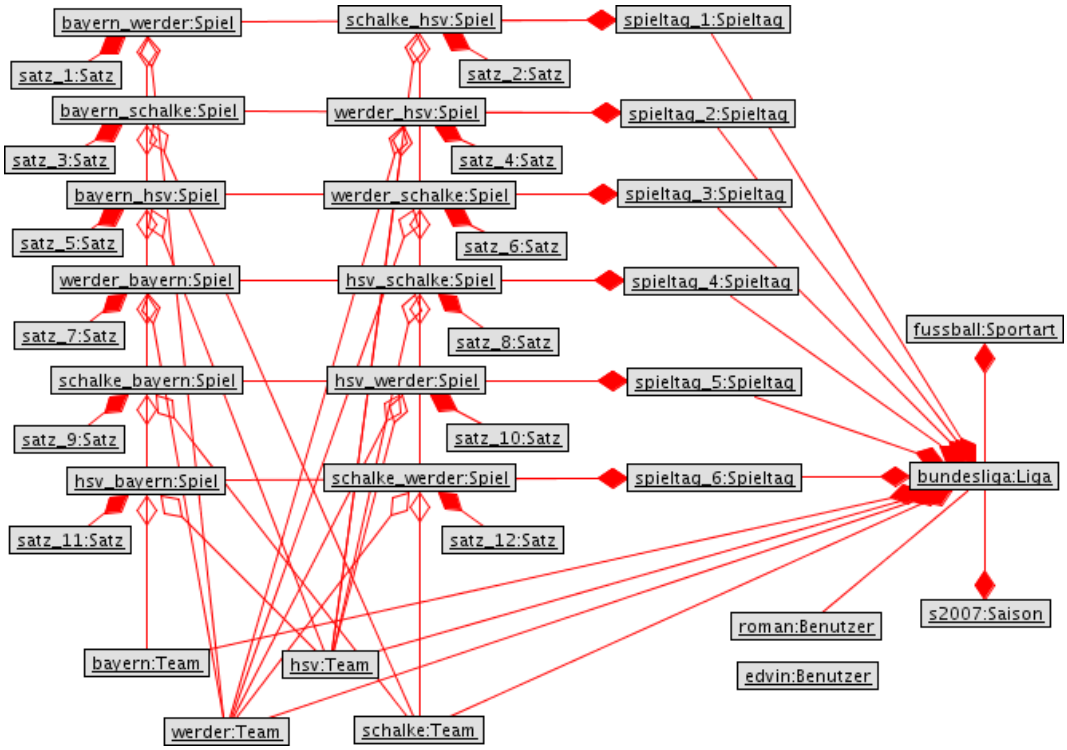


Abbildung 2: Objektdiagramm

Sowohl die Struktur des Klassendiagramms als auch die Beschränkungen durch die Invarianten sind erfüllt (siehe Abb. 3).

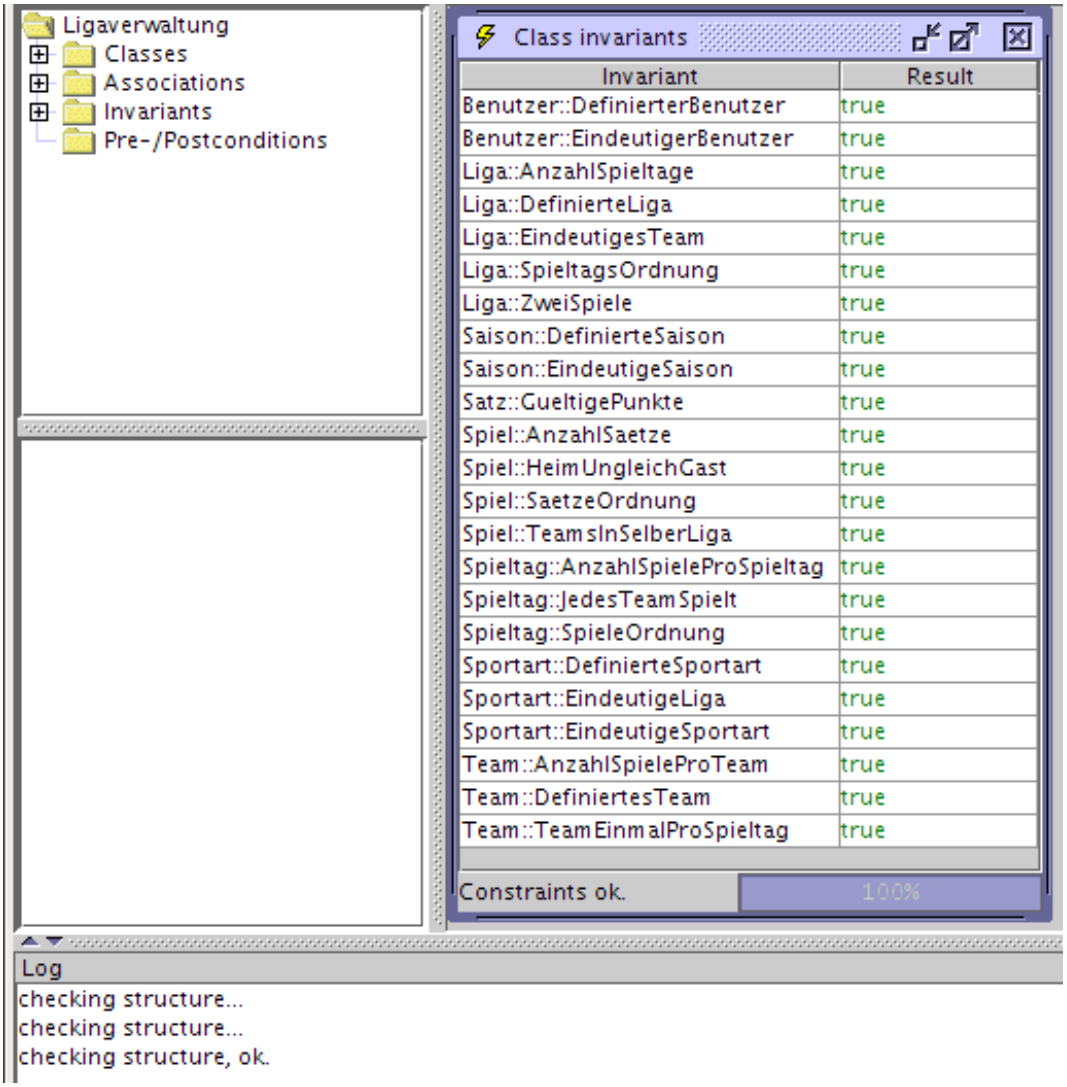


Abbildung 3: Auswertung der Struktur und Invarianten im gültigen Systemzustand

## 4.2 Ungültige Objektdiagramme

Im Folgenden werden einige ungültige Systemzustände gezeigt und durch die definierten Invarianten bestätigt. Ausgang für alle Szenarios ist die Vorlage aus Anhang A.3. Diese Vorlage stellt einen weiteren gültigen Systemzustand dar (Abb. 4), welcher dem jeweiligen Szenario entsprechend modifiziert wird.

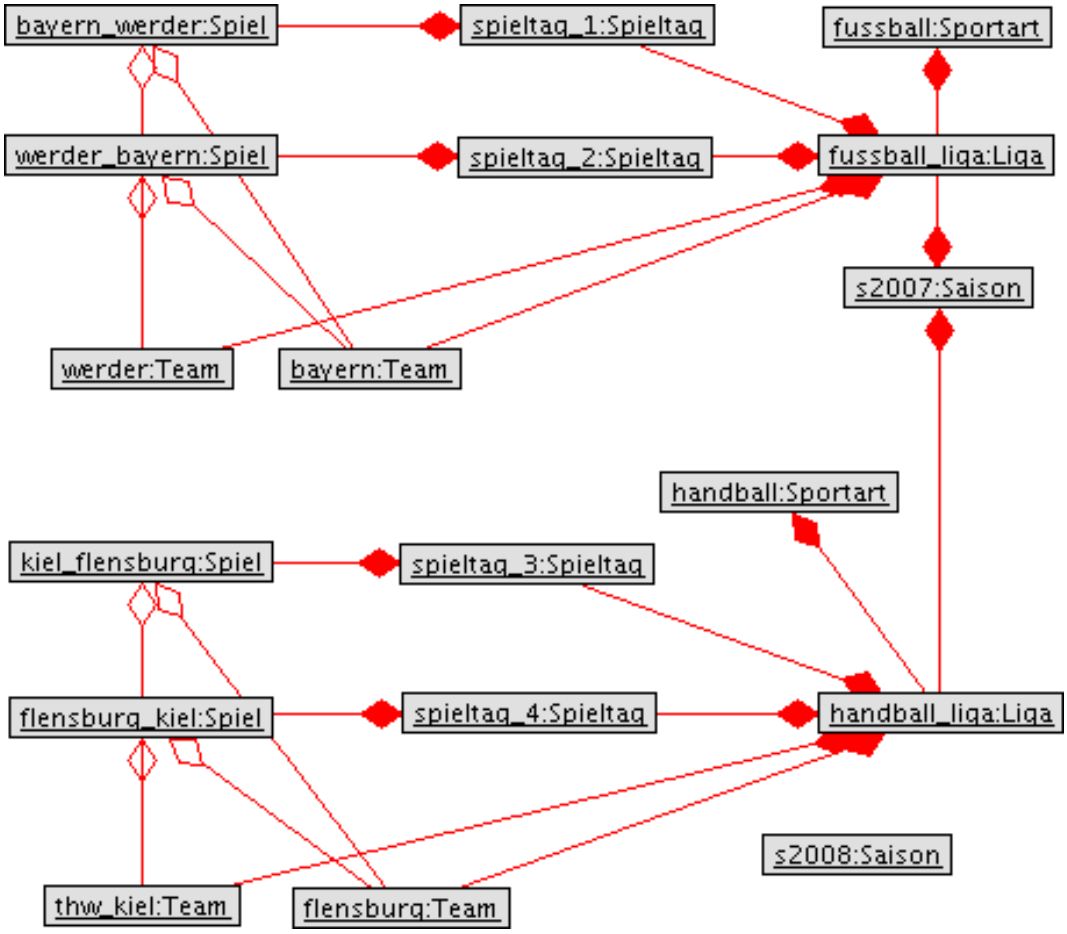


Abbildung 4: Objektdiagramm der Vorlage

### 4.2.1 Szenario: Undefinierte und gleiche Benutzernamen

Im ersten Szenario werden drei Benutzer erstellt, von denen zwei denselben Namen haben und der dritte Benutzername ist undefiniert.

```
1   open -q liga_template.cmd
2
3   -- Testen der Namen von Benutzern
```

```
4
5   !create benutzer1 : Benutzer
6   !create benutzer2 : Benutzer
7   !create benutzer3 : Benutzer
8
9   -- Fehlersituation:
10  -- 1. Test der Invariante Benutzer::EindeutigerBenutzer
11  --    benutzer1 und benutzer2 haben denselben Namen
12  -- 2. Test der Invariante Benutzer::DefinierterBenutzer
13  --    benutzer3 hat keinen Namen
14
15  !set benutzer1.name := 'Max'
16  !set benutzer2.name := benutzer1.name
```

In Abbildung 5 ist zu sehen, dass die beiden Invarianten `Benutzer::Definierter-Benutzer` und `Benutzer::EindeutigerBenutzer` als unwahr ausgewertet werden. Auf der rechten Seite werden die Details der fehlgeschlagenen Invarianten dargestellt.

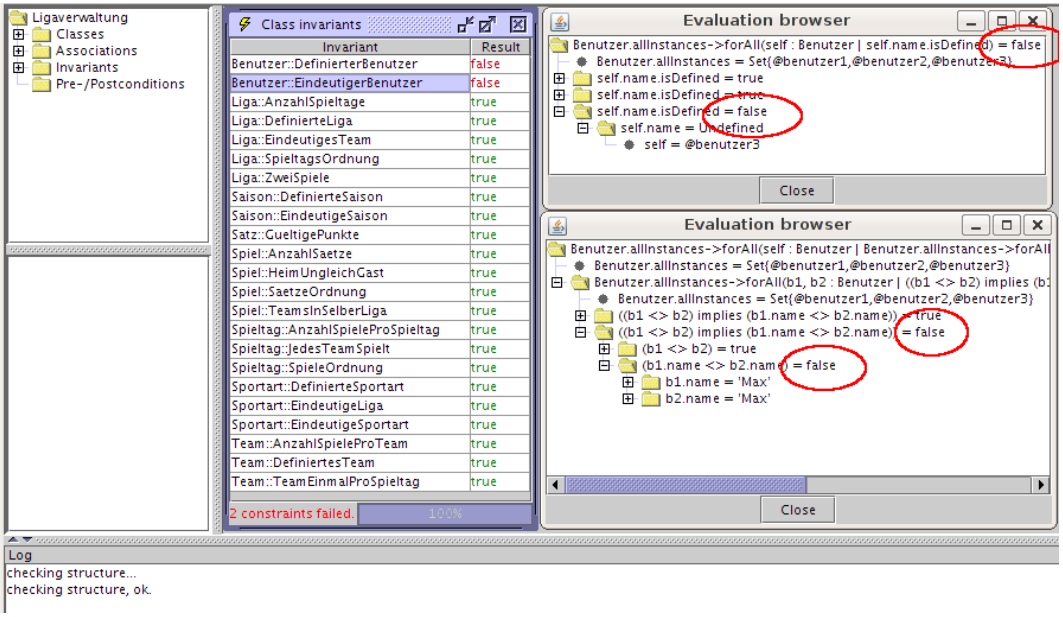


Abbildung 5: Test der Benutzernamen

### 4.2.2 Szenario: Undefinierte und gleiche Bezeichnung von Sportarten

In diesem Szenario werden die Bezeichnungen von Sportarten getestet. Es wird eine neue Instanz einer Sportart erzeugt (`volleyball`) ohne eine Bezeichnung zu vergeben. Außerdem wird den bereits definierten Instanzen der Klasse `Sportart` (`fussball` und `handball`) dieselbe Bezeichnung zugewiesen.

```
1   open -q liga_template.cmd
2
3   -- Testen der Namen von Sportarten
4
5   !create volleyball : Sportart
6
7   -- Fehlersituation:
8   -- 1. Test der Invariante Sportart::EindeutigeSportart
9   --    fussball und handball haben denselben Namen
10  -- 2. Test der Invariante Sportart::DefinierteSportart
11  --    volleyball hat keinen Namen
12
13  !set fussball.name := handball.name
```

In Abbildung 6 ist zu sehen, dass die beiden Invarianten `Sportart::Definierte-Sportart` und `Sportart::EindeutigeSportart` als unwahr ausgewertet werden. Auf der rechten Seite werden die Details der fehlgeschlagenen Invarianten dargestellt.

