Formal Methods in Software Engineering

Master in Informatics and Computing Engineering

4th Year - 1st Semester

Formal Modelling of LinkedIn in VDM++

Daniel Silva Reis - up201308586@fe.up.pt

José Miguel Botelho Mendes - up201304828@fe.up.pt

Luís Ramos Pinto de Figueiredo - up201303501@fe.up.pt

11/12/2016

Table of Contents

1 Informal System Description and List of Requirements

1.1 Informal System Description

This project aims to model a working network akin to the “LinkedIn” social network. The project includes a client interface to interact with the defined LinkedIn model, allowing the client to test it.

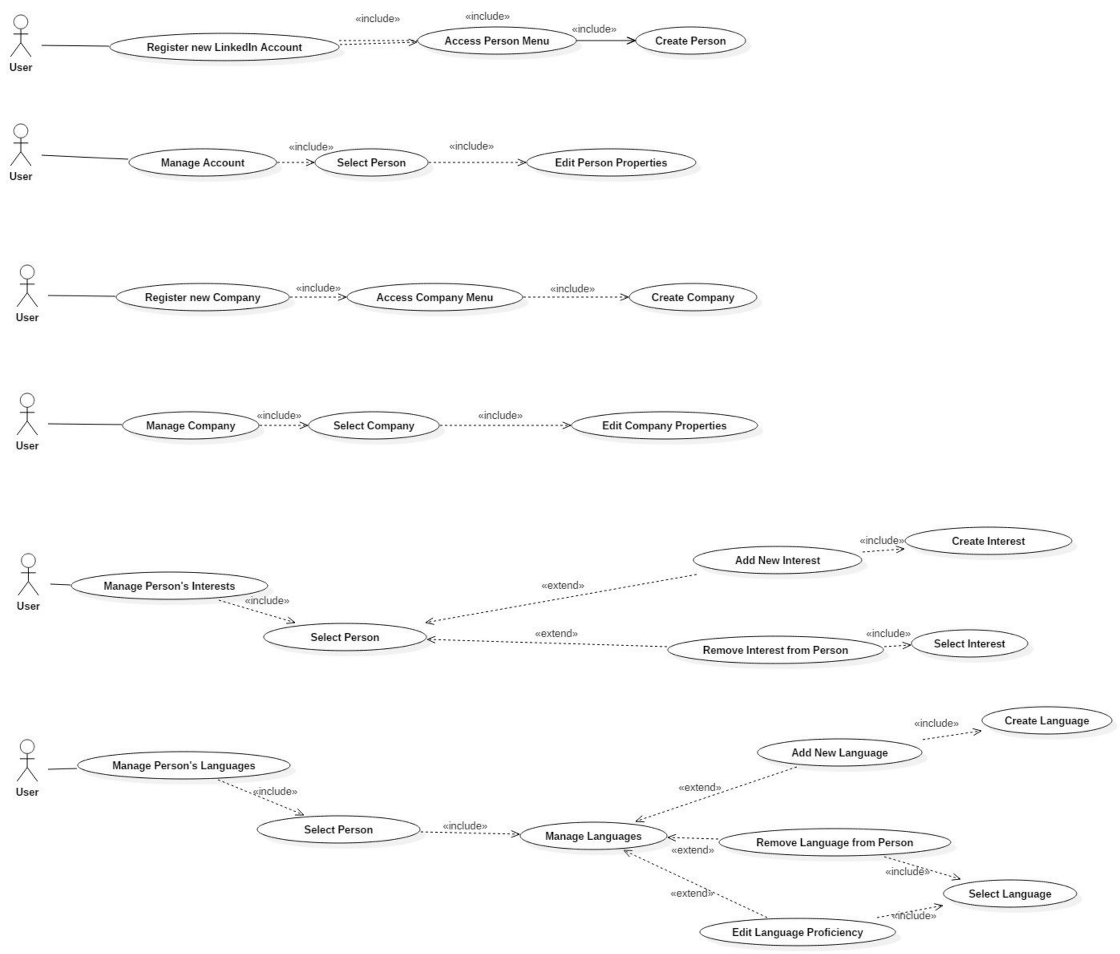
1.2 List of Requirements

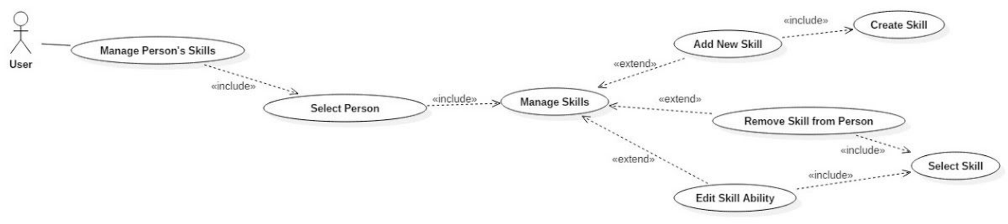
Requirement	Priority	Description
R1	Mandatory	The user can create and add a person
R2	Mandatory	The user can create and add a company
R3	Mandatory	A person can create an interest
R4	Mandatory	A person can add and remove his/her interests
R5	Mandatory	A person can create a skill
R6	Mandatory	A person can add and remove his/her skills
R7	Mandatory	The user can create a workplace
R8	Mandatory	A person can add and remove places he/she worked at
R9	Mandatory	A person can create an education
R10	Mandatory	A person can add and remove his/her educations
R11	Mandatory	A person can create a language
R12	Mandatory	A person can add and remove languages he speaks
R13	Mandatory	A person can add and remove his/her connections

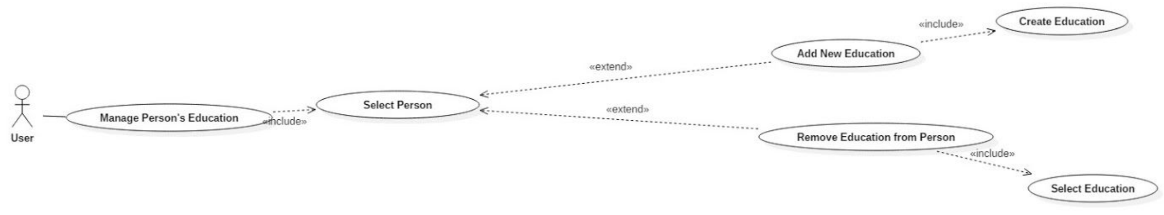
		to other users
R14	Mandatory	A person can find out the common contacts between himself and someone else
R15	Mandatory	A person can search and find a person on the network by name
R16	Mandatory	A person can find the minimum distance in users between himself and someone else
R17	Mandatory	A person can access all of his/her properties in his profile and edit them
R18	Mandatory	A person can get a language from his language set and edit his/her proficiency at it
R19	Mandatory	A person can get a skill from his skill set and edit his/her ability at it
R20	Mandatory	A person can get an interest from his interest set
R21	Mandatory	A person can get an education from his education set and edit its end date.
R22	Mandatory	A person can get a work from his work set and edit its end date
R23	Mandatory	A person can get a company from his company set and edit its description, address and type
R24	Mandatory	The user can find the average minimum distance between all people in the network
R25	Mandatory	The user can find the user in the network that has the most connections

