

Software Engineering: Specification, Implementation, Verification

Chapter 6: Software Verification

Suad Alagić

Springer 2017

Main topics

- Preconditions and postconditions
- Object invariants
- Assertions and inheritance
- Assertions for interfaces
- Sample application
- Transaction verification
- Integrated specification and verification systems

Verification technologies

- Assertion languages
- Programming language extensions
- Static and dynamic verification
- Specification and verification systems

Pre- and post- conditions

- Contracts
- Requires method
- Ensures method
- Pure methods
- References to previous state
- References to result

Preconditions and postconditions

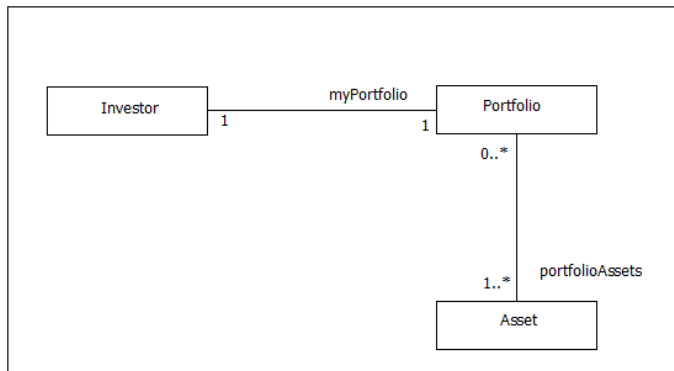


Figure: Associations for investing

Preconditions and postconditions

```
void buyAsset(Asset a)
{ ContractRequires(
!this.getMyPortfolio().getPortfolioAssets().Contains(a))
```

```
    Contract.Ensures(
this.getMyPortfolio().getPortfolioAssets().Contains(a))
// code
}
```

Preconditions and postconditions

```
void sellAsset(Asset a)
{ ContractRequires(
  this.getMyPortfolio().getPortfolioAssets().Contains(a))

  Contract.Ensures(
    !this.getMyPortfolio().getPortfolioAssets().Contains(a))
  // code
}
```

```
class Investor: InvestorI {  
  [Pure]  
  Portfolio getMyPortfolio()  
  { // code  
    }  
  void buyAsset(Asset a)  
  { // code  
    }  
  void sellAsset(Asset a)  
  { // code  
    }  
}
```


Class structure

```
class Portfolio: IPortfolio {  
  [Pure]  
  Collection<Asset> getPortfolioAssets()  
  { // code  
    }  
  [Pure]  
  Broker getBroker()  
  { // code  
    }  
  [Pure]  
  Investor getInvestor()  
  { // code  
    }  
}
```

Associations

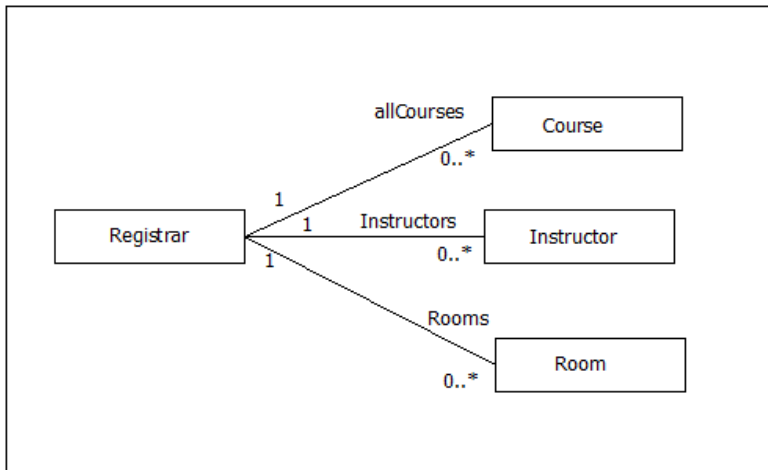


Figure: Associations for scheduling a course

Preconditions and postconditions

`Contract.Requires(!this.getAllCourses().Contains(c))`

`Contract.Requires(Contract.Exists(this.getRooms(), r =>
r.suitableFor(c))`

`Contract.Ensures(this.getAllCourses().Contains(c))`

Preconditions and postconditions

```
void scheduleCourse(Course c)
{ Contract.Requires(!this.getAllCourses().Contains(c));
  Contract.Requires(Contract.Exists(
    this.getRooms(), r => r.suitableFor(c))
  Contract.Ensures(this.getAllCourses().Contains(c));
  // code
}
```