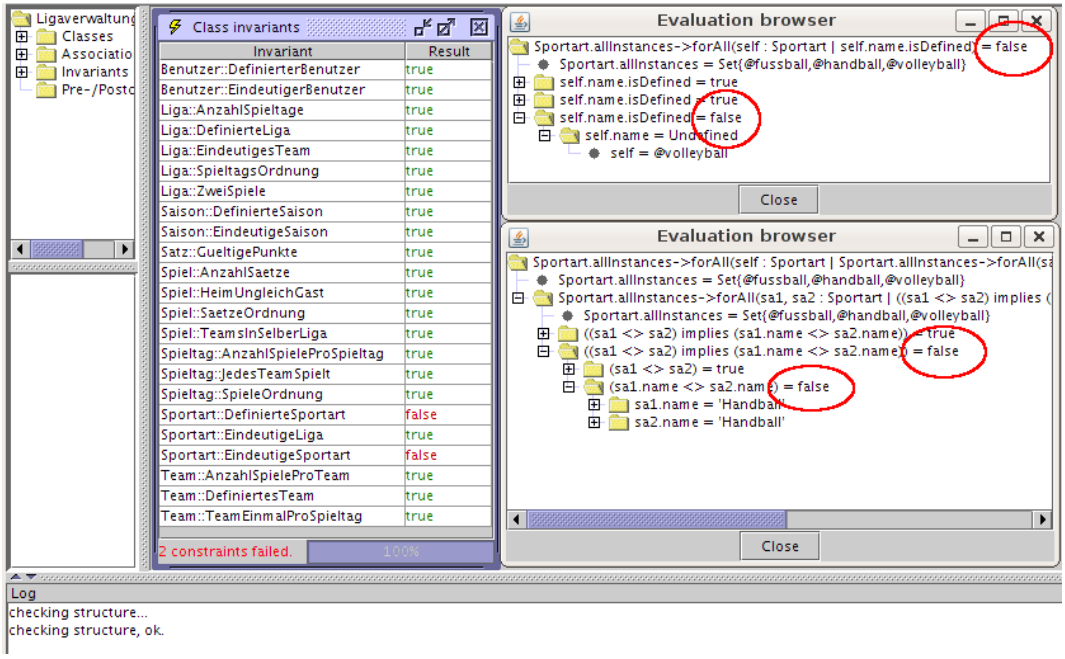


Abbildung 6: Test der Bezeichnungen von Sportarten

### 4.2.3 Szenario: Undefinierte und gleiche Bezeichnung von Ligen

In diesem Szenario werden die Bezeichnungen von Ligen getestet. Da beide Tests gleichzeitig ausgeführt werden, muss für die neue Instanz einer Liga (Objekt `bundes-`

liga) eine kleine Struktur an weiteren Instanzen verschiedener Klassen geschaffen werden, so dass das neue Objekt der Klassenstruktur genügt.

```
1   open -q liga_template.cmd
2
3   -- Testen der Namen von Ligen
4
5   -- Fehlersituation:
6   -- 1. Test der Invariante Sportart::EindeutigeLiga
7   --    fussball_liga und handball_liga haben denselben Namen
8   --    und spielen waehrend der selben Saison und gehoeren
9   --    zur selben Sportart
10  -- 2. Test der Invariante Liga::DefinierteLiga
11  --    bundesliga hat keinen Namen
12
13  -- 1. Test
14  !set handball_liga.name := fussball_liga.name
15  !delete (handball, handball_liga) from Gehoert_zu
16  !insert (fussball, handball_liga) into Gehoert_zu
17
18  -- 2. Test
19  !create bundesliga : Liga
20  !set bundesliga.gewinnsaetze := 1
21  !insert (fussball, bundesliga) into Gehoert_zu
22  !insert (s2008, bundesliga) into Spielt_waehrend
23
24  !create stuttgart : Team
25  !insert (bundesliga, stuttgart) into Spielt_in
26  !set stuttgart.name := 'VfB Stuttgart'
27
28  !create frankfurt : Team
29  !insert (bundesliga, frankfurt) into Spielt_in
30  !set frankfurt.name := 'Eintracht Frankfurt'
31
32  !create spieltag_5 : Spieltag
33  !insert (bundesliga, spieltag_5) into Verteilt_sich_auf
34  !set spieltag_5.ord_nr := 1
35
36  !create spieltag_6 : Spieltag
37  !insert (bundesliga, spieltag_6) into Verteilt_sich_auf
38  !set spieltag_6.ord_nr := 2
39
40  !create stuttgart_frankfurt : Spiel
41  !insert (spieltag_5, stuttgart_frankfurt) into Findet_statt
```

```
42  !insert (stuttgart_frankfurt, stuttgart) into Heim
43  !insert (stuttgart_frankfurt, frankfurt) into Gast
44  !set stuttgart_frankfurt.ord_nr := 1
45
46  !create frankfurt_stuttgart : Spiel
47  !insert (spieltag_6, frankfurt_stuttgart) into Findet_statt
48  !insert (frankfurt_stuttgart, frankfurt) into Heim
49  !insert (frankfurt_stuttgart, stuttgart) into Gast
50  !set frankfurt_stuttgart.ord_nr := 1
```

In Abbildung 7 ist zu sehen, dass die beiden Invarianten `Liga::DefinierteLiga` und `Sportart::EindeutigeLiga` als unwahr ausgewertet werden. Auf der rechten Seite werden die Details der fehlgeschlagenen Invarianten dargestellt.

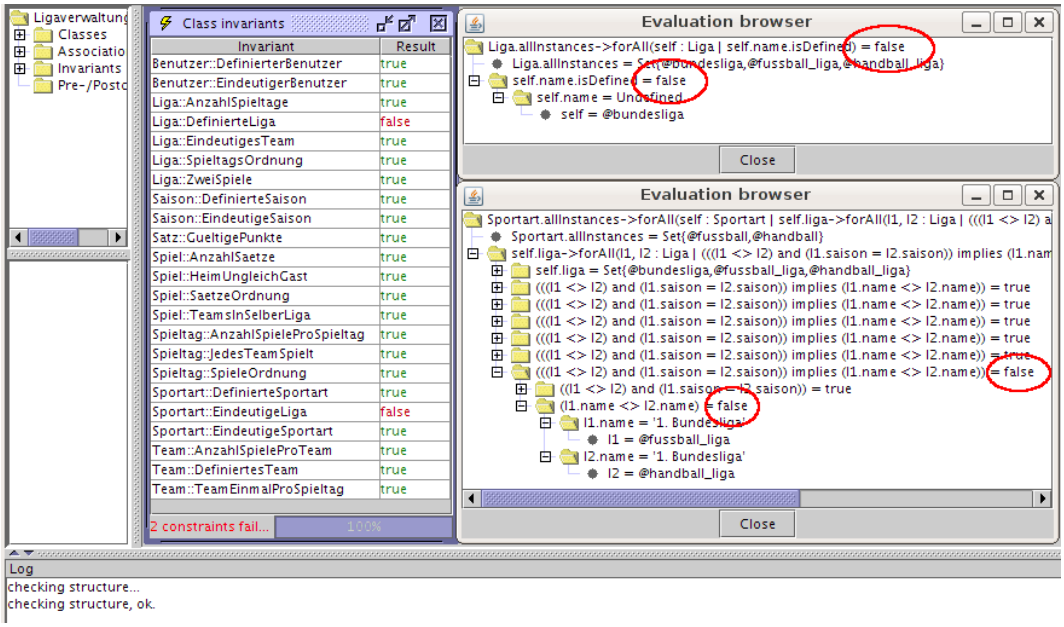


Abbildung 7: Test der Bezeichnungen von Ligen

### 4.2.4 Szenario: Undefinierte und gleiche Bezeichnung von Saisons und Teams

Auf weitere ähnliche Szenarios, in denen die Namen bzw. Bezeichnungen von Instanzen überprüft werden, wird hier verzichtet. Sie wären vom Aufbau her den gezeigten Tests sehr ähnlich. Dies betrifft die Invarianten `Saison::EindeutigeSaison`, `Saison::DefinierteSaison`, `Liga::EindeutigesTeam` und `Team::DefiniertesTeam`.

### 4.2.5 Szenario: Falsche Anzahl von Spieltagen in einer Liga

Dieses Szenario testet die richtige Anzahl an Spieltagen in jeder Liga. Es wird ein weiterer Spieltag (inkl. eines Spiels) erzeugt und zum Objekt `fussball_liga` assoziiert.

```
1   open -q liga_template.cmd
2
3   -- Testen der Anzahl von Spieltagen in den Ligen
4
5   -- Fehlersituation:
6   -- Test der Invariante Liga::AnzahlSpieltage
7   --   Die Liga fussball_liga hat einen Spieltag zu viel
8
9   !create spieltag_5 : Spieltag
10  !insert (fussball_liga, spieltag_5) into Verteilt_sich_auf
11  !set spieltag_5.ord_nr := 3
12
13  !create bayern_werder2 : Spiel
14  !insert (spieltag_5, bayern_werder2) into Findet_statt
15  !insert (bayern_werder2, bayern) into Heim
16  !insert (bayern_werder2, werder) into Gast
17  !set bayern_werder2.ord_nr := 1
```

In Abbildung 8 ist zu sehen, dass die Invariante `Liga::AnzahlSpieltage` als unwahr ausgewertet wird. Auf der rechten Seite werden die Details der fehlgeschlagenen Invariante dargestellt.

### 4.2.6 Szenario: Falsche Anzahl von Spielen bzw. Sätzen

Die Invarianten `Spiel::AnzahlSaetze`, `Team::AnzahlSpieleProTeam` und `Spieltag::AnzahlSpieleProSpieltag` sind der Invariante `Liga::AnzahlSpieltage` ähnlich und werden daher nicht gesondert getestet (siehe Abschnitt 4.2.5).

### 4.2.7 Szenario: Falsche Ordnungsnummern von Spieltagen in einer Liga

Alle Spieltage einer Liga sind aufsteigend von 1 bis $n$ durchnummeriert. Dieses Szenario testet dies, indem einem Spieltag eine falsche Ordnungsnummer zugewiesen wird.

```
1   open -q liga_template.cmd
2
3   -- Testen der Ordnungsnummern von Spieltagen in den Ligen
```

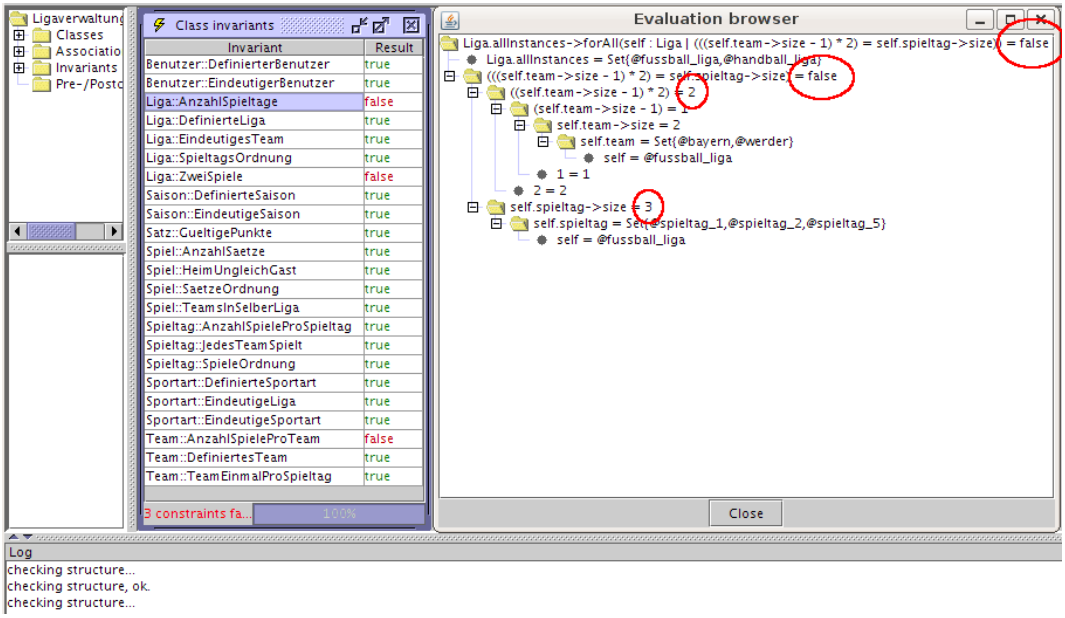


Abbildung 8: Test der Anzahl von Spieltagen in einer Liga

```
4
5   -- Fehlersituation:
6   -- Test der Invariante Liga::SpieltagsOrdnung
7   --   Das Objekt spieltag_4 hat die falsche Ordnungsnummer
8
9   !set spieltag_4.ord_nr := 3
```

In Abbildung 9 ist zu sehen, dass die Invariante `Liga::SpieltagsOrdnung` als unwahr ausgewertet wird. Auf der rechten Seite werden die Details der fehlgeschlagenen Invariante dargestellt.