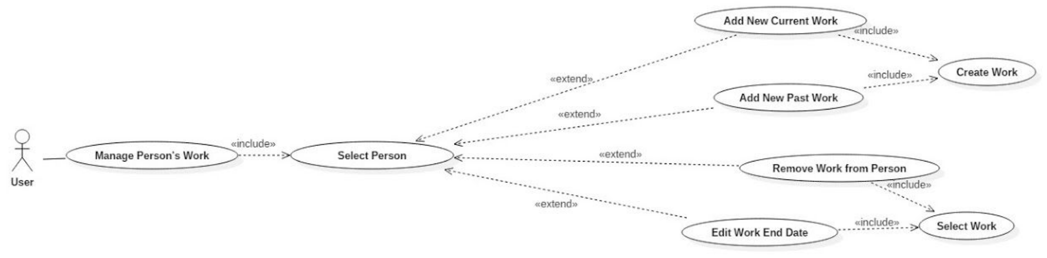
2 Visual UML Model

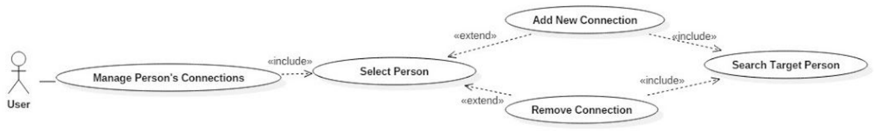
2.1 Use Case Model

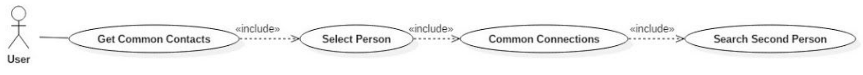


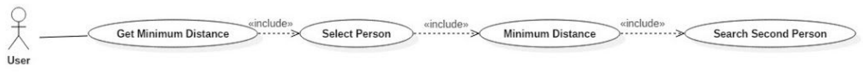


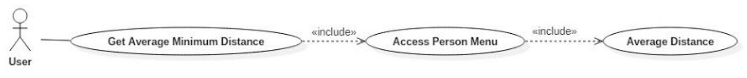


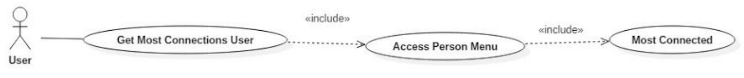












The major use case scenarios are described next.

Scenario	**Register New Account**
Description	Register a new account on the network
Pre-Conditions	The new account cannot already exist
Post-Conditions	The new account was added to the network
Steps	1. Select the Person Menu 2. Choose the “Create Person” option 3. Input the name 4. Submit
Exceptions	1. The user went back to the previous menu 2. The account already exists (step 4)

Scenario	**Manage Account**
Description	Edit a person’s account properties
Pre-Conditions	The account exists
Post-Conditions	The account was edited
Steps	1. Select the Person Menu 2. Choose the “Search Person” or the “List people” option 3. Select or search for wanted person 4. Select “Set Status”, “Set Description” or “Set CV” 5. Input your new information
Exceptions	1. The user went back to the previous menu

Scenario	**Register New Company**
Description	Registers a new company on the network
Pre-Conditions	The new company cannot already exist
Post-Conditions	The new company was added to the network
Steps	1. Select the Company Menu 2. Choose the “create company” option 3. Define company properties 4. Submit
Exceptions	1. The user went back to the previous menu 2. The company already exists (step 4)

Scenario	**Manage Company**
Description	Edit a company’s properties
Pre-Conditions	The company exists
Post-Conditions	The company was edited
Steps	1. Select the Company Menu 2. Choose the “List Companies” or “Search Company” option 3. Select or search for wanted company 4. Select “Type”, “Description” or “Address” 5. Input your new information
Exceptions	1. The user went back to the previous menu

Scenario	**Manage Person’s Interests**
Description	Manage a Person’s interests
Pre-Conditions	The person exists Option 1: The person doesn’t have the interest Option 2: The person has at least one interest
Post-Conditions	Option 1 : The interest was added Option 2: The interest was removed
Steps	1. Select the Person Menu 2. Choose the “List People” or “Search Person” option 3. Select or search for wanted person **Option 1**: “Add Interest” -> Create Interest -> Define Interest Name -> Submit **Option 2**: “Remove Interest” -> Select Interest from list
Exceptions	1. The user went back to the previous menu 2. Option 1 -> submit: The user already has that interest 3. Option 2 -> Select Interest: The user has no interests

Scenario	**Manage Person’s Languages**
Description	Manage a Person’s languages
Pre-Conditions	The person exists Option 1: The person doesn’t have the language Option 2: The person has at least one language Option 3: The person has at least one language
Post-Conditions	Option 1 : The language was added Option 2: The language was removed Option 3: The language was edited with the new properties
Steps	1. Select the Person Menu 2. Choose the “List People” or “Search Person” option 3. Select or search for wanted person 4. Select “Languages” **Option 1**: Add New Language -> Create Language -> Define language properties -> Submit **Option 2**: Remove Language -> Select Language from list **Option 3**: Edit Language -> Select Language from list -> Define new language properties
Exceptions	1. The user went back to the previous menu 2. Option 1 -> Submit: the person already has that language 3. Option 2 -> Select Language: the person has no language 4. Option 3 -> Select Language: the person has no language