Preconditions and postconditions

```
void deleteCourse(Course c)
{ Contract.Requires(this.getAllCourses().Contains(c));
  Contract.Ensures(!this.getAllCourses().Contains(c));
  // code
}
```

```
class Registrar: IRegistrar {  
  [Pure]  
  Collection<Course> getAllCourses()  
  { // code }  
  [Pure]  
  Collection<Instructor> getInstructors()  
  { // code }  
  [Pure]  
  Collection<Students> getStudents()  
  { // code }  
  [Pure]  
  Collection<Room> getRooms()  
  { // code }
```

```
void scheduleCourse(Course c)
{ // code
}
void deleteCourse(Course c)
{ // code
}
}
```

Associations

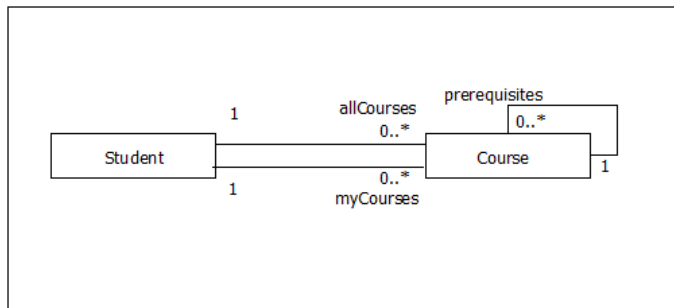


Figure: Associations for enrolling in a course

Preconditions and postconditions

```
void enrollInCourse(Course c)
{ Contract.Requires(!this.getMyCourses().Contains(c));
  Contract.Requires(Contract.ForAll(
    c.getPrerequisites(), p => this.getMyCourses().Contains(p)));
  Contract.Ensures(this.getMyCourses().Contains(c));
// code
}
```

Preconditions and postconditions

```
void dropCourse(Course c)
{ Contract.Requires(this.getMyCourses().Contains(c));
  Contract.Ensures(!this.getMyCourses().Contains(c));
  // code
}
```

Class structure

```
class Student: IStudent {  
  [Pure]  
  Collection<Course> getAllCourses()  
  { // code  
  }  
  [Pure]  
  Collection<Course> getMyCourses();  
  { // code  
  }  
  void enrollInCourse(Course c)  
  { // code  
  }  
  void dropCourse(Course c)  
  { // code  
  }  
}
```

Object invariants

- Method Invariant
- Specifying object invariants
- ForAll method
- Exists method

Object invariants

```
class Flight: IFlight {  
    // fields etc.  
    [ContractInvariantMethod]  
    void ObjectInvariant()  
    { Contract.Invariant(  
        this.to != this.from);  
        Contract.Invariant(  
            this.departureTime.precedes(this.arrivalTime));  
        Contract.Invariant(  
            this.flightStatus != "idle" ||  
            Time.now().precedes(this.departureTime));  
    }  
}
```

Object invariants

```
Contract.Invariant(  
    Time.now().after(this.departureTime) &&  
    (!Time.now().precedes(arrivalTime) ||  
    (flightStatus = "takeoff" || flightStatus = "flying" ||  
    this.flightStatus = "landing")))  
// dummy code  
}  
}
```

Object invariants

```
class FlightSchedule: IFlightSchedule {  
    //... ;  
    [ContractInvariantMethod]  
    void ObjectInvariant()  
    { Contract.Invariant(  
        Contract.ForAll( this.flights,  
            (f1,f2) =>!(f1.flightId = f2.flightId) || f1.Equals(f2))  
    )  
}
```