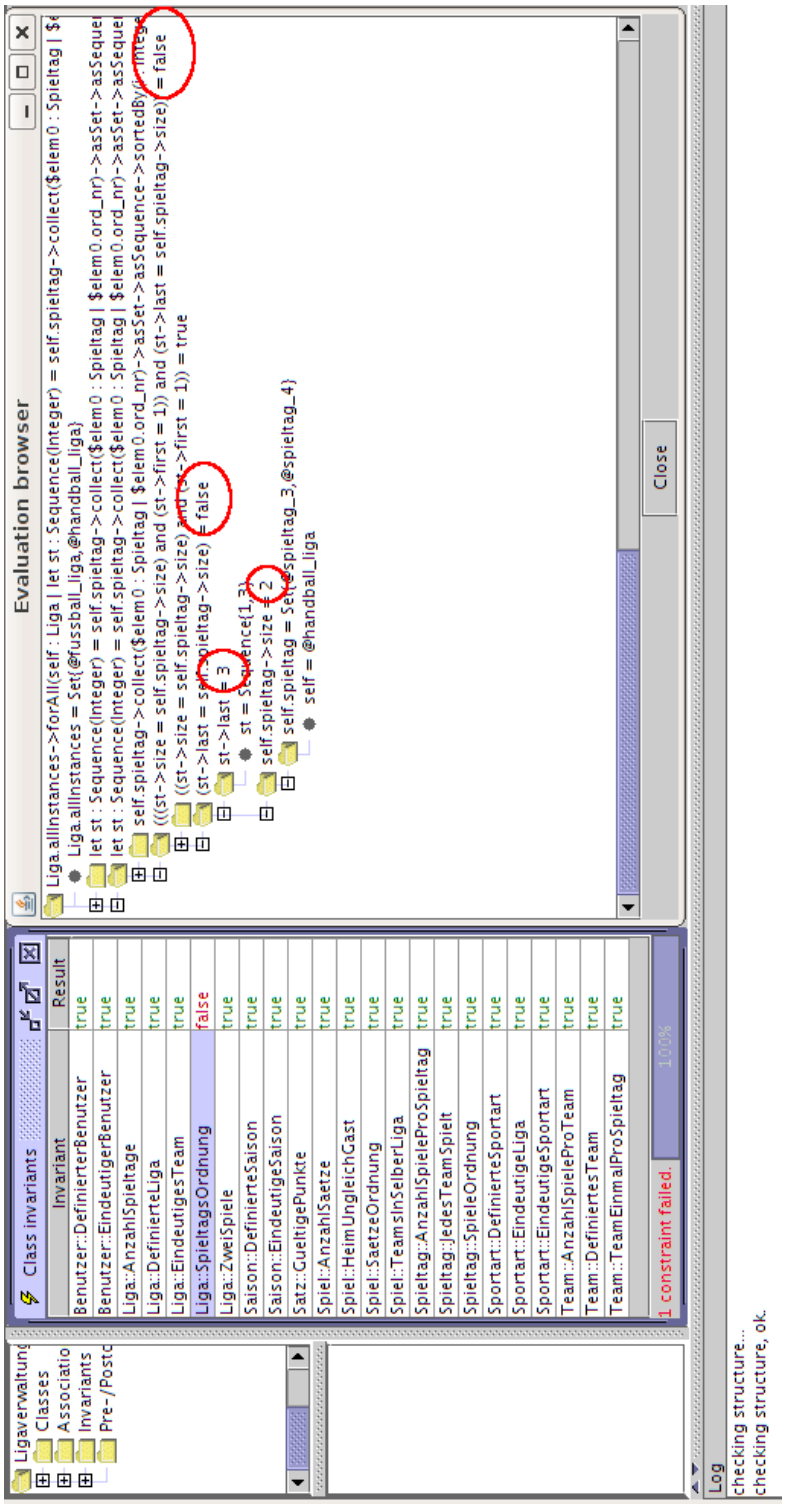


Abbildung 9: Test der Ordnungsnummern von Spieltagen in einer Liga

### 4.2.8 Szenario: Falsche Ordnungsnummern von Spielen und Sätzen

Die Invarianten `Spieltag::SpieleOrdnung` und `Spiel::SaetzeOrdnung` werden nicht explizit getestet, da sie sehr ähnlich zum vorherigen Szenario (Abschnitt 4.2.7) sind.

### 4.2.9 Szenario: Falsche Anzahl von Begegnungen zweier Teams

Dieses Szenario testet, ob jede Mannschaft einer Liga genau zwei Mal gegen jede andere Mannschaft aus derselben Liga spielt.

```
1  open -q liga_template.cmd
2
3  -- Testen, ob jede Mannschaft genau zwei Spiele gegen
4  -- jede andere Mannschaft einer Liga spiel.
5
6  -- Fehlersituation:
7  -- Test der Invariante Liga::ZweiSpiele
8  --   Das Spiel Werder gegen Bayern wird ein
9  --   weiteres mal ausgetragen.
10
11 !create werder_bayern2 : Spiel
12 !insert (spieltag_2, werder_bayern2) into Findet_statt
13 !insert (werder_bayern2, werder) into Heim
14 !insert (werder_bayern2, bayern) into Gast
15 !set werder_bayern2.ord_nr := 1
```

In Abbildung 10 ist zu sehen, dass die Invariante `Liga::ZweiSpiele` als unwahr ausgewertet wird. Auf der rechten Seite werden die Details der fehlgeschlagenen Invariante dargestellt.

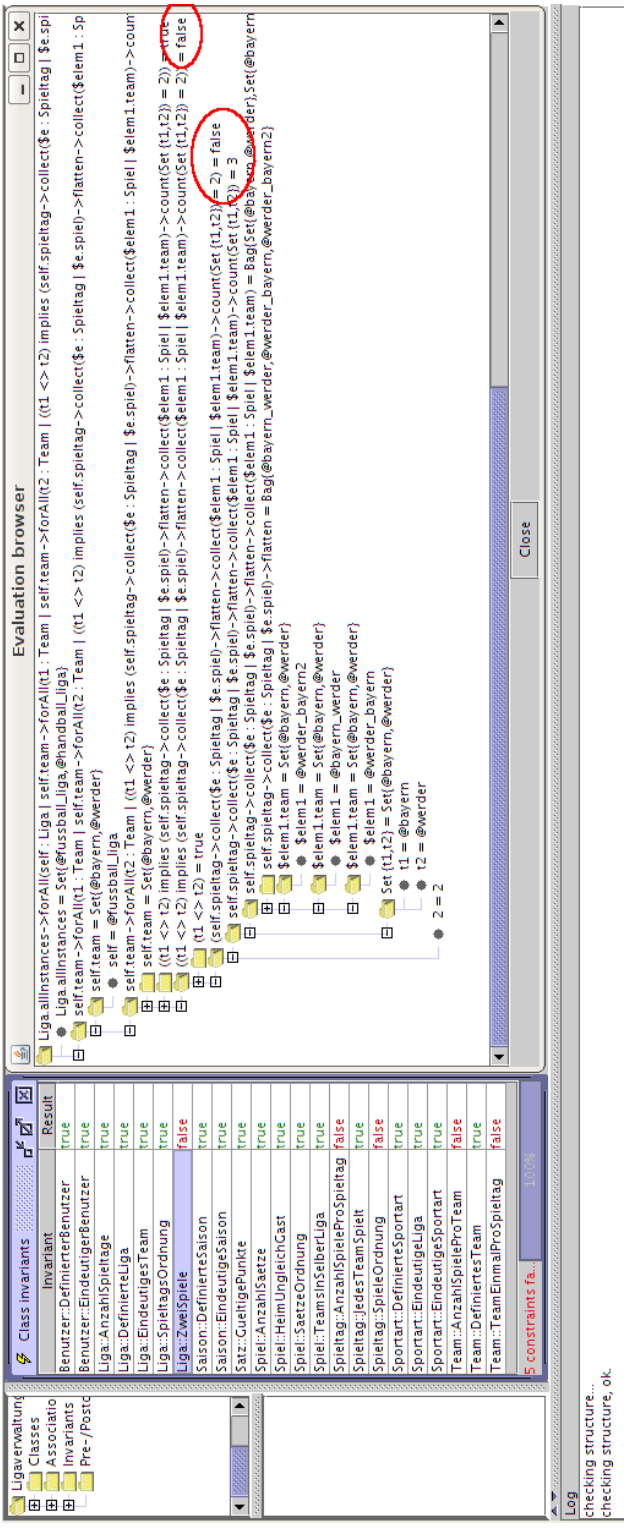


Abbildung 10: Test der Anzahl von Begegnungen zweier Teams

### 4.2.10 Szenario: Ungültige Punkte/Tore

Dieses Szenario testet das Ergebnis eines Spiels. Die erzielten Punkte/Tore dürfen nicht negativ sein.

```
1   open -q liga_template.cmd
2
3   -- Testen, ob die Punkte/Tore in Saetzen gueltig sind
4
5   -- Fehlersituation:
6   -- Test der Invariante Satz::GueltigePunkte
7   --   Die Gastmannschaft hat eine negative Anzahl
8   --   an Toren geschossen.
9
10  !create satz : Satz
11  !insert (werder_bayern, satz) into Ergebnis
12  !set satz.ord_nr := 1
13  !set satz.heim_punkte := 4
14  !set satz.gast_punkte := -1
```

In Abbildung 11 ist zu sehen, dass die Invariante `Satz::GueltigePunkte` als unwahr ausgewertet wird. Auf der rechten Seite werden die Details der fehlgeschlagenen Invariante dargestellt.

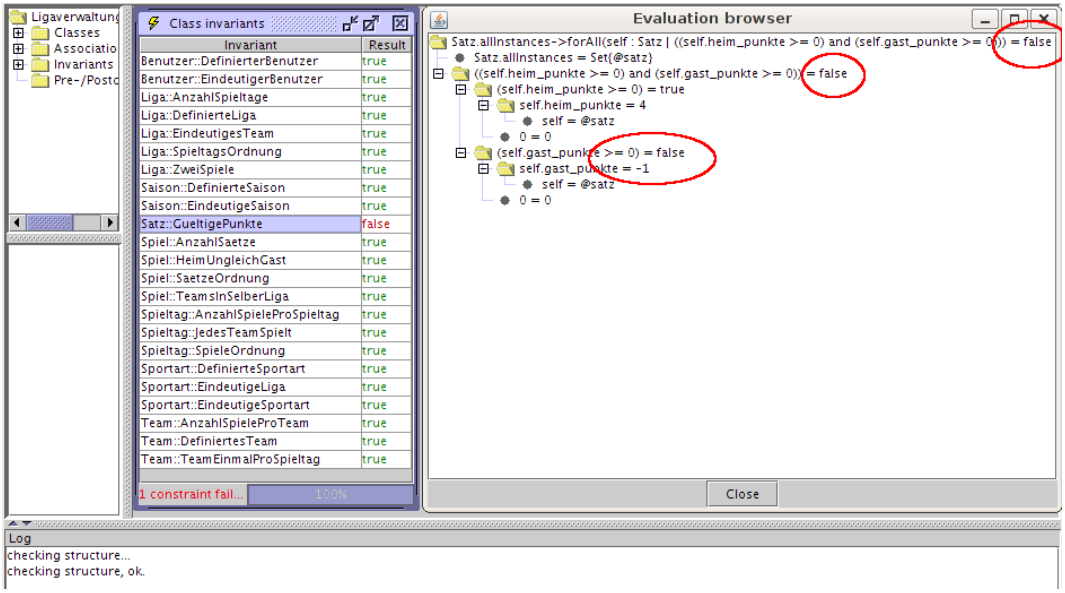


Abbildung 11: Test der Punkte/Tore auf Gültigkeit

### 4.2.11 Szenario: Heim- und Gastmannschaft sind identisch

Dieses Szenario überprüft, dass die Invariante `Spiel::HeimUngleichGast` eine Spielbegegnung verhindert, in der die Heim- und Gastmannschaft identisch sind.

```
1   open -q liga_template.cmd
2
3   -- Testen, ob die Heim- und Gastmannschaft identisch sind.
4
5   -- Fehlersituation:
6   -- Test der Invariante Spiel::HeimUngleichGast
7   --   Die Heim- und Gastmannschaft sind identisch.
8
9   !delete (bayern_werder, werder) from Gast
10  !insert (bayern_werder, bayern) into Gast
```

In Abbildung 12 ist zu sehen, dass die Invariante `Spiel::HeimUngleichGast` als unwahr ausgewertet wird. Auf der rechten Seite werden die Details der fehlgeschlagenen Invariante dargestellt.

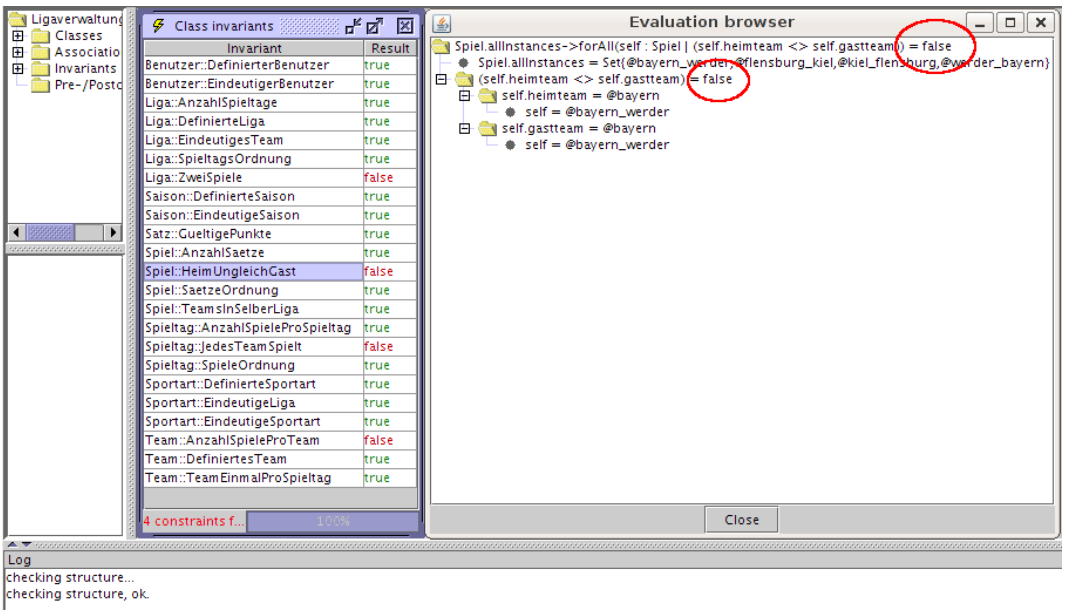


Abbildung 12: Test der Spielmannschaften auf Gleichheit

### 4.2.12 Szenario: Liga der Heim- und Gastmannschaft ist nicht dieselbe

Das Szenario überprüft, ob die Invariante `Spiel::TeamsInSelberLiga` den ungültigen Systemzustand erkennt, in welchem die Mannschaften einer Spielbegegnung aus

verschiedenen Ligen kommen.

```
1   open -q liga_template.cmd
2
3   -- Testen, ob die Heim- und Gastmannschaft in der
4   -- selben Liga sind.
5
6   -- Fehlersituation:
7   -- Test der Invariante Spiel::TeamsInSelberLiga
8   --   Die Heim- und Gastmannschaft kommen aus verschiedenen Ligen.
9
10  !delete (bayern_werder, werder) from Gast
11  !insert (bayern_werder, thw_kiel) into Gast
```

In Abbildung 13 ist zu sehen, dass die Invariante `Spiel::TeamsInSelberLiga` als unwahr ausgewertet wird. Auf der rechten Seite werden die Details der fehlgeschlagenen Invariante dargestellt.