Scenario	**Manage Person's Skills**
Description	Manage a Person's skills
Pre-Conditions	The person exists Option 1: The person doesn't have the skill Option 2: The person has at least one skill Option 3: The person has at least one skill
Post-Conditions	Option 1 : The skill was added Option 2: The skill was removed Option 3: The skill was edited with the new properties
Steps	1. Select the Person Menu 2. Choose the "List People" or "Search Person" option 3. Select or search for wanted person 4. Select "Skills" **Option 1**: Add New Skill -> Create Skill -> Define skill properties -> Submit **Option 2**: Remove Skill -> Select Skill from list **Option 3**: Edit Skill -> Select Skill from list -> Define new skill properties
Exceptions	1. The user went back to the previous menu 2. Option 1 -> Submit: the person already has that skill 3. Option 2 -> Select Skill: the person has no skill 4. Option 3 -> Select Skill: the person has no skill

Scenario	**Manage Person’s Education**
Description	Manage a Person’s educations
Pre-Conditions	The person exists Option 1: The person doesn’t have the education Option 2: The person has at least one education
Post-Conditions	Option 1 : The education was added Option 2: The education was removed
Steps	1. Select the Person Menu 2. Choose the “List People” or “Search Person” option 3. Select or search for wanted person **Option 1**:”Add Education” -> Create Education -> Define education properties -> Submit **Option 2**: “Remove Education” -> Select Education from list
Exceptions	1. The user went back to the previous menu 2. Option 1 -> Submit: the person already has that education 3. Option 2 -> Select Education: the person has no education

Scenario	**Manage Person’s Work**
Description	Manage a Person’s works
Pre-Conditions	The person exists Option 1: The person doesn’t have the work in his past works Option 1: The person is currently unemployed Option 2: The person has at least one work in his past works Option 3: The person has at least one work in his past works
Post-Conditions	Option 1 : The work was added Option 2: The work was added Option 2: The status changed to Employed Option 3: The work was removed
Steps	1. Select the Person Menu 2. Choose the “List People” or “Search Person” option 3. Select or search for wanted person **Option 1**: Add New Current Work -> Create Work -> Define work properties **Option 2**: Add New Past Work -> Create Work -> Define Work properties **Option 3**: Remove Past Work -> Select Work from list
Exceptions	1. The user went back to the previous menu 2. Option 1 -> Create New Work: the user is already employed 3. Option 1 -> Submit: the person already has that language 4. Option 2 -> Select Work: the person has no past work 5. Option 3 -> Select Work: the person is unemployed and/or has no past work

Scenario	**Manage Person’s Connections**
Description	Manage a Person’s connections. A connection binds two people together.
Pre-Conditions	The person exists The target person isn’t the same as the original Option 1: The connection doesn’t exist yet Option 2: The connection already exists
Post-Conditions	Option 1 : The connection was added to the first person Option 2: The connection was removed from the first person
Steps	1. Select the Person Menu 2. Choose the “List People” or “Search Person” option 3. Select or search for wanted person **Option 1**: Add New Connection -> Select Person from list **Option 2**: Remove Connection -> Select Person from list
Exceptions	1. The user went back to the previous menu 2. Option 1 -> Select Person: There is no person to target 3. Option 2 -> Select Person: There is no person to target

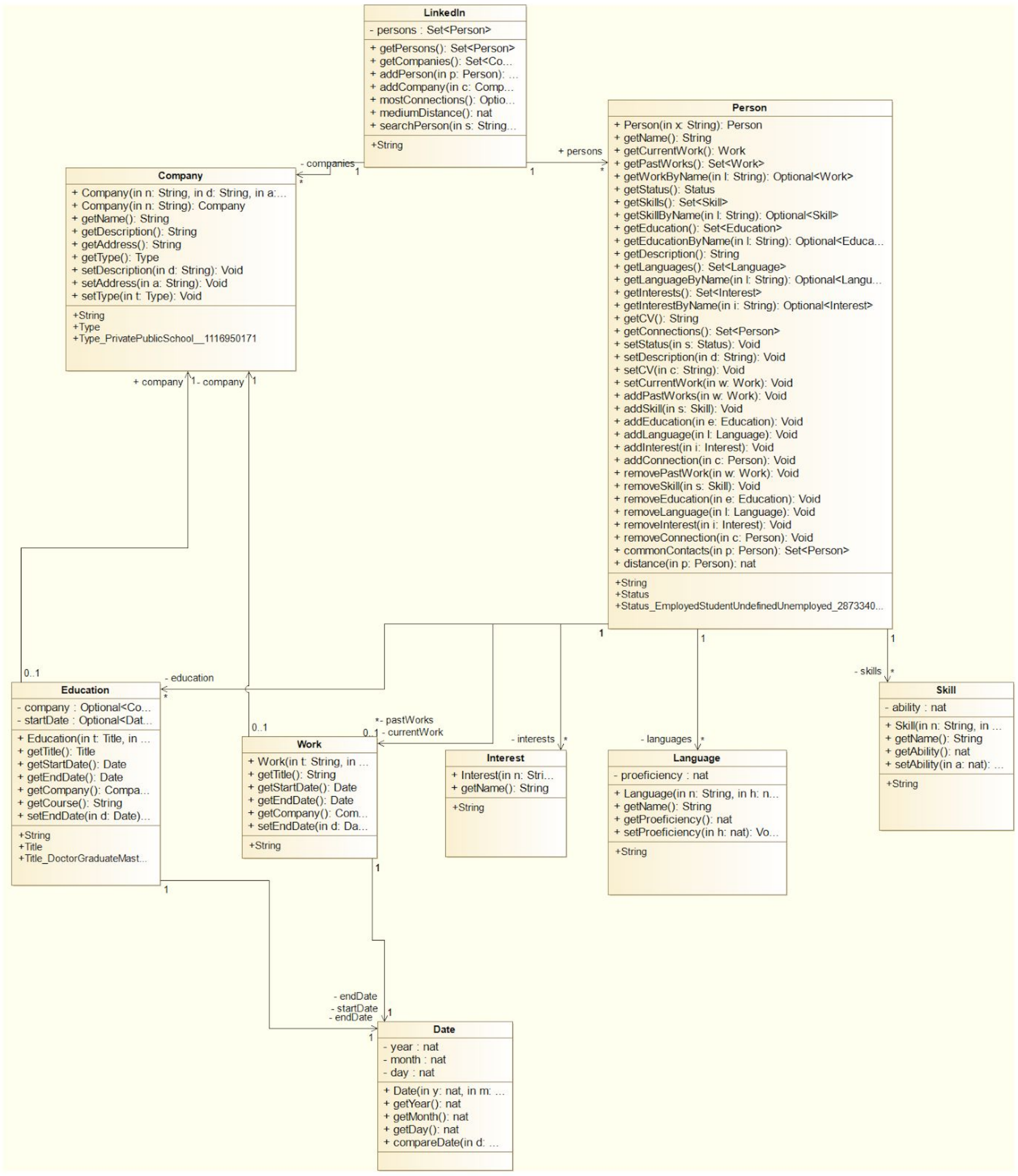
Scenario	**Get Common Contacts**
Description	See the common contacts between yourself and someone else
Pre-Conditions	The person exists The target person is different from the original
Post-Conditions	---
Steps	1. Select the Person Menu 2. Choose the “List People” or “Search Person” option 3. Select or search for wanted person 4. Select “Common Connections” 5. Select target person from list
Exceptions	1. The user went back to the previous menu 2. There is no person to target (step 5)