Object invariants

```
Contract.Invariant(ForAll(this.flights,  
    f => Contract.Exists(this.planes,  
        p => f.plane=p))  
Contract.Invariant(Contract.ForAll(this.flights,  
    f => Contract.Exists(this.airports,  
        a => f.from=a)));  
Contract.Invariant(Contract.ForAll(this.flights,  
    f => Contract.Exists(this.airports,  
        a => f.to=a)));  
// dummy code  
}  
}
```


Object invariants

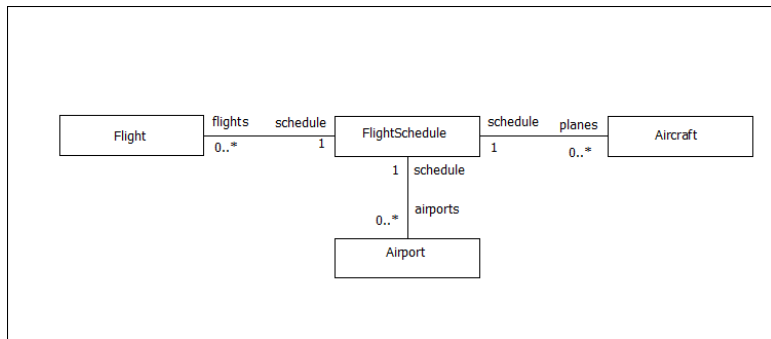


Figure: Associations for flight scheduling

Preconditions and postconditions

```
void scheduleFlight(String flightId,  
                    Airport from,to,  
                    Time departureTime, arrivalTime  
                    Aircraft plane)  
{ Contract.Requires(!to.Equals(from));  
  Contract.Requires(departureTime.precedes(arrivalTime));
```

Preconditions and postconditions

```
Contract.Requires(Contract.ForAll(this.flights,  
    f => f.flightId != flightId);  
Contract.Requires(Contract.Exists(this.planes,  
    p => plane==p))  
Contract.Ensures(Contract.Exists(this.flights,  
    f => f.flightId==flightId))  
// code  
}
```

Preconditions and postconditions

```
void cancelFlight(String flightId)
    Contract.Requires(ContractExists(this.flights,
        f => f.flightId=flightId))
    Contract.Requires(ContractForAll(this.flights,
        f => f.flightId != flightId || f.flightStatus != "landing"))
    Contract.Ensures(Contract.ForAll(this.flights,
        f => f.flightId != flightId))
// code
}
```

Preconditions and postconditions

```
void redirectFlight(String flightId,  
                    Airport newDestination)  
    Contract.Requires(Contract.Exists(this.flights,  
        f => f.flightId==flightId))  
    Contract.Requires(Contract.ForAll(this.flights,  
        f => f.flightId != flightId || f.from != newDestination))  
    Contract.Ensures(Contract.ForAll(this.flights,  
        f => f.flightId != flightId ||  
        f.destination.Equals(newDestination)))  
// code  
}
```

Assertions and inheritance

- Strengthening inherited invariants
- Strengthening inherited postconditions
- Invariance of preconditions

Assertions and inheritance

```
class Airport {  
    ... ;  
    [ContractInvariantMethod]  
    void ObjectInvariant ()  
    { Contract.Invariant(this.numOfRunways >= 1 &&  
        this.numOfRunways <= 30)  
    // dummycode  
    }
```

Assertions and inheritance

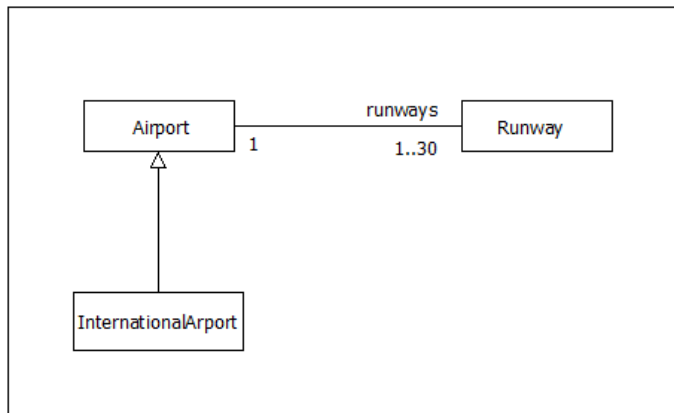


Figure: Inheritance for airports

Assertions and inheritance

```
void addRunway(Runway strip)
{ Contract.Requires(Contract.ForAll(this.runways,
    r => !r.Equals(strip)));
  Contract.Ensures(Contract.Exists(this.runways,
    r => r.Equals(strip)))
// code
}
```