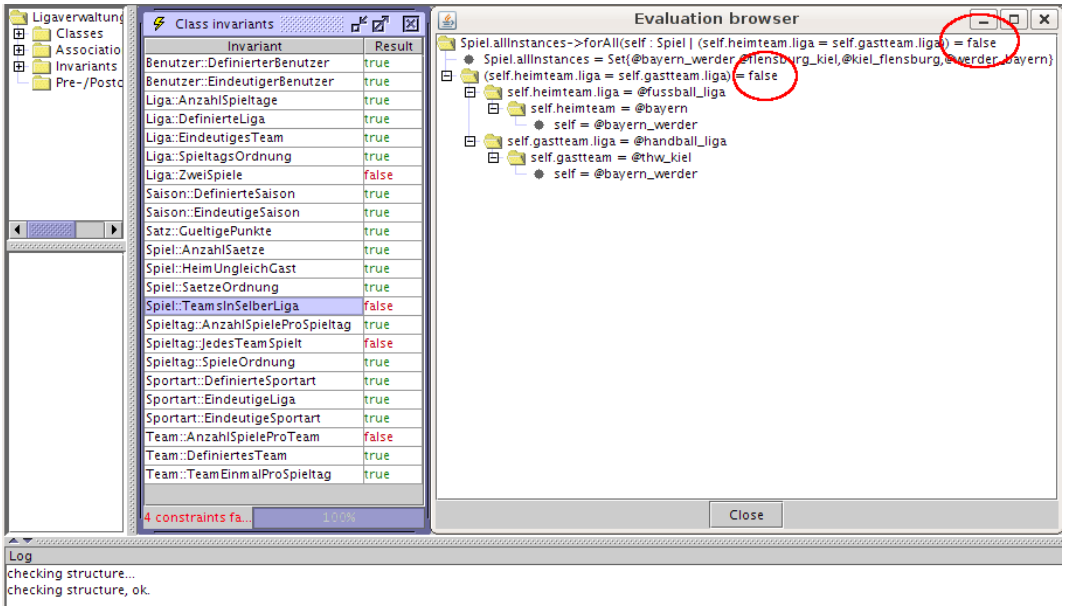


Abbildung 13: Test der Spielmannschaften auf Gleichheit der Ligen

### 4.2.13 Szenario: Nicht alle Teams einer Liga spielen mit

Alle Teams einer Liga müssen auch spielen. Diesen Umstand überwacht die Invariante `Spieltag::JedesTeamSpielt`.

```
1   open -q liga_template.cmd
```

```
2
3   -- Testen, ob alle Teams einer Liga spielen.
4
5   -- Fehlersituation:
6   -- Test der Invariante Spieltag::JedesTeamSpielt
7   --   Die Mannschaft lemgo spielt nie.
8
9   !create lemgo : Team
10  !insert (handball_liga, lemgo) into Spielt_in
11  !set lemgo.name := 'Lemgo'
```

In Abbildung 14 ist zu sehen, dass die Invariante `Spieltag::JedesTeamSpielt` als unwahr ausgewertet wird. Auf der rechten Seite werden die Details der fehlgeschlagenen Invariante dargestellt.

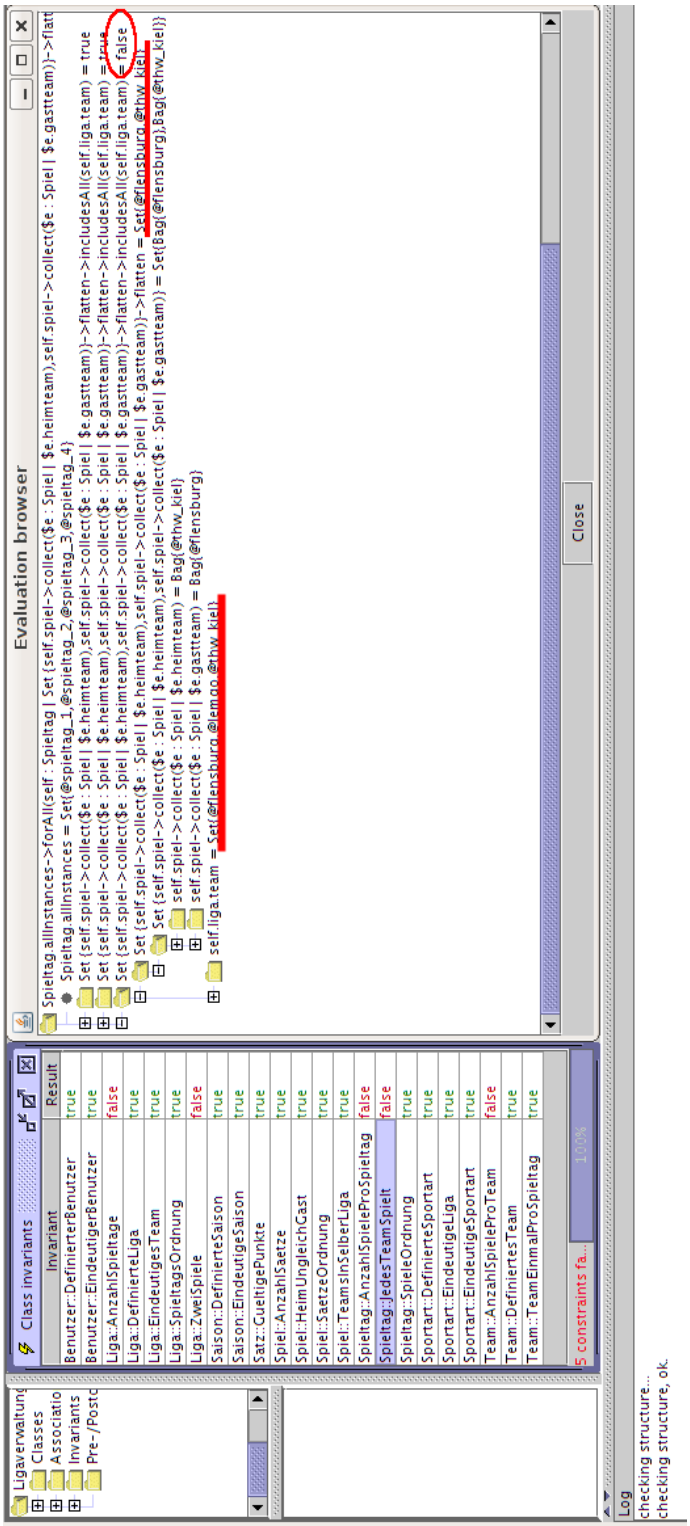


Abbildung 14: Test der Spielteilnahme aller Teams

### 4.2.14 Szenario: Teams spielen mehr als einmal pro Spieltag

Die Invariante `Team::TeamEinmalProSpieltag` verhindert, dass eine Mannschaft mehr als einmal pro Spieltag spielt.

```
1   open -q liga_template.cmd
2
3   -- Testen, ob Teams mehr als einmal pro Spieltag spielen.
4
5   -- Fehlersituation:
6   -- Test der Invariante Team::TeamEinmalProSpieltag
7   --
8
9   !create werder_bayern2 : Spiel
10  !insert (spieltag_1, werder_bayern2) into Findet_statt
11  !insert (werder_bayern2, werder) into Heim
12  !insert (werder_bayern2, bayern) into Gast
13  !set werder_bayern2.ord_nr := 2
```

In Abbildung 15 ist zu sehen, dass die Invariante `Spieltag::JedesTeamSpielt` als unwahr ausgewertet wird. Auf der rechten Seite werden die Details der fehlgeschlagenen Invariante dargestellt.

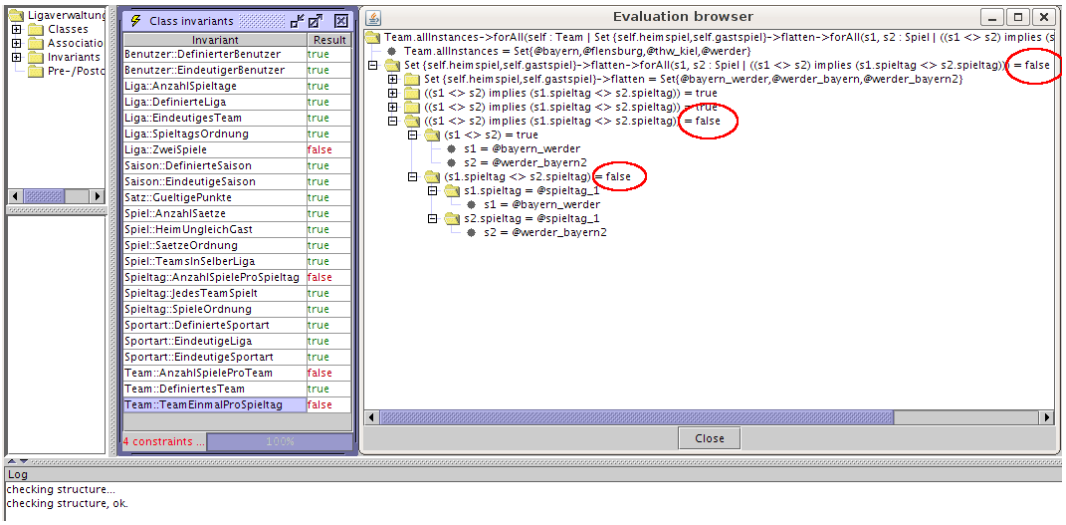


Abbildung 15: Test der Spielteilnahme pro Spieltag

# 5 Sequenzdiagramme

In den folgenden Beispielen wird die Auswertung der Vor- und Nachbedingungen bei Operationen anhand von Sequenzdiagrammen und der Ausgabe in der USE-Konsole dargestellt und erläutert. Zunächst wird ein gültiger Ablauf vorgestellt, anschließend verschiedene ungültige Abläufe als Testfälle präsentiert.

## 5.1 Sequenzdiagramm der Erzeugung einer Liga

Die Abbildung 16 zeigt die sequenzielle Abarbeitung der in Anhang A.4 gegebenen Operationsaufrufe[3] mit den entsprechenden Prüfungen der Vor- und Nachbedingungen.

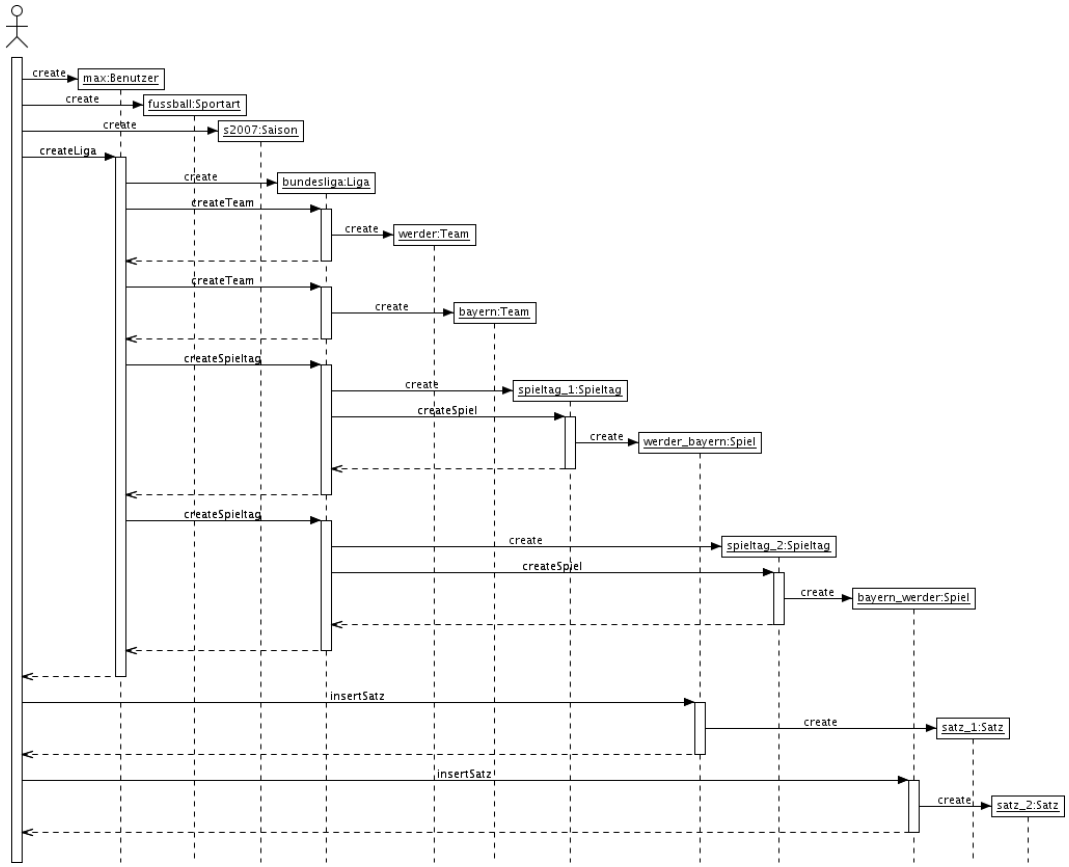


Abbildung 16: Erstellen einer Liga als Sequenzdiagramm

Das Sequenzdiagramm zeigt, wie zunächst ein Benutzer, dann eine Sportart und eine Saison erzeugt werden. Im Kontext des Benutzers wird anschließend eine Liga

[3]Gezeigt werden nur die selbst definierten Operationen und ihre enthaltenen `create`-Aufrufe. `set-` und `insert`-Aufrufe wurden der Übersichtlichkeit halber weggelassen.

erstellt, ihre Attribute werden gesetzt und die Assoziationen mit den neuen Objekten des Benutzers, der Sportart und der Saison werden hergestellt. Im Kontext der Liga werden die Teams mit ihren Attributen und ihrer Assoziation zur Liga erstellt. Ebenfalls im Kontext der Liga werden die Spieltage angelegt, ihre Attribute gesetzt und die Assoziation mit der Liga hergestellt.
Im Kontext der Spieltage werden darauf folgend die Spiele erstellt. Diese erhalten ebenfalls ihre Attribute und die Assoziationen mit den Teams und dem Spieltag.
Damit ist die Erstellung einer minimalen Liga abgeschlossen, die in ihrer Struktur und ihren Einschränkungen vollständig ist. Im Kontext der Spiele werden anschließend noch die Sätze erstellt. Diese erhalten ihre Attribute und die Assoziation mit dem Spiel.