Scenario	**Get Minimum Distance**
Description	See the minimum distance between yourself and someone else
Pre-Conditions	The person exists
Post-Conditions	---
Steps	1. Select the Person Menu 2. Choose the “List People” or “Search Person” option 3. Select or search for wanted person 4. Select “Minimum Distance” 5. Select target person from list
Exceptions	1. There are less than 2 people in the network (returns -1) 2. There is no distance possible between them (returns -1)

Scenario	**Get Average Minimum Distance**
Description	See the average minimum distance between the people in the network
Pre-Conditions	There are at least 2 people in the network There is at least 1 connection in the network
Post-Conditions	---
Steps	1. Select the Person Menu 2. Choose the “Average Distance” option
Exceptions	1. There are less than 2 people in the network (outputs -1) 2. There are no connections in the network (outputs -1)

Scenario	**Get Most Connected User**
Description	See the user that has the most connections in the network
Pre-Conditions	There is at least 1 user in the network
Post-Conditions	---
Steps	1. Select the Person Menu 2. Choose the “Most Connected” option
Exceptions	1. There is no user (outputs null) 2. There are no connections in the network (outputs the last user)

2.2 Class Model


LinkedIn
- persons : Set<Person>
+ getPersons(): Set<Person>
+ getCompanies(): Set<Co...
+ addPerson(in p: Person): ...
+ addCompany(in c: Comp...
+ mostConnections(): Optio...
+ mediumDistance(): nat
+ searchPerson(in s: String...
+String
- companies
+ persons
Company
+ Company(in n: String, in d: String, in a:...
+ Company(in n: String): Company
+ getName(): String
+ getDescription(): String
+ getAddress(): String
+ getType(): Type
+ setDescription(in d: String): Void
+ setAddress(in a: String): Void
+ setType(in t: Type): Void
+String
+Type
+Type_PrivatePublicSchool__1116950171
+ company
- company
Person
+ Person(in x: String): Person
+ getName(): String
+ getCurrentWork(): Work
+ getPastWorks(): Set<Work>
+ getWorkByName(in l: String): Optional<Work>
+ getStatus(): Status
+ getSkills(): Set<Skill>
+ getSkillByName(in l: String): Optional<Skill>
+ getEducation(): Set<Education>
+ getEducationByName(in l: String): Optional<Educa...
+ getDescription(): String
+ getLanguages(): Set<Language>
+ getLanguageByName(in l: String): Optional<Langu...
+ getInterests(): Set<Interest>
+ getInterestByName(in i: String): Optional<Interest>
+ getCV(): String
+ getConnections(): Set<Person>
+ setStatus(in s: Status): Void
+ setDescription(in d: String): Void
+ setCV(in c: String): Void
+ setCurrentWork(in w: Work): Void
+ addPastWorks(in w: Work): Void
+ addSkill(in s: Skill): Void
+ addEducation(in e: Education): Void
+ addLanguage(in l: Language): Void
+ addInterest(in i: Interest): Void
+ addConnection(in c: Person): Void
+ removePastWork(in w: Work): Void
+ removeSkill(in s: Skill): Void
+ removeEducation(in e: Education): Void
+ removeLanguage(in l: Language): Void
+ removeInterest(in i: Interest): Void
+ removeConnection(in c: Person): Void
+ commonContacts(in p: Person): Set<Person>
+ distance(in p: Person): nat
+String
+Status
+Status_EmployedStudentUndefinedUnemployed_2873340...
- education
- skills
Education
- company : Optional<Co...
- startDate : Optional<Dat...
+ Education(in t: Title, in ...
+ getTitle(): Title
+ getStartDate(): Date
+ getEndDate(): Date
+ getCompany(): Compa...
+ getCourse(): String
+ setEndDate(in d: Date)...
+String
+Title
+Title_DoctorGraduateMast...
- pastWorks
- currentWork
Work
+ Work(in t: String, in ...
+ getTitle(): String
+ getStartDate(): Date
+ getEndDate(): Date
+ getCompany(): Com...
+ setEndDate(in d: Da...
+String
- interests
Interest
+ Interest(in n: Stri...
+ getName(): String
+String
- languages
Language
- proeficiency : nat
+ Language(in n: String, in h: n...
+ getName(): String
+ getProeficiency(): nat
+ setProeficiency(in h: nat): Vo...
+String
Skill
- ability : nat
+ Skill(in n: String, in ...
+ getName(): String
+ getAbility(): nat
+ setAbility(in a: nat): ...
+String
- endDate
- startDate
- endDate
Date
- year : nat
- month : nat
- day : nat
+ Date(in y: nat, in m: ...
+ getYear(): nat
+ getMonth(): nat
+ getDay(): nat
+ compareDate(in d: ...

Class	Description
LinkedIn	The main class, represents the LinkedIn network. Holds a set of people and companies, as well as functions to get, add and remove them. Also contains some utility functions, such as: search for a person, get the most connected user and the average distance in the network

Person	Class representative of a person. A person has a name, a description, a workplace (or not), the past places he/she worked at, a skillset, a language set, an education set, a set of his/her interests, a CV and a set of connections. All of these sets can be changed.
Company	Class representative of a company. A company has a name, a description, an address and a type. These can be edited, other than the name. A company type can be either “School”, “Private” or “Public”.
Date	Utility class that represents a date. Allows for easier abstraction of a date and has some utility functions, namely, comparing two dates.
Education	Class representative of an education. A user can add an education to his education set. An education has a title (Graduate, Master, …), a start date, an end date, a course name and finally, a company (typically a school, but not always). Can be read as “This person achieved title “title” at the course “course” in the company “company”, between “start date” and “end date” “.
Interest	Class representative of an interest. A user can add an interest to his interest set. An interest has a name, which works as its identifier and description. Can be read as “This person is interested in “name” “.
Language	Class representative of a language. A user can add a language to his language set. A language has a name and a proficiency level, which represents how good the person is at that language in his opinion. Can be read as “This person has achieved proficiency level “proficiency” in “language” “
Skill	Class representative of a skill. A user can add a skill to his skillset. A skill has a name and an ability level, which represents how good the person is at that skill in his opinion.
Work	Class representative of a work place. A user can add a work to his current work if he is unemployed or add a work to his past workplaces.