Assertions and inheritance

```
void closeRunway(Runway strip)
{ Contract.Requires(Contract.Exists(this.runways,
    r => r.Equals(strip)));
  Contract.Requires(this.numOfRunways > 1);
  Contract.Ensures(Contract.ForAll(this.runways,
    r => !r.Equals(strip)));
// code
}
```

Assertions and inheritance

```
class InternationalAirport: Airport {  
    ... ;  
    [ContractInvariantMethod]  
    void ObjectInvariant ()  
    { Contract.Invariant(this.numOfRunways >= 10);  
      Contract.Invariant(Contract.Exists(this.runways,  
        r => r.international=true))  
    // dummy code  
    }
```

Assertions and inheritance

```
void closeRunway(Runway strip)
{ Contract.Ensures(this.noOfRunways >= 10);
  Contract.Ensures(Contract.Exist(this.runways,
    r => r.international=true))
// code
}
```

Assertions for interfaces

- Specifying interfaces
- Contract classes for interfaces

Assertions for interfaces

```
[ContractClass(typeof(ContractforIRegistrar))]  
interface IRegistrar  
{ void scheduleCourse(Course c);  
  void deleteCourse(Course c);  
}
```

Assertions for interfaces

```
[ContractClassFor(typeof(IRegistrar))]  
sealed class ContractforIRegistrar: IRegistrar  
{ void scheduleCourse(Course c)  
{ Contract.Requires(!this.getAllCourses().Contains(c));  
  Contract.Requires(Contract.Exists(  
    this.getAllRooms(), r => r.suitableFor(c))  
  Contract.Ensures(this.getAllCourses().Contains(c));  
// dummy code  
}
```

Assertions for interfaces

```
void deleteCourse(Course c)
{ Contract.Requires(this.getAllCourses().Contains(c));
  Contract.Ensures(!this.getAllCourses().Contains(c));
  // dummy code
}
```


Assertions for interfaces

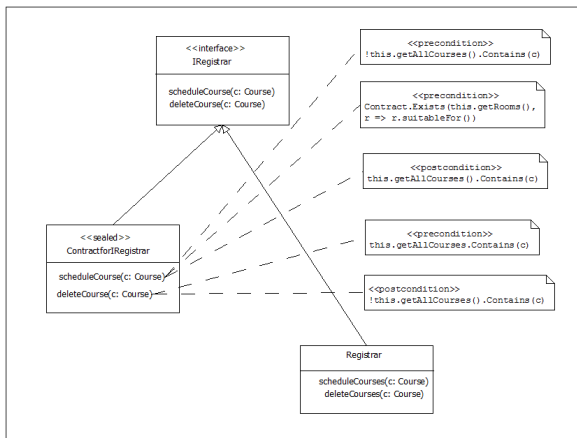


Figure: Constraints for the interface IRegistrar

Assertions for interfaces

```
[ContractClass(typeof(ContractforIStudent))]
```

```
interface IStudent
```

```
{ void enrollInCourse(Course c);
```

```
  void dropCourse(Course c);
```

```
}
```

Assertions for interfaces

```
[ContractClassFor(typeof(IStudent))]  
sealed class ContractforIStudent: IStudent  
void enrollInCourse(Course c)  
{ Contract.Requires(!this.getMyCourses().Contains(c));  
  ContractRequires(Contract.ForAll(  
    c.getPrerequisites(), p => this.getMyCourses().Contains(p)));  
  Contract.Ensures(this.getMyCourses().Contains(c));  
// dummy code  
}
```

Assertions for interfaces

```
void dropCourse(Course c)
{ Contract.Requires(this.getMyCourses().Contains(c));
  Contract.Ensures(!this.getMyCourses().Contains(c));
  // dummycode
}
```