## 5.2 Tests der Vor- und Nachbedingungen

Die folgenden Testfälle zeigen die Auswertung der Vor- und Nachbedingungen bei ungültigen Abläufen. Die Programmausdrucke zeigen jeweils den Aufruf der Vorlage aus Anhang A.3. Diese stellt zunächst eine gültige Objektstruktur her.

### 5.2.1 Test der Operation `createLiga`

In diesem Testfall werden die Vorbedingungen getestet. Dazu wird ein Benutzer mit einem Namen erstellt, in dessen Kontext die Operation `createLiga` mit einem bereits existierenden Namen und ungültigen Werten für die weiteren Parameter ausgeführt wird.

```
1   open -q liga_template.cmd
2
3   -- Testen der Vorbedingungen der Operation createLiga()
4
5   -- Benutzer anlegen
6   !create max:Benutzer
7   !set max.name := 'Max'
8
9   -- Aufruf der Operation createLiga mit einem Namen,
10  -- der schon existiert und ungueltigen Werten fuer die
11  -- weiteren Parameter.
12  !openter max createLiga('1. Bundesliga',0,-1,-1,-1,fussball,s2007)
```

Wie im Konsolenausdruck zu sehen ist, schlagen alle Vorbedingungen fehl, da für alle Parameter ungültige Werte angegeben wurden.

```
use> open -q liga_pre_test_create_liga.cmd
precondition ‘NameNichtVorhanden’ is false
```

```
precondition ‘GewinnsaetzeGueltig’ is false
precondition ‘SiegPunkteGueltig’ is false
precondition ‘NiederlagePunkteGueltig’ is false
precondition ‘UnentschiedenPunkteGueltig’ is false
```

Die Abbildung 17 zeigt den Fehlschlag der Vorbedingungen.

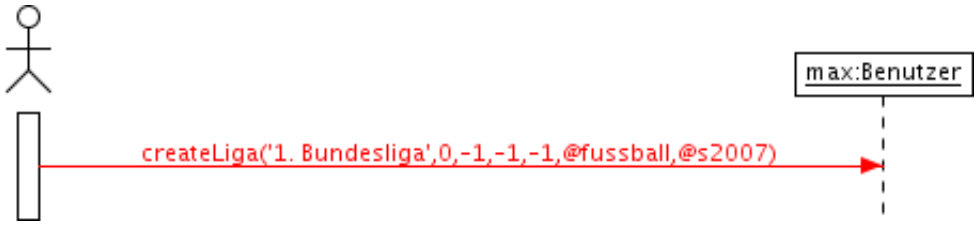


Abbildung 17: Test der Vorbedingungen zu `createLiga`

Im folgenden Testfall wird die Nachbedingung getestet. Dazu wird neben dem Erstellen eines Benutzers die Operation `createLiga` mit gültigen Parametern aufgerufen und sofort wieder verlassen.

```
1   open -q liga_template.cmd
2
3   -- Testen der Nachbedingungen der Operation createLiga()
4
5   -- Benutzer anlegen
6   !create max:Benutzer
7   !set max.name := ’Max’
8
9   -- Aufruf der Operation createLiga mit einem Namen,
10  -- der noch nicht existiert und gueltigen Werten fuer die
11  -- weiteren Parameter.
12  !openter max createLiga(’Bundesliga’, 1, 3, 0, 1, fussball, s2007)
13
14  -- Verlassen der Operation, ohne die richtigen Dinge
15  -- getan zu haben
16  !opexit
```

Wie im Konsolenausdruck zu sehen ist, schlägt die Nachbedingung fehl, da nicht genau ein neues Objekt erzeugt wurde.

```
use> open -q liga_post_test_create_liga.cmd
precondition ‘NameNichtVorhanden’ is true
precondition ‘GewinnsaetzeGueltig’ is true
precondition ‘SiegPunkteGueltig’ is true
```

```
precondition 'NiederlagePunkteGueltig' is true
precondition 'UnentschiedenPunkteGueltig' is true
postcondition 'NurEinObjektMehr' is false
evaluation results:
  Liga.allInstances : Set(Liga) = Set{@fussball_liga,@handball_liga}
  Liga.allInstances@pre : Set(Liga) =
                          Set{@fussball_liga,@handball_liga}
  (Liga.allInstances - Liga.allInstances@pre) : Set(Liga) = Set{}
  (Liga.allInstances - Liga.allInstances@pre)->size : Integer = 0
  1 : Integer = 1
  ((Liga.allInstances - Liga.allInstances@pre)->size = 1) :
  Boolean = false
postcondition 'NeuesObjekt' is true
```

Die Abbildung 18 zeigt den Fehlschlag der Nachbedingung.


Abbildung 18: Test der Nachbedingungen zu `createLiga`

### 5.2.2 Test der Operation `createTeam`

In diesem Testfall wird die Vorbedingung beim Aufruf der Operation `createTeam` getestet. Diese wird mit dem Namen eines bereits existierenden Teams in der Liga aufgerufen.

```
1  open -q liga_template.cmd
2
3  -- Testen der Vorbedingungen der Operation createTeam()
4
5  -- Aufruf der Operation createTeam mit einem Namen,
6  -- der schon existiert.
7  !openter fussball_liga createTeam('Werder Bremen')
```

Der Konsolenausdruck zeigt den Fehlschlag der Vorbedingung.

```
use> open -q liga_pre_test_create_team.cmd
precondition ‘NameNichtVorhanden’ is false
```

Die Abbildung 19 zeigt den Fehlschlag der Vorbedingung.

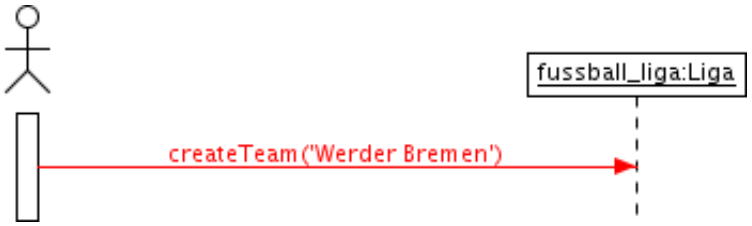


Abbildung 19: Test der Vorbedingung zu `createTeam`

Im folgenden Testfall wird die Nachbedingung getestet. Dazu wird die Operation `createTeam` mit einem gültigen Parameter aufgerufen und sofort wieder verlassen.

```
1   open -q liga_template.cmd
2
3   -- Testen der Nachbedingungen der Operation createTeam()
4
5   -- Aufruf der Operation createTeam mit einem Namen,
6   -- der noch nicht existiert.
7   !openter fussball_liga createTeam(’HSV’)
8
9   -- Verlassen der Operation, ohne die richtigen Dinge
10  -- getan zu haben
11  !opexit
```

Wie im Konsolenausdruck zu sehen ist, schlägt die Nachbedingung fehl, da nicht genau ein neues Objekt erzeugt wurde.

```
use> open -q liga_post_test_create_team.cmd
precondition ‘NameNichtVorhanden’ is true
postcondition ‘NurEinObjektMehr’ is false
evaluation results:
  self : Liga = @fussball_liga
  self.team : Set(Team) = Set{@bayern,@werder}
  self : Liga = @fussball_liga
  self.team@pre : Set(Team) = Set{@bayern,@werder}
  (self.team - self.team@pre) : Set(Team) = Set{}
  (self.team - self.team@pre)->size : Integer = 0
  1 : Integer = 1
  ((self.team - self.team@pre)->size = 1) : Boolean = false
postcondition ‘NeuesObjekt’ is true
```

Die Abbildung 20 zeigt den Fehlschlag der Nachbedingung.

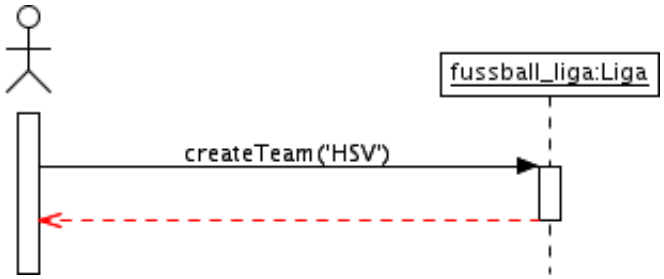


Abbildung 20: Test der Nachbedingungen zu `createTeam`

### 5.2.3 Test der Operation `createSpieltag`

In diesem Testfall wird die Vorbedingung beim Aufruf der Operation `createSpieltag` getestet. Es wird versucht einen zusätzlichen Spieltag zu erstellen, obwohl die maximale Anzahl an Spieltagen in dieser Liga bereits erreicht ist.

```
1  open -q liga_template.cmd
2
3  -- Testen der Vorbedingung der Operation createSpieltag()
4
5  -- Aufruf der Operation createSpieltag fuer eine Liga,
6  -- in der alle Spieltage bereits erstellt wurden.
7  !openter fussball_liga createSpieltag()
```

Der Konsolenausdruck zeigt den Fehlschlag der Vorbedingung.

```
use> open -q liga_pre_test_create_spieltag.cmd
precondition ‘MaxAnzahl’ is false
```

Die Abbildung 21 zeigt den Fehlschlag der Vorbedingung.

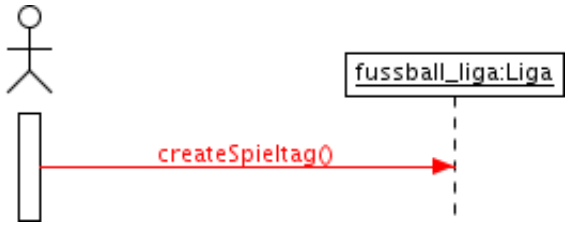


Abbildung 21: Test der Vorbedingung zu `createSpieltag`

Im folgenden Testfall werden die Nachbedingungen getestet. Dazu wird zunächst ein Spieltag aus der gültigen Struktur entfernt, damit die Vorbedingung erfüllt werden kann. Anschließend wird die Operation `createSpieltag` aufgerufen und sofort wieder verlassen.

```
1   open -q liga_template.cmd
2
3   -- Testen der Nachbedingungen der Operation createSpieltag()
4
5   !delete (fussball_liga, spieltag_1) from Verteilt_sich_auf
6
7   -- Aufruf der Operation createSpieltag fuer eine Liga,
8   -- in der nicht alle Spieltage erstellt wurden.
9   !openter fussball_liga createSpieltag()
10
11  -- Verlassen der Operation, ohne die richtigen Dinge
12  -- getan zu haben.
13  !opexit
```

Wie im Konsolenausdruck zu sehen ist, schlägt die erste Nachbedingung fehl, da nicht genau ein neues Objekt erzeugt wurde. Außerdem schlägt auch die zweite Nachbedingung fehl, da keine Ordnungsnummer vergeben wurde.

```
use> open -q liga_post_test_create_spieltag.cmd
precondition ‘MaxAnzahl’ is true
postcondition ‘NurEinObjektMehr’ is false
evaluation results:
  self : Liga = @fussball_liga
  self.spieltag : Set(Spieltag) = Set{@spieltag_2}
  self : Liga = @fussball_liga
  self.spieltag@pre : Set(Spieltag) = Set{@spieltag_2}
  (self.spieltag - self.spieltag@pre) : Set(Spieltag) = Set{}
  (self.spieltag - self.spieltag@pre)->size : Integer = 0
  1 : Integer = 1
  ((self.spieltag - self.spieltag@pre)->size = 1) : Boolean = false
postcondition ‘NeuesObjekt’ is true
postcondition ‘OrdnungsnummerOK’ is false
evaluation results:
  self : Liga = @fussball_liga
  self.spieltag : Set(Spieltag) = Set{@spieltag_2}
  self.spieltag->size : Integer = 1
  self : Liga = @fussball_liga
  self.spieltag : Set(Spieltag) = Set{@spieltag_2}
  self : Liga = @fussball_liga
```

```
self.spieltag@pre : Set(Spieltag) = Set{@spieltag_2}
(self.spieltag - self.spieltag@pre) : Set(Spieltag) = Set{}
(self.spieltag - self.spieltag@pre)->collect($elem4 : Spieltag |
  $elem4.ord_nr) : Bag(Integer) = Bag{}
(self.spieltag - self.spieltag@pre)->collect($elem4 : Spieltag |
  $elem4.ord_nr)->asSequence : Sequence(Integer) = Sequence{}
(self.spieltag - self.spieltag@pre)->collect($elem4 : Spieltag |
  $elem4.ord_nr)->asSequence->first : Integer = Undefined
(self.spieltag->size = (self.spieltag - self.spieltag@pre)
  ->collect($elem4 : Spieltag | $elem4.ord_nr)
    ->asSequence->first) : Boolean = false
```

Die Abbildung 22 zeigt den Fehlschlag der Nachbedingungen.