3 Formal VDM++ model

3.1 Class Company

```
class Company
       types
              public String = seq of char;

              -- This is the type of the company. It can either be a School, a
Private Company, or a Public one.
              public Type = <School> | <Private> | <Public>;

       instance variables
              private name : String;
              private description : String;
              private address : String;
              private type : Type;

       operations
              -- Create a new Company with all given parameters --
              public Company: String * String * String * Type ==> Company
      Company(n, d, a, t) == (name := n; description := d; address := a; type := t;
return self)
      pre n <> [] and d <> [] and a <> [];

              -- Create a new company only with a given name --
              public Company : String ==> Company
                     Company(n) == (name := n; return self)
                     pre n <> [];
              --START GETS--

              -- Returns the name of the company --
              public getName : () ==> String
                     getName() == (return name);

              -- Returns the description of the company --
              public getDescription : () ==> String
                     getDescription() == (return description);

              -- Returns the address of the company --
              public getAddress : () ==> String
                     getAddress() == (return address);

              -- Returns the type of the company --
              pure public getType : () ==> Type
                     getType() == (return type);
              --END GETS--
```

```
                --START SETS--

                -- Changes the description of the company --
                public setDescription : String ==> ()
                        setDescription(d) == (description := d; return);

                -- Changes the address of the company --
                public setAddress : String ==> ()
                        setAddress(a) == (address := a; return);

                -- Changes the type of the company --
                public setType : Type ==> ()
                        setType(t) == (type := t; return);
                --END SETS--
end Company
```

3.2 Class Date

```
class Date

        types

        instance variables
                private year : nat;
                private month : nat;
                private day : nat;
        operations

                -- Creates a new date given its year, month and day. --
                public Date: nat * nat * nat ==> Date
                Date(y, m, d) == (year := y; month := m; day := d; return self)
                pre y > 0 and m > 0 and m <= 12 and d > 0 and d <= 31;

                --START GETS--

                -- Returns the year of the date --
                pure public getYear : () ==> nat
                        getYear() == (return year);

                -- Returns the month of the date --
                pure public getMonth : () ==> nat
                        getMonth() == (return month);

                -- Returns the day of the date --
                pure public getDay : () ==> nat
                        getDay() == (return day);
                --END GETS--

                -- Compares two dates like date1.compareDate(date2). --
```

```
            -- If the date1 is "bigger" than the date 2, returns true. Else,
returns false --
            pure public compareDate : Date ==> bool
                  compareDate(d) == (
                        return ( (d.getYear() * d.getMonth() * d.getDay()) <
(year * month * day) )
                  );

end Date
```

3.3 Class Education

```
class Education
      types
            public String = seq of char;

            -- This title is the qualification received at the school.
            public Title = <Graduate> | <Master> | <Doctor> | <UnderGrad> |
<Student>

      instance variables
            private company : [Company];
            private title : Title;

            -- The date in which the education started. --
            private startDate : [Date];

            -- The date in which the education ended. --
            private endDate : Date;

            -- The course taken at the school. --
            private course : String;

            inv startDate <> nil;
            inv company <> nil;

      operations

            -- Creates a new Education, given its title, company, the date in
which it started and the course taken. --
            public Education: Title * Company * Date * String ==> Education
      Education(t, c, d, co) == (title := t; startDate := d; company := c; course :=
co; return self)
      pre co <> [] and c.getType() = <School>;
            --START GETS--

            -- Returns the title of the Education. --
```

```
		public getTitle : () ==> Title
			getTitle() == (return title);

		-- Returns the date in which the Education started. --

		public getStartDate : () ==> Date
			getStartDate() == (return startDate);

		-- Returns the date in which the Education ended. --
		public getEndDate : () ==> Date
			getEndDate() == (return endDate);

		-- Returns the company of the Education. --
		public getCompany : () ==> Company
			getCompany() == (return company);

		-- Returns the course of the Education. --
		public getCourse : () ==> String
			getCourse() == (return course);
		--END GETS--

		--START SETS--

		-- Sets the date in which the Education ended. --
		-- It checks if the end date is "bigger" than the date in which it
started. --
		public setEndDate : Date ==> ()
			setEndDate(d) == (endDate := d; return)
			pre d.compareDate(startDate);
		--END SETS--

end Education
```

3.4 Class Interest

```
class Interest
	types
		public String = seq of char;

	instance variables
		private name : String;
		-- for future implementations, like give it a rating, and all that
stuff --

	operations
		-- Creates a new Interest given its name. --
		public Interest: String ==> Interest
	Interest(n) == (name := n; return self)
	pre n <> [];
```

```
		--START GETS--

		-- Returns the name of the Interest. --
		public getName : () ==> String
			getName() == (return name);
		--END GETS--
end Interest
```

3.5 Class Language

```
class Language
	types
		public String = seq of char;

	instance variables
		private name : String;

		-- The proeficiency in the language. It goes from 1 to 10. --
		private proeficiency : nat;

	operations

		-- Creates a new Language, given its name and the proeficiency of the
user in it. --
		public Language: String * nat ==> Language
		  Language(n, h) == (name := n; proeficiency := h; return self)
		  pre h > 0 and h <= 10 and n <> [];
		--START GETS--

		-- Returns the name of the language. --
		public getName : () ==> String
			getName() == (return name);

		-- Returns the user's proeficiency in the language. --
		public getProeficiency : () ==> nat
			getProeficiency() == (return proeficiency);
		--END GETS--

		--START SETS--

		-- Changes the user's proeficiency in the given language. --
		public setProeficiency : nat ==> ()
			setProeficiency(h) == (proeficiency := h; return)
			pre h > 0 and h <= 10;
		--END SETS--
end Language
```