Assertions for interfaces

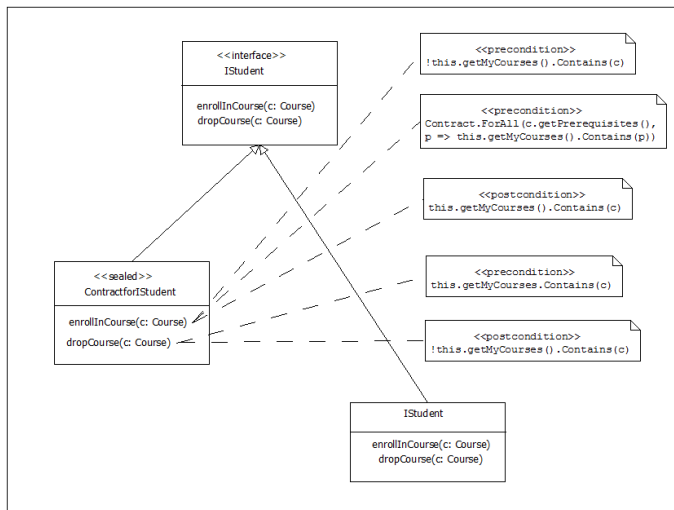


Figure: Constraints for interface `IStudent`

Sample application

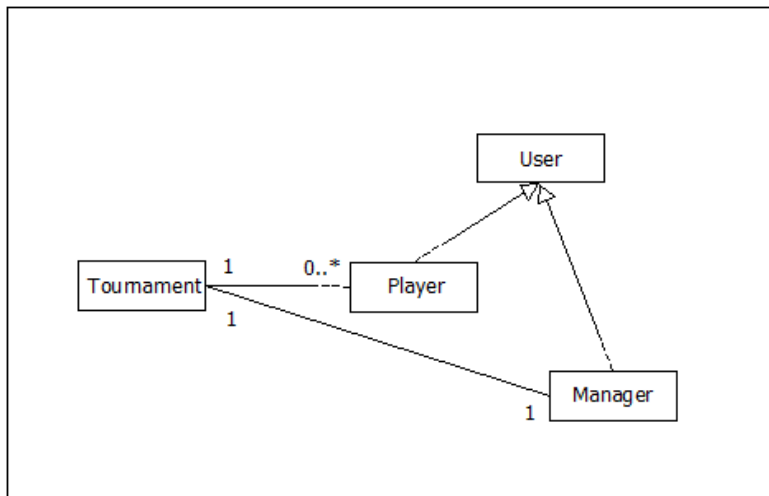


Figure: Tournament management application

Sample application

```
class Tournament {  
    String name;  
    Manager manager;  
    List<Player> players = new List<Player>();  
    // other fields  
    // constructor  
    public String Name  
    {  
        get { return name; }  
        set { name = value; }  
    }  
    // other properties  
}
```

Sample application

```
abstract class User {  
    String IDNum;  
    String name;  
    String role;  
    [ContractInvariantMethod]  
    void ObjectInvariant() {  
        Contract.Invariant(this.UserName != null);  
        Contract.Invariant(this.ID != null);  
    }  
}
```


Sample application

```
public String ID
{
    get { return IDNum; }
    set { IDNum = value; }
}
public String UserName
{
    get { return name; }
    set { user = value; }
}
// Role property
}
```

Sample application

```
class Player : User {  
    int winCount;  
    public int WinCount  
    {  
        get { return winCount; }  
        set { winCount = value; }  
    }  
    //other properties  
    [ContractInvariantMethod]  
    void ObjectInvariant() {  
        Contract.Invariant(this.WinCount >= 0);  
    }  
    // constructor and other methods  
}
```

Sample application

```
class Manager : User {  
    // fields  
    [ContractInvariantMethod]  
    void ObjectInvariant() {  
    { Contract.Invariant(Role.ToUpper().Contains("MANAGER"));  
    }  
    // methods  
}
```

Sample application

```
[Pure]
public boolean playerRegistered(Player newPlayer,
    Tournament tournament) {
    foreach (Player player in tournament.players)
    { if (newPlayer.UserName.ToUpper().Equals
        (player.UserName.ToUpper()))
        return true;
    }
    return false;
}
```