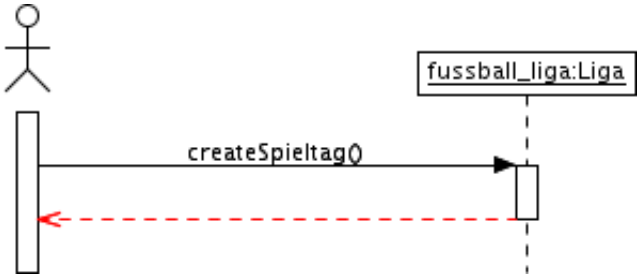


Abbildung 22: Test der Nachbedingungen zu `createSpieltag`

### 5.2.4 Test der Operation `createSpiel`

Dieser Testfall prüft die Vorbedingungen beim Anlegen eines Spiels. Dazu wird die Operation `createSpiel` mit zwei Teams als Parameter aufgerufen, die nicht in derselben Liga sind. Zudem ist bereits die maximale Anzahl an Spielen für diesen Spieltag erreicht.

```
1  open -q liga_template.cmd
2
3  -- Testen der Vorbedingungen der Operation createSpiel()
4
5  -- Aufruf der Operation createSpiel fuer einen Spieltag,
6  -- in welchem alle Spiele bereits erstellt wurden. Ausserdem
7  -- sind die uebergebenen Teams nicht in der selben Liga
8  -- wie der Spieltag
9  !openter spieltag_1 createSpiel(flensburg, thw_kiel)
```

Der Konsolenausdruck zeigt, wie die einzelnen Vorbedingungen fehlschlagen.

```
use> open -q liga_pre_test_create_spiel.cmd
precondition 'MaxAnzahl' is false
precondition 'HeimInSpieltagsLiga' is false
precondition 'GastInSpieltagsLiga' is false
```

Die Abbildung 23 zeigt den Fehlschlag der Vorbedingungen.


Abbildung 23: Test der Vorbedingungen zu `createSpiel`

Im folgenden Testfall wird die Nachbedingung getestet. Dazu wird zunächst die Verbindung eines Spiels zu einem Spieltag entfernt, damit die Vorbedingungen gültig sind. Anschließend wird die Operation `createSpiel` mit gültigen Parametern ausgeführt und sofort wieder verlassen.

```
1   open -q liga_template.cmd
2
3   -- Testen der Nachbedingungen der Operation createSpiel()
4
5   !delete (spieltag_1, bayern_werder) from Findet_statt
6
7   -- Aufruf der Operation createSpiel fuer einen Spieltag,
8   -- in welchem noch nicht alle Spiele erstellt wurden.
9   !openter spieltag_1 createSpiel(bayern, werder)
10
11  -- Verlassen der Operation, ohne die richtigen Dinge
12  -- getan zu haben.
13  !opexit
```

Der Konsolenausdruck zeigt den entsprechenden Fehlschlag der Nachbedingungen, da nicht genau ein neues Objekt erzeugt und keine Ordnungsnummer vergeben wurde.

```
use> open -q liga_post_test_create_spiel.cmd
precondition 'MaxAnzahl' is true
precondition 'HeimInSpieltagsLiga' is true
precondition 'GastInSpieltagsLiga' is true
postcondition 'NurEinObjektMehr' is false
```

```
evaluation results:
  self : Spieltag = @spieltag_1
  self.spiel : Set(Spiel) = Set{}
  self : Spieltag = @spieltag_1
  self.spiel@pre : Set(Spiel) = Set{}
  (self.spiel - self.spiel@pre) : Set(Spiel) = Set{}
  (self.spiel - self.spiel@pre)->size : Integer = 0
  1 : Integer = 1
  ((self.spiel - self.spiel@pre)->size = 1) : Boolean = false
postcondition 'NeuesObjekt' is true
postcondition 'OrdnungsnummerOK' is false
evaluation results:
  self : Spieltag = @spieltag_1
  self.spiel : Set(Spiel) = Set{}
  self.spiel->size : Integer = 0
  self : Spieltag = @spieltag_1
  self.spiel : Set(Spiel) = Set{}
  self : Spieltag = @spieltag_1
  self.spiel@pre : Set(Spiel) = Set{}
  (self.spiel - self.spiel@pre) : Set(Spiel) = Set{}
  (self.spiel - self.spiel@pre)->collect($elem5 : Spiel |
    $elem5.ord_nr) : Bag(Integer) = Bag{}
  (self.spiel - self.spiel@pre)->collect($elem5 : Spiel |
    $elem5.ord_nr)->asSequence : Sequence(Integer) = Sequence{}
  (self.spiel - self.spiel@pre)->collect($elem5 : Spiel |
    $elem5.ord_nr)->asSequence->first : Integer = Undefined
  (self.spiel->size = (self.spiel - self.spiel@pre)
    ->collect($elem5 : Spiel | $elem5.ord_nr)
      ->asSequence->first) : Boolean = false
```

Die Abbildung 24 zeigt den Fehlschlag der Nachbedingungen.

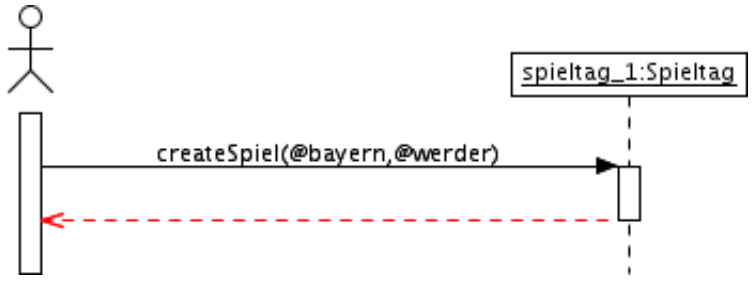


Abbildung 24: Test der Nachbedingungen zu `createSpiel`

### 5.2.5 Test der Operation `insertSatz`

Im folgenden Testfall werden die Vorbedingungen beim Hinzufügen eines Satzes geprüft. Dazu wird zunächst die maximale Anzahl an Sätzen zu einem Spiel erzeugt. Anschließend wird versucht einen neuen Satz mit der Operation `insertSatz` und ungültigen Parametern für Heim- und Gastpunkte hinzuzufügen.

```
1   open -q liga_template.cmd
2
3   -- Testen der Vorbedingungen der Operation insertSatz()
4
5   -- Eine Satz bereits mit dem Spiel verbinden.
6   !create satz_1:Satz
7   !insert (werder_bayern, satz_1) into Ergebnis
8   !set satz_1.ord_nr := 1
9   !set satz_1.heim_punkte := 4
10  !set satz_1.gast_punkte := 0
11
12
13  -- Aufruf der Operation insertSatz fuer ein Spiel,
14  -- fuer das bereits die maximale Anzahl an Ergebnissen
15  -- eingefuegt wurde. Ausserdem sind die uebergebenen
16  -- Punkte nicht gueltig.
17  !openter werder_bayern insertSatz(-1, -1)
```

Der Konsolenausdruck zeigt wie die Vorbedingungen jeweils fehlschlagen.

```
use> open -q liga_pre_test_insert_satz.cmd
precondition ‘MaxAnzahl’ is false
precondition ‘HeimPunkteGueltig’ is false
precondition ‘GastPunkteGueltig’ is false
```

Die Abbildung 25 zeigt den Fehlschlag der Vorbedingungen.

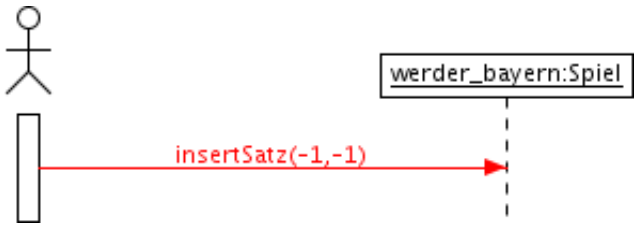


Abbildung 25: Test der Vorbedingungen zu `insertSatz`

In diesem Testfall werden die Nachbedingungen beim Hinzufügen eines Satzes geprüft. Dazu wird die Operation `insertSatz` mit gültigen Parametern ausgeführt und sofort wieder verlassen.

```
1   open -q liga_template.cmd
2
3   -- Testen der Nachbedingungen der Operation insertSatz()
4
5   -- Aufruf der Operation insertSatz fuer ein Spiel,
6   -- fuer das noch nicht die maximale Anzahl an Ergebnissen
7   -- eingefuegt wurde.
8   !openter werder_bayern insertSatz(4, 0)
9
10  -- Verlassen der Operation, ohne die richtigen Dinge
11  -- getan zu haben.
12  !opexit
```

Der Konsolenausdruck zeigt den Fehlschlag der Nachbedingungen, da nicht genau ein neues Objekt erzeugt und keine Ordnungsnummer vergeben wurde.

```
use> open -q liga_post_test_insert_satz.cmd
precondition 'MaxAnzahl' is true
precondition 'HeimPunkteGueltig' is true
precondition 'GastPunkteGueltig' is true
postcondition 'NurEinObjektMehr' is false
evaluation results:
  self : Spiel = @werder_bayern
  self.satz : Set(Satz) = Set{}
  self : Spiel = @werder_bayern
  self.satz@pre : Set(Satz) = Set{}
  (self.satz - self.satz@pre) : Set(Satz) = Set{}
  (self.satz - self.satz@pre)->size : Integer = 0
  1 : Integer = 1
  ((self.satz - self.satz@pre)->size = 1) : Boolean = false
postcondition 'NeuesObjekt' is true
postcondition 'OrdnungsnummerOK' is false
evaluation results:
  self : Spiel = @werder_bayern
  self.satz : Set(Satz) = Set{}
  self.satz->size : Integer = 0
  self : Spiel = @werder_bayern
  self.satz : Set(Satz) = Set{}
  self : Spiel = @werder_bayern
  self.satz@pre : Set(Satz) = Set{}
  (self.satz - self.satz@pre) : Set(Satz) = Set{}
  (self.satz - self.satz@pre)->collect($elem6 : Satz |
    $elem6.ord_nr) : Bag(Integer) = Bag{}
  (self.satz - self.satz@pre)->collect($elem6 : Satz |
```

```
  $elem6.ord_nr)->asSequence : Sequence(Integer) = Sequence{}
(self.satz - self.satz@pre)->collect($elem6 : Satz |
  $elem6.ord_nr)->asSequence->first : Integer = Undefined
(self.satz->size = (self.satz - self.satz@pre)
  ->collect($elem6 : Satz | $elem6.ord_nr)
    ->asSequence->first) : Boolean = false
```

Die Abbildung 26 zeigt den Fehlschlag der Nachbedingungen.