3.6 Class LinkedIn

```
class LinkedIn
      types
            public String = seq of char;
      instance variables

            -- All the people in LinkedIn. --
            private persons: set of Person:= {};

            -- All the companies in LinkedIn. --
            private companies: set of Company:= {};

      operations

            -- Get all the people in LinkedIn. --
            public getPersons : () ==> set of Person
                  getPersons() == (return persons);

            -- Get all the companies in LinkedIn. --
            public getCompanies : () ==> set of Company
                  getCompanies() == (return companies);

            -- Add a person to LinkedIn. --
            public addPerson: Person ==> ()
                  addPerson(p) == persons :=  {p} union persons
                  pre p not in set persons
                  post persons =  {p} union persons~;

            -- Add a company to LinkedIn. --
            public addCompany: Company ==> ()
                  addCompany(c) == companies :=  {c} union companies
                  pre c not in set companies
                  post companies =  {c} union companies~;

            -- Returns the person with the most connections in LinkedIn. --
            public mostConnections: () ==> [Person]
                  mostConnections() ==
                  (
                        dcl person : Person:= new Person();
                        if card persons > 0
                        then
                        (
                              for all p in set persons do
                              if card p.getConnections() > card
person.getConnections()
                                    then person := p;
                              return person
                        )
                        else return nil
                );

        -- Returns the medium distance between all the people in LinkedIn. --
        public mediumDistance: () ==> int
                mediumDistance() ==
                (
                        dcl num : int := 0;
                        dcl b : bool := false;
                        dcl den : nat := 0;
                        for all p in set persons do
                        (
                                for all p1 in set p.getConnections() do
                                (
                                        b := true;
                                        num := num + p1.distance(p);
                                        den := den + 1
                                );
                        );
                        if (not b)
                                then return -1
                        else return num / den;
                );

        -- Receives a string and returns the Person with the name or
description equal to the string. --
        public searchPerson: String ==> set of Person
                searchPerson(s)==
                (
                        dcl res: set of Person:= {};
                                for all p in set persons do
                                        if (p.getName() = s or
p.getDescription()=s)
                                                then (res:= {p} union res);
                        return res
                )
                pre s <> [];

end LinkedIn
```