Sample application

```
public void addPlayer(Player newPlayer, Tournament tournament) {  
    Contract.Requires(newPlayer != null);  
    Contract.Requires(tournament != null);  
    Contract.Requires(! playerRegistered(newPlayer, tournament));  
    Contract.Ensures(playerRegistered(newPlayer, tournament));  
    Contract.Ensures((tournament.Players.Count) =  
        (Contract.OldValue(tournament.Players.Count) + 1));  
    tournament.Players.Add(newPlayer);  
}
```

Sample application

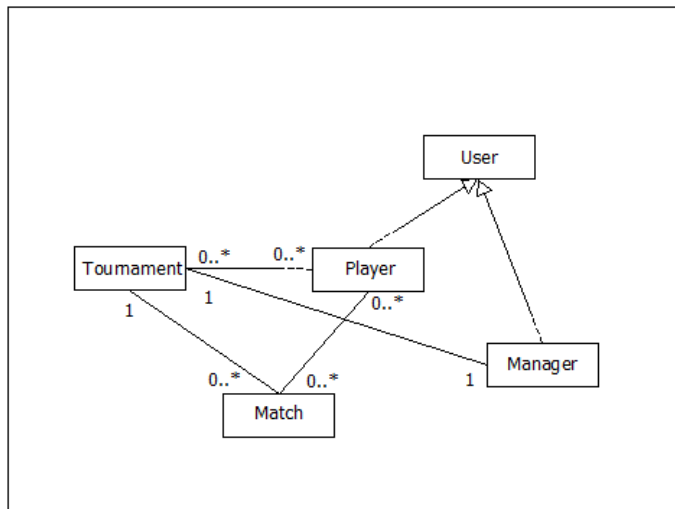


Figure: Tournaments with matches

Transactions and integrity

- Schema integrity constraints
- Enforcing integrity constraints
- Execution of transactions with constraints
- Transaction verification environment

Transaction verification

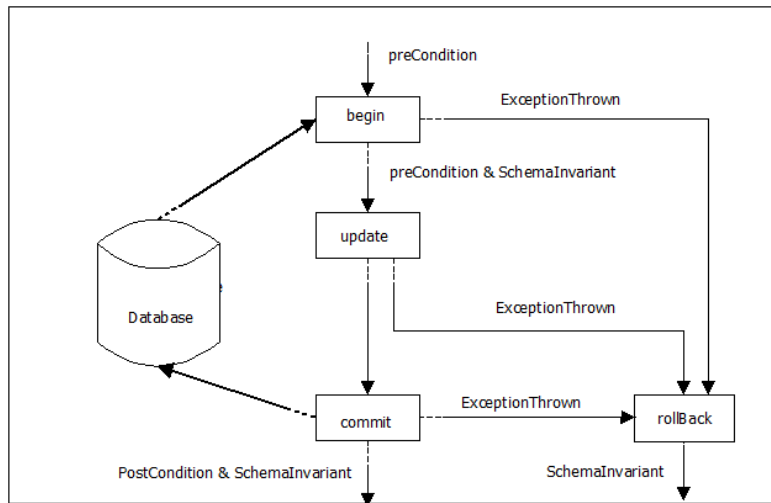


Figure: Transaction execution

Transaction verification

```
public abstract class Schema {  
  [Pure]  
  public static abstract bool integrityConstraints();  
}
```

Transaction verification

```
public abstract class Transaction<T> where T: Schema {  
  [Pure]  
  public abstract bool preCondition();  
  [Pure]  
  public abstract bool postCondition();  
  public sealed begin() {  
    Contract.Requires(this.preCondition());  
    Contract.Ensures(this.preCondition());  
    Contract.Ensures(T.integrityConstraints());  
  }  
  // system implementation  
}
```

Transaction verification

```
public sealed commit() {  
    Contract.Ensures(this.postCondition());  
    Contract.Ensures(T.integrityConstraints());  
    // system implementation  
}  
  
public sealed rollBack() {  
    Contract.Ensures(T.integrityConstraints());  
    // system implementation  
}  
}
```