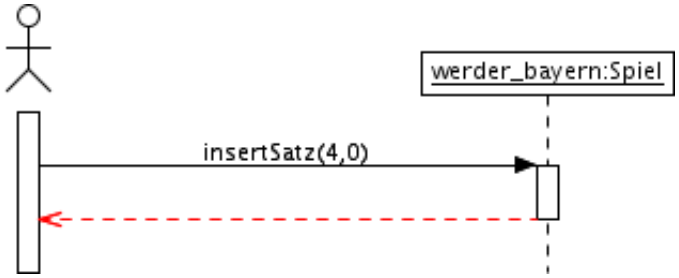


Abbildung 26: Test der Nachbedingungen zu `insertSatz`

# A Spezifikation und Skripte

## A.1 Definition der Klassen, Assoziationen und Invarianten

```
1  model Ligaverwaltung
2
3  -- Klassen
4
5  class Benutzer
6    attributes
7      name : String
8    operations
9      createLiga(n  : String,  gs : Integer,
10                ps : Integer, pn : Integer,
11                pu : Integer, sa : Sportart,
12                s  : Saison)
13 end
14
15 class Sportart
16   attributes
17     name : String
18 end
19
20 class Saison
21   attributes
22     name : String
23 end
24
25 class Liga
26   attributes
27     name : String
28     gewinnsaetze : Integer
29     punkte_sieg : Integer
30     punkte_niederlage : Integer
31     punkte_unentschieden : Integer
32   operations
33     createTeam(n : String)
34     createSpieltag()
35 end
36
37 class Team
38   attributes
39     name : String
40 end
```

```
41
42  class Spieltag
43    attributes
44      ord_nr : Integer
45    operations
46      createSpiel(h : Team, g : Team)
47  end
48
49  class Spiel
50    attributes
51      datum : String
52      ord_nr : Integer
53    operations
54      insertSatz(hp : Integer, gp : Integer)
55  end
56
57  class Satz
58    attributes
59      ord_nr : Integer
60      heim_punkte : Integer
61      gast_punkte : Integer
62  end
63
64
65  -- Assoziationen
66
67  association Verwaltet between
68    Benutzer[0..*]
69    Liga[0..*]
70  end
71
72  composition Gehoert_zu between
73    Sportart[1]
74    Liga[0..*]
75  end
76
77  composition Spielt_waehrend between
78    Saison[1]
79    Liga[0..*]
80  end
81
82  composition Spielt_in between
83    Liga[1]
84    Team[2..*]
```

```
85  end
86
87  composition Verteilt_sich_auf between
88    Liga[1]
89    Spieltag[2..*]
90  end
91
92  composition Findet_statt between
93    Spieltag[1]
94    Spiel[1..*]
95  end
96
97  composition Ergebnis between
98    Spiel[1]
99    Satz[0..*]
100 end
101
102 aggregation Heim between
103   Spiel[0..*] role heimspiel
104   Team[1] role heimteam
105 end
106
107 aggregation Gast between
108   Spiel[0..*] role gastspiel
109   Team[1] role gastteam
110 end
111
112
113
114 -- Einschraenkungen
115
116 constraints
117
118 -- context Benutzer
119
120 -- Name des Benutzers ist eindeutig
121 context Benutzer inv EindeutigerBenutzer:
122   Benutzer.allInstances->forAll(b1, b2 |
123     b1 <> b2 implies b1.name <> b2.name)
124
125 -- Name des Benutzers ist definiert
126 context Benutzer inv DefinierterBenutzer:
127   self.name.isDefined()
128
```

```
129  -- Vor-/Nachbedingungen beim Anlegen einer Liga: Es darf keine
130  -- Liga mit dem als Parameter uebergebenen Namen existieren; Die
131  -- Anzahl der Gewinnsaetze muss groesser 0 sein, sowie Punkte
132  -- fuer Sieg, Niederlage und Unentschieden muessen uebergeben
133  -- werden; Nach dem Anlegen muss genau ein neues Objekt
134  -- existieren;
135  context Benutzer::createLiga(n  : String,  gs : Integer,
136                               ps : Integer, pn : Integer,
137                               pu : Integer, sa : Sportart,
138                               s  : Saison)
139    pre  NameNichtVorhanden         : Liga.allInstances
140                                        ->forAll(l | l.name <> n)
141    pre  GewinnsaetzeGueltig        : gs > 0
142    pre  SiegPunkteGueltig          : ps >= 0
143    pre  NiederlagePunkteGueltig    : pn >= 0
144    pre  UnentschiedenPunkteGueltig : pu >= 0
145    post NurEinObjektMehr           :
146          (Liga.allInstances - Liga.allInstances@pre)->size() = 1
147    post NeuesObjekt                :
148          (Liga.allInstances - Liga.allInstances@pre)
149          ->forAll(l | l.oclIsNew())
150
151
152  -- context Sportart
153
154  -- Name der Sportart ist eindeutig
155  context Sportart inv EindeutigeSportart:
156    Sportart.allInstances->forAll(sa1, sa2 |
157      sa1 <> sa2 implies sa1.name <> sa2.name)
158
159  -- Name der Sportart ist definiert
160  context Sportart inv DefinierteSportart:
161    self.name.isDefined()
162
163  -- Name der Liga in einer Saison, innerhalb einer Sportart,
164  -- ist eindeutig
165  context Sportart inv EindeutigeLiga:
166    self.liga->forAll(l1, l2 |
167      l1 <> l2 and l1.saison = l2.saison implies l1.name <> l2.name)
168
169
170  -- context Saison
171
172  -- Name der Saison ist eindeutig
```

```
173  context Saison inv EindeutigeSaison:
174    Saison.allInstances->forAll(s1, s2 |
175      s1 <> s2 implies s1.name <> s2.name)
176
177  -- Name der Saison ist definiert
178  context Saison inv DefinierteSaison:
179    self.name.isDefined()
180
181
182  -- context Liga
183
184  -- In der Liga sind bei n Teams genau (n - 1) * 2 Spieltage
185  -- definiert.
186  context Liga inv AnzahlSpieltage:
187    (self.team->size() - 1) * 2 = self.spieltag->size()
188
189  -- Name der Liga ist definiert
190  context Liga inv DefinierteLiga:
191    self.name.isDefined()
192
193  -- Name der Mannschaft innerhalb einer Liga ist eindeutig
194  context Liga inv EindeutigesTeam:
195    self.team->forAll(t1, t2 |
196      t1 <> t2 implies t1.name <> t2.name)
197
198  -- Ordnungsnummern der Spieltage gehen von 1..n
199  context Liga inv SpieltagsOrdnung:
200    let st = self.spieltag->collect(ord_nr)->asSet()
201             ->asSequence()->sortedBy(i|i)
202    in
203      st->size()  = self.spieltag->size() and
204      st->first() = 1                     and
205      st->last()  = self.spieltag->size()
206
207  -- Jedes Team einer Liga hat genau zwei Begegnungen gegen jedes
208  -- andere Team
209  context Liga inv ZweiSpiele:
210    self.team->forAll(t1 | self.team->forAll (t2 |
211     t1 <> t2 implies
212        self.spieltag.spiel->collect(Set{heimteam, gastteam})
213                            ->count(Set{t1, t2}) = 2))
214
215  -- Vor-/Nachbedingungen beim Anlegen eines Teams: Ein Team mit
216  -- dem als Parameter uebergebenen Namen darf noch nicht
```

```
217  -- existieren; Nach Anlegen des Teams muss genau ein neues
218  -- Objekt Team existieren
219  context Liga::createTeam(n : String)
220    pre  NameNichtVorhanden : self.team->forAll(t | t.name <> n)
221    post NurEinObjektMehr   : (self.team -
222                               self.team@pre)->size() = 1
223    post NeuesObjekt        : (self.team - self.team@pre)
224                              ->forAll(t | t.oclIsNew())
225
226  -- Vor-/Nachbedingungen beim Anlegen eines Spieltags: Es darf
227  -- noch nicht die maximale Anzahl an Spieltagen fuer die Liga
228  -- erreicht sein; Nach Anlegen des Spieltags muss genau ein
229  -- neues Objekt Spieltag existieren; Das neue Objekt Spieltag
230  -- muss die hoechste Ordnungsnummer als Attribut besitzen.
231  context Liga::createSpieltag()
232    pre  MaxAnzahl        : self.spieltag->size() <
233                            (self.team->size() - 1) * 2
234    post NurEinObjektMehr : (self.spieltag -
235                             self.spieltag@pre)->size() = 1
236    post NeuesObjekt      : (self.spieltag - self.spieltag@pre)
237                             ->forAll(st | st.oclIsNew())
238    post OrdnungsnummerOK : self.spieltag->size() =
239                         (self.spieltag - self.spieltag@pre)
240                         ->collect(ord_nr)->asSequence()->first()
241
242  -- context Team
243
244  -- Jede Mannschaft spielt, bei n Teams in der Liga,
245  -- genau (n-1) * 2 Mal
246  context Team inv AnzahlSpieleProTeam:
247    (self.liga.team->size() - 1) * 2 =
248    Set{self.heimspiel, self.gastspiel}->flatten()->size()
249
250  -- Jede Mannschaft spielt nur einmal an einem Spieltag
251  context Team inv TeamEinmalProSpieltag:
252    Set{self.heimspiel, self.gastspiel}->flatten()
253      ->forAll(s1, s2 | s1 <> s2 implies
254                s1.spieltag <> s2.spieltag)
255
256  -- Name der Mannschaft ist definiert
257  context Team inv DefiniertesTeam:
258    self.name.isDefined()
259
260
```

```
261  -- context Spieltag
262
263  -- An jedem Spieltag spielt jede Mannschaft
264  context Spieltag inv JedesTeamSpielt:
265    Set{self.spiel.heimteam, self.spiel.gastteam}->flatten()
266    ->includesAll(self.liga.team)
267
268  -- Bei n Teams in einer Liga gibt es genau n/2 Spiele
269  -- pro Spieltag
270  context Spieltag inv AnzahlSpieleProSpieltag:
271    self.liga.team->size() / 2 = self.spiel->size()
272
273  -- Ordnungsnummern der Spiele gehen von 1..n
274  context Spieltag inv SpieleOrdnung:
275    let st = self.spiel->collect(ord_nr)->asSet()
276             ->asSequence()->sortedBy(i|i)
277    in
278      st->size()  = self.spiel->size() and
279      st->first() = 1                  and
280      st->last()  = self.spiel->size()
281
282  -- Vor-/Nachbedingungen beim Anlegen eines Spiels: Es darf noch
283  -- nicht die maximale Anzahl an Spielen fuer den Spieltag
284  -- erreicht sein; Die uebergebenen Teams muessen aus der selben
285  -- Liga wie der Spieltag kommen; Nach Anlegen des Spiels muss
286  -- genau ein neues Objekt Spiel existieren; Das neue Objekt
287  -- Spiel muss die hoechste Ordnungsnummer als Attribut besitzen.
288  context Spieltag::createSpiel(h: Team, g : Team)
289    pre  MaxAnzahl           : self.spiel->size() <
290                               self.liga.team->size() / 2
291    pre  HeimInSpieltagsLiga : self.liga = h.liga
292    pre  GastInSpieltagsLiga : self.liga = g.liga
293    post NurEinObjektMehr    : (self.spiel -
294                                self.spiel@pre)->size() = 1
295    post NeuesObjekt         : (self.spiel - self.spiel@pre)
296                               ->forAll(s | s.oclIsNew())
297    post OrdnungsnummerOK    : self.spiel->size() =
298                           (self.spiel - self.spiel@pre)
299                           ->collect(ord_nr)->asSequence()->first()
300
301
302  -- context Spiel
303
304  -- In einem Spiel koennen sich nur Teams der selben
```

```
305  -- Liga begegnen
306  context Spiel inv TeamsInSelberLiga:
307    self.heimteam.liga = self.gastteam.liga
308
309  -- Ein Team darf nicht gegen sich selber spielen
310  context Spiel inv HeimUngleichGast:
311    self.heimteam <> self.gastteam
312
313  -- Anzahl der Saetze in einem Spiel zwischen Liga.gewinnsaetze
314  -- und Liga.gewinnsaetze * 2 - 1 oder keine.
315  context Spiel inv AnzahlSaetze:
316    let saetze : Integer = self.satz->size()
317    in
318      saetze >= self.spieltag.liga.gewinnsaetze and
319      saetze <= self.spieltag.liga.gewinnsaetze * 2 - 1 or
320      saetze = 0
321
322  -- Ordnungsnummern der Saetze gehen von 1..n,
323  -- wenn es Saetze gibt.
324  context Spiel inv SaetzeOrdnung:
325    let st = self.satz->collect(ord_nr)->asSet()
326             ->asSequence()->sortedBy(i|i)
327    in
328      st->size()  = 0 or
329      st->size()  = self.satz->size() and
330      st->first() = 1                 and
331      st->last()  = self.satz->size()
332
333  -- Vor-/Nachbedingungen beim Hinzufuegen eines Satzes: Es darf
334  -- noch nicht die maximale Anzahl Saetze fuer das Spiel erreicht
335  -- sein; Nach Anlegen des Satzes muss genau ein neues Objekt
336  -- Satz existieren; Das neue Objekt Satz muss die hoechste
337  -- Ordnungsnummer als Attribut besitzen.
338  context Spiel::insertSatz(hp : Integer, gp : Integer)
339    pre  MaxAnzahl         : self.satz->size() <
340                               self.spieltag.liga.gewinnsaetze *2 -1
341    pre  HeimPunkteGueltig : hp >= 0
342    pre  GastPunkteGueltig : gp >= 0
343    post NurEinObjektMehr  : (self.satz -
344                               self.satz@pre)->size() = 1
345    post NeuesObjekt       : (self.satz - self.satz@pre)
346                               ->forAll(s | s.oclIsNew())
347    post OrdnungsnummerOK  : self.satz->size() =
348                            (self.satz - self.satz@pre)
```

```
349                          ->collect(ord_nr)->asSequence()->first()
350
351
352   -- context Satz
353
354   -- Die Punkte/Tore muessen groesser gleich 0 sein.
355   context Satz inv GueltigePunkte:
356     self.heim_punkte >= 0 and self.gast_punkte >= 0
```