3.7 Class Person

```
class Person
        types
                public String = seq of char;

                -- The status of the person, based on its work. --
                public Status = <Unemployed> | <Student> | <Employed> | <Undefined>;
        instance variables

                -- The name of the Person. --
                private name : String;

                -- The current Work of the Person. --
                private currentWork : [Work] := nil;

                -- A set of all the previous works of the Person. --
                private pastWorks : set of Work := {};
                private status : Status := <Undefined>;

                -- A set of the all skills the Person has. --
                private skills : set of Skill := {};

                -- A set of all the previous Schools the Person frequented. --
                private education : set of Education := {};
                private description : String := "";

                -- A set of all the languages the Person has added to its LinkedIn
Profile. --
                private languages : set of Language := {};

                -- A set of all the interests the Person has added to its LinkedIn
Profile. --
                private interests : set of Interest := {};
                private cv : String;

                -- The connections of a Person, as in, all of the other Persons he
"follows". --
                private connections : set of Person := {};

                inv {currentWork} inter pastWorks = {}

        operations

                -- Creates a new Person given its name. --
                public Person: String ==> Person
      Person(x) == (name := x; return self);

                --START GETS--

                -- Returns the name of the Person. --
                public getName : () ==> String
                        getName() == (return name);

                -- Returns the current work of the person. --
                public getCurrentWork : () ==> Work
                        getCurrentWork() == (return currentWork);

                -- Returns all of the Person's previous works. --
        public getPastWorks : () ==> set of Work
               getPastWorks() == (return pastWorks);

        -- Returns a work based on its name, from the set of pastWorks of the
Person. --
        public getWorkByName : String ==> [Work]
               getWorkByName(l) ==
               (
                      for all work in set pastWorks do
                             if l = work.getCompany().getName()
                                    then return work;
                      return nil
               );

        -- Returns the status of the Person. --
        public getStatus : () ==> Status
               getStatus() == (return status);

        -- Returns all of the Person's skills.
        public getSkills : () ==> set of Skill
               getSkills() == (return skills);

        -- Returns a skill based on its name, from the set of skills of the
Person. --
        public getSkillByName : String ==> [Skill]
               getSkillByName(l) ==
               (
                      for all skill in set skills do
                             if l = skill.getName()
                                    then return skill;
                      return nil
               );

        -- Returns all of the Person's schools he frequented. --
        public getEducation : () ==> set of Education
               getEducation() == (return education);

        -- Returns an education based on its name, from the set of educations
of the Person. --
        public getEducationByName : String ==> [Education]
               getEducationByName(l) ==
               (
                      for all e in set education do
                      (
                             if l = e.getCompany().getName()
                                    then return e
                      );
                      return nil
               );

        -- Returns a description of the Person. --
```

```
		public getDescription : () ==> String
			getDescription() == (return description);

		-- Returns all of the Person's languages. --
		public getLanguages : () ==> set of Language
			getLanguages() == (return languages);

		-- Returns a language based on its name, from the set of languages of
the Person. --
		public getLanguageByName : String ==> [Language]
			getLanguageByName(l) ==
			(
				for all language in set languages do
					if l = language.getName()
						then return language;
				return nil
			);

		-- Returns all of the Person's interests. --
		public getInterests : () ==> set of Interest
			getInterests() == (return interests);

		-- Returns an interest based on its name, from the set of interests of
the Person. --
		public getInterestByName : String ==> [Interest]
			getInterestByName(i) ==
			(
				dcl t : Interest := new Interest();
				for all interest in set interests do
					(if i = interest.getName()
						then return interest);
				return nil
			);

		-- Returns the Person's CV. --
		public getCV : () ==> String
			getCV() == (return cv);

		-- Returns all of the Person's connections. --
		public getConnections : () ==> set of Person
			getConnections() == (return connections);
		--END GETS--

		--START SETS--

		-- Changes the Person's status. --
		-- If the person goes from being employed, to unemployed, the current
work ends and it's added to the past works set. --
		public setStatus : Status ==> ()
			setStatus(s) ==
			(
                    if (status = <Employed> and s = <Unemployed>)
                    then
                    (
                           dcl data : Date := new Date(2017,12,30); -- ATÉ AO
PEITO

                           dcl temp : Work := currentWork;
                           currentWork := nil;
                           temp.setEndDate(data);
                           pastWorks:= pastWorks union {temp}
                    );

                    status := s;
                    return
             );

      -- Changes the Person's description. --
      public setDescription : String ==> ()
             setDescription(d) == (description := d; return);

      -- Changes the Person's CV. --
      public setCV : String ==> ()
             setCV(c) == (cv := c; return);

      public setCurrentWork : Work ==> ()
             setCurrentWork(w) == (currentWork := w; status := <Employed>;
return)

             pre status <> <Employed> and w not in set pastWorks;
      --END SETS--

      --START ADDS--

      -- Adds a past work to the existent set. --
      -- It checks if the work is not in the set and if it was indeed added.
--
      public addPastWorks : Work ==> ()
             addPastWorks(w) == (pastWorks := pastWorks union {w}; return)
             pre w not in set pastWorks and w <> currentWork
             post pastWorks = pastWorks~ union {w};

      -- Adds a skill to the existent set. --
      -- It checks if the skill is not in the set and if it was indeed
added. --
      public addSkill : Skill ==> ()
             addSkill(s) == (skills := skills union {s}; return)
             pre s not in set skills
             post  skills = skills~ union {s};


      -- Adds a education to the existent set. --
      -- It checks if the education is not in the set and if it was indeed
added. --
```

```
              public addEducation : Education ==> ()
                     addEducation(e) == (education := education union {e}; return)
                     pre e not in set education
                     post education = education~ union {e};


              -- Adds a language to the existent set. --
              -- It checks if the language is not in the set and if it was indeed
added. --
              public addLanguage : Language ==> ()
                     addLanguage(l) == (languages := languages union {l}; return)
                     pre l not in set languages
                     post languages = languages~ union {l};

              -- Adds an interest to the existent set. --
              -- It checks if the interest is not in the set and if it was indeed
added. --
              public addInterest : Interest ==> ()
                     addInterest(i) == (interests := interests union {i}; return)
                     pre i not in set interests
                     post interests = interests~ union {i};

              -- Adds a connection to the existent set. --
              -- It checks if the connection is not in the set and if it was indeed
added. --
              public addConnection : Person ==> ()
                     addConnection(c) == (connections := connections union {c};
return)

                     pre c not in set connections and c <> self
                     post connections = connections~ union {c};
              --END ADDS--

              --START REMOVES--

              -- Removes a past work from the existent set. --
              -- It checks if the work is in the set and if it was indeed removed.
--
              public removePastWork : Work ==> ()
                     removePastWork(w) == (pastWorks := pastWorks \ {w}; return)
                     pre w in set pastWorks
                     post {} = pastWorks inter {w};

              -- Removes a skill from the existent set. --
              -- It checks if the skill is in the set and if it was indeed removed.
--
              public removeSkill : Skill ==> ()
                     removeSkill(s) == (skills := skills \ {s}; return)
                     pre s in set skills
                     post {} = skills inter {s};

              -- Removes a education from the existent set. --
```

```
		-- It checks if the education is in the set and if it was indeed
removed. --
		public removeEducation : Education ==> ()
			removeEducation(e) == (education := education \ {e}; return)
			pre e in set education
			post {} = education inter {e};

		-- Removes a language from the existent set. --
		-- It checks if the language is in the set and if it was indeed
removed. --
		public removeLanguage : Language ==> ()
			removeLanguage(l) == (languages := languages \ {l}; return)
			pre l in set languages
			post {} = languages inter {l};

		-- Removes an interest from the existent set. --
		-- It checks if the interest is in the set and if it was indeed
removed. --
		public removeInterest : Interest ==> ()
			removeInterest(i) == (interests := interests \ {i}; return)
			pre i in set interests
			post {} = interests inter {};

		-- Removes a connection from the existent set. --
		-- It checks if the connection is in the set and if it was indeed
removed. --
		public removeConnection : Person ==> ()
			removeConnection(c) == (connections := connections \ {c};
return)

			pre c in set connections and c <> self
			post {} = connections inter {};
		--END REMOVES

		-- Returns all of the common contacts between two persons. --
		public commonContacts : Person ==> set of Person
			commonContacts(p) ==
			(
				return p.getConnections() inter connections
			)
			pre p <> self;

		-- Returns the distance from one Person to the other. --
		public distance : Person ==> int
			distance(p) ==
			(
				dcl distancia : int := 1;
				dcl temp : set of Person := connections;
				dcl visited : set of Person := {};

				while p not in set temp do
				(
                                                distancia := distancia + 1;

                                                for all t in set temp do
                                                (
                                                        visited := visited union {t};
                                                        temp := temp union t.getConnections() \
visited
                                                );
                                                if (card temp = 0)
                                                        then return -1;
                                        );
                                        return distancia
                                )
                                pre p <> self;
end Person
```

3.8 Class Skill

```
class Skill
        types
                public String = seq of char;

        instance variables
                private name : String;

                -- The user's ability in the Skill. It varies from 1 to 10. --
                private ability : nat;

        operations

                -- Creates a new Skill, given its name and the user's ability in it.
--
                public Skill: String * nat ==> Skill
       Skill(n, a) == (name := n; ability := a; return self)
       pre a > 0 and a <= 10 and n <> [];
                --START GETS--

                -- Returns the name of the Skill. --
                public getName : () ==> String
                        getName() == (return name);

                -- Returns the user's ability in the Skill. --
                public getAbility : () ==> nat
                        getAbility() == (return ability);
                --END GETS--

                --START SETS--

                -- Changes the user's ability in the Skill. --
```

```
			public setAbility : nat ==> ()
				setAbility(a) == (ability := a; return)
				pre a > 0 and a <= 10;
			--END SETS--
end Skill
```

3.9 Class Work

```
class Work
	types
		public String = seq of char;

	instance variables
		-- The title of the user in the company, the "job". --
		private title : String;

		-- The date in which the user started working in the company. --
		private startDate : [Date];

		-- The date in which the user quit working in the company. --
		private endDate : Date;

		-- The company in which the user worked. --
		private company : [Company];

		inv startDate <> nil;
		inv company <> nil;

	operations

		-- Creates a new Work, given its title, the date in which it started
and the company. --
		public Work: String * Date * Company ==> Work
	Work(t, d, c) == (title := t; startDate := d; company := c; return self)
	pre t <> [];
		--START GETS--

		-- Returns the title of the Work. --
		public getTitle : () ==> String
			getTitle() == (return title);

		-- Returns the start date of the Work. --
		public getStartDate : () ==> Date
			getStartDate() == (return startDate);

		-- Returns the end date of the Work. --
		public getEndDate : () ==> Date
			getEndDate() == (return endDate);
```

```
              -- Returns the company of the Work. --
              public getCompany : () ==> Company
                     getCompany() == (return company);
              --END GETS--

              --START SETS--

              -- Changes the date in which the user ended its Work. --
              public setEndDate : Date ==> ()
                     setEndDate(d) == (endDate := d; return)
                     pre d.compareDate(startDate);
              --END SETS--
end Work
```