Transaction verification

```
public class FlightSchedule: Schema {  
  public Table<Flight> flights;  
  [Pure]  
  public static abstract bool integrityConstraints() {  
    return(Contract.ForAll(this.flights,  
      (f1,f2) => (f1.flightId != f2.flightId || f1.Equals(f2)));  
  }  
  
  class Flight {  
    String flightId;  
    . . .  
  }  
}
```

Transaction verification

```
public class ScheduleFlight: Transaction<FlightSchedule> {  
  [Pure]  
  public bool precondition() {  
    Flight newFlight = get new flight;  
    return Contract.ForAll(FlightSchedule.flights,  
      f => f.flightId != newFlight.flightId)  
  }  
  [Pure]  
  public bool postCondition() {  
    Flight newFlight = get new flight;  
    return Contract.Exists(FlightSchedule.flights,  
      f => f.flightId = newFlight.flightId)  
  }  
}
```

Transaction verification

```
public void schedule(Flight newFlight) {  
    Contract.Requires(this.preCondition());  
    Contract.Ensures(this.postCondition());  
    // code  
}
```

Transaction verification

```
Flight newFlight = . . .  
ScheduleFlight Tx = new ScheduleFlight();  
try {  
    Tx.begin();  
    Tx.schedule(newFlight);  
    Tx.commit();  
}  
catch (Exception ex)  
    { Tx.rollback;}
```

- Unified assertion and programming languages
- Automatic static verification
- Specifying schemas with integrity constraints
- Universal and existential constraints
- Specifying schema methods with constraints
- Specifying transactions

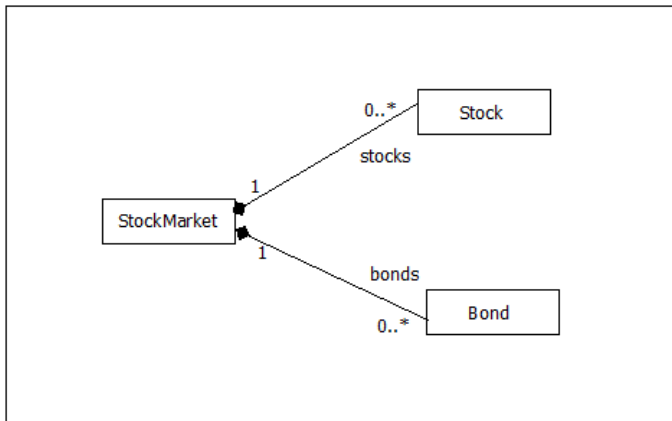


Figure: Associations for the stock market application

```
class StockMarket: Schema {  
  [SpecPublic][Rep] private Set<Stock> stocks;  
  [SpecPublic][Rep] private Set<Broker> brokers;  
  // . . .  
}
```

invariant

forall {Stock s1,s2 **in** stocks:
(s1.stockId()==s2.stockId()) **implies** s1.equals(s2) };

invariant

forall {Broker b1,b2 **in** brokers:
(b1.brokerId()== b2.brokerId()) **implies** b1.equals(b2) };

```
class Stock {  
    string stockId();  
    float price();  
    // public methods  
}  
  
class Broker {  
    string brokerId();  
    string name();  
    Set<Stock> stocks();  
    // public methods  
}
```

```
class StockMarket: Schema {  
  [SpecPublic][Rep] private Set<Stock> stocks;  
  [SpecPublic][Rep] private Set<Broker> brokers;  
  invariant  
  forall {Stock s1,s2 in stocks:  
    (s1.stockId()=s2.stockId()) implies s1.equals(s2) };  
  invariant  
  forall {Broker b1,b2 in brokers:  
    (b1.brokerId()= b2.brokerId()) implies b1.equals(b2) };  
  invariant  
  forall {Broker b in brokers: forall {Stock sb in b.stocks():  
    exists {Stock s in stocks: (sb.stockId() = s.stockId()) }  
    };  
  // public methods for insertions, updates, and deletions  
  // of stocks and brokers  
}
```