## A.2 Skript des gültigen Systemzustands

```
1   reset
2
3   -- Liga
4   !create bundesliga : Liga
5   !set bundesliga.gewinnsaetze := 1
6   !set bundesliga.name := '1. Bundesliga'
7
8
9   -- Sportart
10  !create fussball : Sportart
11  !insert (fussball, bundesliga) into Gehoert_zu
12  !set fussball.name := 'Fussball'
13
14
15  -- Saison
16  !create s2007 : Saison
17  !insert (s2007, bundesliga) into Spielt_waehrend
18  !set s2007.name := 'Saison 2007/08'
19
20
21  -- Benutzer
22  !create edvin : Benutzer
23  !set edvin.name := 'Edvin'
24
25  !create roman : Benutzer
26  !set roman.name := 'Roman'
27  !insert (roman, bundesliga) into Verwaltet
28
29
30  -- Teams
31
32  !create bayern : Team
```

```
33  !insert (bundesliga, bayern) into Spielt_in
34  !set bayern.name := 'Bayern'
35
36  !create werder : Team
37  !insert (bundesliga, werder) into Spielt_in
38  !set werder.name := 'Werder'
39
40  !create schalke : Team
41  !insert (bundesliga, schalke) into Spielt_in
42  !set schalke.name := 'Schalke'
43
44  !create hsv : Team
45  !insert (bundesliga, hsv) into Spielt_in
46  !set hsv.name := 'HSV'
47
48
49  -- Spieltage
50
51  !create spieltag_1 : Spieltag
52  !insert (bundesliga, spieltag_1) into Verteilt_sich_auf
53  !set spieltag_1.ord_nr := 1
54
55  !create spieltag_2 : Spieltag
56  !insert (bundesliga, spieltag_2) into Verteilt_sich_auf
57  !set spieltag_2.ord_nr := 2
58
59  !create spieltag_3 : Spieltag
60  !insert (bundesliga, spieltag_3) into Verteilt_sich_auf
61  !set spieltag_3.ord_nr := 3
62
63  !create spieltag_4 : Spieltag
64  !insert (bundesliga, spieltag_4) into Verteilt_sich_auf
65  !set spieltag_4.ord_nr := 4
66
67  !create spieltag_5 : Spieltag
68  !insert (bundesliga, spieltag_5) into Verteilt_sich_auf
69  !set spieltag_5.ord_nr := 5
70
71  !create spieltag_6 : Spieltag
72  !insert (bundesliga, spieltag_6) into Verteilt_sich_auf
73  !set spieltag_6.ord_nr := 6
74
75
76  -- Spiele
```

```
77
78  !create bayern_werder : Spiel
79  !insert (spieltag_1, bayern_werder) into Findet_statt
80  !insert (bayern_werder, bayern) into Heim
81  !insert (bayern_werder, werder) into Gast
82  !set bayern_werder.ord_nr := 1
83
84  !create schalke_hsv : Spiel
85  !insert (spieltag_1, schalke_hsv) into Findet_statt
86  !insert (schalke_hsv, schalke) into Heim
87  !insert (schalke_hsv, hsv) into Gast
88  !set schalke_hsv.ord_nr := 2
89
90  !create bayern_schalke : Spiel
91  !insert (spieltag_2, bayern_schalke) into Findet_statt
92  !insert (bayern_schalke, bayern) into Heim
93  !insert (bayern_schalke, schalke) into Gast
94  !set bayern_schalke.ord_nr := 1
95
96  !create werder_hsv : Spiel
97  !insert (spieltag_2, werder_hsv) into Findet_statt
98  !insert (werder_hsv, werder) into Heim
99  !insert (werder_hsv, hsv) into Gast
100 !set werder_hsv.ord_nr := 2
101
102 !create bayern_hsv : Spiel
103 !insert (spieltag_3, bayern_hsv) into Findet_statt
104 !insert (bayern_hsv, bayern) into Heim
105 !insert (bayern_hsv, hsv) into Gast
106 !set bayern_hsv.ord_nr := 1
107
108 !create werder_schalke : Spiel
109 !insert (spieltag_3, werder_schalke) into Findet_statt
110 !insert (werder_schalke, werder) into Heim
111 !insert (werder_schalke, schalke) into Gast
112 !set werder_schalke.ord_nr := 2
113
114 !create werder_bayern : Spiel
115 !insert (spieltag_4, werder_bayern) into Findet_statt
116 !insert (werder_bayern, werder) into Heim
117 !insert (werder_bayern, bayern) into Gast
118 !set werder_bayern.ord_nr := 1
119
120 !create hsv_schalke : Spiel
```

```
121  !insert (spieltag_4, hsv_schalke) into Findet_statt
122  !insert (hsv_schalke, hsv) into Heim
123  !insert (hsv_schalke, schalke) into Gast
124  !set hsv_schalke.ord_nr := 2
125
126  !create schalke_bayern : Spiel
127  !insert (spieltag_5, schalke_bayern) into Findet_statt
128  !insert (schalke_bayern, schalke) into Heim
129  !insert (schalke_bayern, bayern) into Gast
130  !set schalke_bayern.ord_nr := 1
131
132  !create hsv_werder : Spiel
133  !insert (spieltag_5, hsv_werder) into Findet_statt
134  !insert (hsv_werder, hsv) into Heim
135  !insert (hsv_werder, werder) into Gast
136  !set hsv_werder.ord_nr := 2
137
138  !create hsv_bayern : Spiel
139  !insert (spieltag_6, hsv_bayern) into Findet_statt
140  !insert (hsv_bayern, hsv) into Heim
141  !insert (hsv_bayern, bayern) into Gast
142  !set hsv_bayern.ord_nr := 1
143
144  !create schalke_werder : Spiel
145  !insert (spieltag_6, schalke_werder) into Findet_statt
146  !insert (schalke_werder, schalke) into Heim
147  !insert (schalke_werder, werder) into Gast
148  !set schalke_werder.ord_nr := 2
149
150
151  -- Saetze
152
153  !create satz_1 : Satz
154  !insert (bayern_werder, satz_1) into Ergebnis
155  !set satz_1.ord_nr := 1
156  !set satz_1.heim_punkte := 0
157  !set satz_1.gast_punkte := 4
158
159  !create satz_2 : Satz
160  !insert (schalke_hsv, satz_2) into Ergebnis
161  !set satz_2.ord_nr := 1
162  !set satz_2.heim_punkte := 0
163  !set satz_2.gast_punkte := 4
164
```

```
165  !create satz_3 : Satz
166  !insert (bayern_schalke, satz_3) into Ergebnis
167  !set satz_3.ord_nr := 1
168  !set satz_3.heim_punkte := 0
169  !set satz_3.gast_punkte := 4
170
171  !create satz_4 : Satz
172  !insert (werder_hsv, satz_4) into Ergebnis
173  !set satz_4.ord_nr := 1
174  !set satz_4.heim_punkte := 0
175  !set satz_4.gast_punkte := 4
176
177  !create satz_5 : Satz
178  !insert (bayern_hsv, satz_5) into Ergebnis
179  !set satz_5.ord_nr := 1
180  !set satz_5.heim_punkte := 0
181  !set satz_5.gast_punkte := 4
182
183  !create satz_6 : Satz
184  !insert (werder_schalke, satz_6) into Ergebnis
185  !set satz_6.ord_nr := 1
186  !set satz_6.heim_punkte := 0
187  !set satz_6.gast_punkte := 4
188
189  !create satz_7 : Satz
190  !insert (werder_bayern, satz_7) into Ergebnis
191  !set satz_7.ord_nr := 1
192  !set satz_7.heim_punkte := 0
193  !set satz_7.gast_punkte := 4
194
195  !create satz_8 : Satz
196  !insert (hsv_schalke, satz_8) into Ergebnis
197  !set satz_8.ord_nr := 1
198  !set satz_8.heim_punkte := 0
199  !set satz_8.gast_punkte := 4
200
201  !create satz_9 : Satz
202  !insert (schalke_bayern, satz_9) into Ergebnis
203  !set satz_9.ord_nr := 1
204  !set satz_9.heim_punkte := 0
205  !set satz_9.gast_punkte := 4
206
207  !create satz_10 : Satz
208  !insert (hsv_werder, satz_10) into Ergebnis
```

```
209  !set satz_10.ord_nr := 1
210  !set satz_10.heim_punkte := 0
211  !set satz_10.gast_punkte := 4
212
213  !create satz_11 : Satz
214  !insert (hsv_bayern, satz_11) into Ergebnis
215  !set satz_11.ord_nr := 1
216  !set satz_11.heim_punkte := 0
217  !set satz_11.gast_punkte := 4
218
219  !create satz_12 : Satz
220  !insert (schalke_werder, satz_12) into Ergebnis
221  !set satz_12.ord_nr := 1
222  !set satz_12.heim_punkte := 0
223  !set satz_12.gast_punkte := 4
```

### A.3 Vorlagenskript

```
1   reset
2
3   -- Liga
4   !create fussball_liga : Liga
5   !set fussball_liga.name := '1. Bundesliga'
6   !set fussball_liga.gewinnsaetze := 1
7
8   !create handball_liga : Liga
9   !set handball_liga.name := '1. Bundesliga'
10  !set handball_liga.gewinnsaetze := 1
11
12  -- Sportart
13  !create fussball : Sportart
14  !insert (fussball, fussball_liga) into Gehoert_zu
15  !set fussball.name := 'Fussball'
16
17  !create handball : Sportart
18  !insert (handball, handball_liga) into Gehoert_zu
19  !set handball.name := 'Handball'
20
21
22  -- Saison
23  !create s2007 : Saison
24  !insert (s2007, fussball_liga) into Spielt_waehrend
25  !insert (s2007, handball_liga) into Spielt_waehrend
```

```
26  !set s2007.name := 'Saison 2007'
27
28  !create s2008 : Saison
29  !set s2008.name := 'Saison 2008'
30
31  -- Teams
32
33  !create bayern : Team
34  !insert (fussball_liga, bayern) into Spielt_in
35  !set bayern.name := 'Bayern Muenchen'
36
37  !create werder : Team
38  !insert (fussball_liga, werder) into Spielt_in
39  !set werder.name := 'Werder Bremen'
40
41  !create thw_kiel : Team
42  !insert (handball_liga, thw_kiel) into Spielt_in
43  !set thw_kiel.name := 'THW Kiel'
44
45  !create flensburg : Team
46  !insert (handball_liga, flensburg) into Spielt_in
47  !set flensburg.name := 'Flensburg'
48
49
50  -- Spieltage
51
52  !create spieltag_1 : Spieltag
53  !insert (fussball_liga, spieltag_1) into Verteilt_sich_auf
54  !set spieltag_1.ord_nr := 1
55
56  !create spieltag_2 : Spieltag
57  !insert (fussball_liga, spieltag_2) into Verteilt_sich_auf
58  !set spieltag_2.ord_nr := 2
59
60  !create spieltag_3 : Spieltag
61  !insert (handball_liga, spieltag_3) into Verteilt_sich_auf
62  !set spieltag_3.ord_nr := 1
63
64  !create spieltag_4 : Spieltag
65  !insert (handball_liga, spieltag_4) into Verteilt_sich_auf
66  !set spieltag_4.ord_nr := 2
67
68
69  -- Spiele
```

```
70
71  !create bayern_werder : Spiel
72  !insert (spieltag_1, bayern_werder) into Findet_statt
73  !insert (bayern_werder, bayern) into Heim
74  !insert (bayern_werder, werder) into Gast
75  !set bayern_werder.ord_nr := 1
76
77  !create werder_bayern : Spiel
78  !insert (spieltag_2, werder_bayern) into Findet_statt
79  !insert (werder_bayern, werder) into Heim
80  !insert (werder_bayern, bayern) into Gast
81  !set werder_bayern.ord_nr := 1
82
83  !create kiel_flensburg : Spiel
84  !insert (spieltag_3, kiel_flensburg) into Findet_statt
85  !insert (kiel_flensburg, thw_kiel) into Heim
86  !insert (kiel_flensburg, flensburg) into Gast
87  !set kiel_flensburg.ord_nr := 1
88
89  !create flensburg_kiel : Spiel
90  !insert (spieltag_4, flensburg_kiel) into Findet_statt
91  !insert (flensburg_kiel, flensburg) into Heim
92  !insert (flensburg_kiel, thw_kiel) into Gast
93  !set flensburg_kiel.ord_nr := 1
```

## A.4 Sequenzskript

```
1  reset
2
3  -- Benutzer anlegen
4  !create max:Benutzer
5  !set max.name := 'Max'
6
7  -- Sportart anlegen
8  !create fussball:Sportart
9  !set fussball.name := 'Fussball'
10
11 -- Saison anlegen
12 !create s2007:Saison
13 !set s2007.name := 'Saison 2007'
14
15 -- Benutzer legt Liga an
16 !openter max createLiga('Bundesliga', 1, 3, 0, 1, fussball, s2007)
```

```
17      !create bundesliga:Liga
18      !set bundesliga.name := n
19      !set bundesliga.gewinnsaetze := gs
20      !set bundesliga.punkte_sieg := ps
21      !set bundesliga.punkte_niederlage := pn
22      !set bundesliga.punkte_unentschieden := pu
23      !insert (sa, bundesliga) into Gehoert_zu
24      !insert (s, bundesliga) into Spielt_waehrend
25      !insert (self, bundesliga) into Verwaltet
26
27      -- Anlegen der Teams in der Liga
28      !openter bundesliga createTeam('Werder Bremen')
29        !create werder:Team
30        !set werder.name := n
31        !insert (self, werder) into Spielt_in
32      !opexit
33
34      !openter bundesliga createTeam('Bayern Muenchen')
35        !create bayern:Team
36        !set bayern.name := n
37        !insert (self, bayern) into Spielt_in
38      !opexit
39
40      -- Anlegen des 1. Spieltags in der Liga
41      !openter bundesliga createSpieltag()
42        !create spieltag_1:Spieltag
43        !set spieltag_1.ord_nr := 1
44        !insert (self, spieltag_1) into Verteilt_sich_auf
45
46        -- Anlegen der Spiele dieses Spieltags
47        !openter spieltag_1 createSpiel(werder, bayern)
48          !create werder_bayern:Spiel
49          !set werder_bayern.ord_nr := 1
50          !insert (self, werder_bayern) into Findet_statt
51          !insert (werder_bayern, h) into Heim
52          !insert (werder_bayern, g) into Gast
53        !opexit
54
55      -- Verlassen des Spieltags
56      !opexit
57
58      -- Anlegen des 2. Spieltags in der Liga
59      !openter bundesliga createSpieltag()
60        !create spieltag_2:Spieltag
```

```
61      !set spieltag_2.ord_nr := 2
62      !insert (self, spieltag_2) into Verteilt_sich_auf
63
64      -- Anlegen der Spiele dieses Spieltags
65      !openter spieltag_2 createSpiel(bayern, werder)
66        !create bayern_werder:Spiel
67        !set bayern_werder.ord_nr := 1
68        !insert (self, bayern_werder) into Findet_statt
69        !insert (bayern_werder, h) into Heim
70        !insert (bayern_werder, g) into Gast
71      !opexit
72
73    -- Verlassen des Spieltags
74    !opexit
75
76  --Verlassen der Liga
77  !opexit
78
79
80  -- Setzen der Ergebnisse
81  !openter werder_bayern insertSatz(4, 0)
82    !create satz_1:Satz
83    !set satz_1.ord_nr := 1
84    !set satz_1.heim_punkte := hp
85    !set satz_1.gast_punkte := gp
86    !insert (werder_bayern, satz_1) into Ergebnis
87  !opexit
88
89  !openter bayern_werder insertSatz(0, 2)
90    !create satz_2:Satz
91    !set satz_2.ord_nr := 1
92    !set satz_2.heim_punkte := hp
93    !set satz_2.gast_punkte := gp
94    !insert (bayern_werder, satz_2) into Ergebnis
95  !opexit
```

# B Literaturangaben

[1] Booch, Grady; Rumbaugh, James; Jacobson, Ivar (2006): Das UML Benutzerhandbuch. Aktuell zur Version 2.0. München: Addison-Wesley.

[2] Rumbaugh, James; Jacobson, Ivar; Booch, Grady (1999): The Unified Modeling Language Reference Manual. Reading, Massachusetts: Addison-Wesley Longman.

[3] Warmer, Jos; Kleppe, Anneke (2004): Object Constraint Language 2.0. Bonn: mitp.

[4] Folien zur Veranstaltung 'Entwurf von Informationssystemen' im SoSe 2007. http://db.informatik.uni-bremen.de/teaching/courses/ss2007_eis/ Abgerufen am 20. September 2007.

[5] USE - UML Based Specification Environment, Version 2.3.1 http://www.db.informatik.uni-bremen.de/projects/USE/

# Abbildungsverzeichnis