4 Model Validation

4.1 Class CompanyTest

```
class CompanyTest is subclass of TestCaseLinkedIn

       instance variables
              cI : Company := new Company("Primavera ERP");
              cF : Company := new Company("Autoridade Tributaria e Aduaneira",
"Recolher Impostos", "Avenida da Liberdade, nº40, 4400-200", <Public>);

       operations

              -- START GETS --
              private testGetName: () ==> ()
              testGetName() ==
           (
             assertEqual(cI.getName(), "Primavera ERP");
             assertEqual(cF.getName(), "Autoridade Tributaria e Aduaneira");
           );
         private testGetDescription: () ==> ()
              testGetDescription() ==
              (
              assertEqual(cF.getDescription(), "Recolher Impostos");
              );
         private testGetAddress: () ==> ()
           testGetAddress() ==
           (
             assertEqual(cF.getAddress(), "Avenida da Liberdade, nº40, 4400-200");
           );
         private testGetType: () ==> ()
              testGetType() ==
              (
              assertEqual(cF.getType(), <Public>);
              );
         -- END GETS --

         -- START SETS --
         private testSetDescription: () ==> ()
              testSetDescription() ==
              (
              cI.setDescription("Faturas/Recibos");

              assertEqual(cI.getDescription(), "Faturas/Recibos");
              );
         private testSetAddress: () ==> ()
              testSetAddress() ==
              (
              cI.setAddress("Rua Doutor Roberto Frias");

              assertEqual(cI.getAddress(), "Rua Doutor Roberto Frias");
              );
          private testSetType: () ==> ()
              testSetType() ==
              (
              cI.setType(<School>);

              assertEqual(cI.getType(), <School>);
              );
          -- END SETS --

        public static main: () ==> ()
              main() ==
              (
                 dcl teste : CompanyTest := new CompanyTest();
              teste.testGetName();
              teste.testGetDescription();
              teste.testGetAddress();
              teste.testGetType();
              teste.testSetDescription();
              teste.testSetAddress();
              teste.testSetType();
           );

    end CompanyTest
```

4.2 Class DateTest

```
class DateTest is subclass of TestCaseLinkedIn

    instance variables
          dI : Date := new Date(2016, 12, 30);
          dF : Date := new Date(2016, 10, 30);

    operations

          private testCompareDate: () ==> ()
          testCompareDate() ==
          (
          assertEqual(dI.compareDate(dF), true);
          assertEqual(dF.compareDate(dI), false);
          );

      -- START GETS --
      private testGetYear: () ==> ()
          testGetYear() ==
          (
          assertEqual(dI.getYear(), 2016);
          assertEqual(dF.getYear(), 2016);
          );
     private testGetMonth: () ==> ()
          testGetMonth() ==
          (
          assertEqual(dI.getMonth(), 12);
          assertEqual(dF.getMonth(), 10);
          );
     private testGetDay: () ==> ()
          testGetDay() ==
          (
          assertEqual(dI.getDay(), 30);
          assertEqual(dF.getDay(), 30);
          );
     -- END GETS --

    public static main: () ==> ()
          main() ==
          (
             dcl teste : DateTest := new DateTest();
     teste.testCompareDate();
          teste.testGetYear();
          teste.testGetMonth();
          teste.testGetDay();
        );

end DateTest
```

4.3 Class EducationTest

```
class EducationTest is subclass of TestCaseLinkedIn

    instance variables

          cF : Company := new Company("Faculdade de Engenharia da Universidade
do Porto", "Formar Pessoas", "Rua Doutor Roberto Frias", <School>);
          cI : Company := new Company("Autoridade Tributaria e Aduaneira",
"Recolher Impostos", "Avenida da Liberdade, nº40, 4400-200", <School>);

          dF : Date := new Date(2016, 10, 30);
          dI : Date := new Date(2016, 10, 31);

          eF : Education := new Education(<Graduate>, cF, dF, "Mestrado
Integrado Em Engenharia Informatica e Computacao");
          eI : Education := new Education(<UnderGrad>, cI, dF, "Estagio Final");
```

```
operations

  -- START GETS --
       private testGetTitle: () ==> ()
       testGetTitle() ==
       (
       assertEqual(eI.getTitle(), <UnderGrad>);
       assertEqual(eF.getTitle(), <Graduate>);
       );
  private testGetStartDate: () ==> ()
       testGetStartDate() ==
       (
       assertEqual(eI.getStartDate(), dF);
       assertEqual(eF.getStartDate(), dF);
       );
  private testGetEndDate: () ==> ()
       testGetEndDate() ==
       (
       assertEqual(eI.getEndDate(), dI);
       assertEqual(eF.getEndDate(), dI);
       );
  private testGetCompany: () ==> ()
       testGetCompany() ==
       (
       assertEqual(eI.getCompany(), cI);
       assertEqual(eF.getCompany(), cF);
       );
private testGetCourse: () ==> ()
    testGetCourse() ==
      (
      assertEqual(eI.getCourse(), "Estagio Final");
      assertEqual(eF.getCourse(), "Mestrado Integrado Em Engenharia Informatica e
Computacao");
       );
  -- END GETS --

  -- START SETS --
  private testSetEndDate: () ==> ()
       testSetEndDate() ==
       (
       eI.setEndDate(dI);
       eF.setEndDate(dI);

       assertEqual(eI.getEndDate(), dI);
       assertEqual(eF.getEndDate(), dI);
       );
  -- END SETS --

public static main: () ==> ()
       main() ==
        (
           dcl teste : EducationTest := new EducationTest();
   teste.testGetTitle();
   teste.testGetStartDate();
   teste.testGetCompany();
   teste.testGetCourse();
   teste.testSetEndDate();
   teste.testGetEndDate();
        );

end EducationTest
```

4.4 Class InterestTest

```
class InterestTest is subclass of TestCaseLinkedIn

    instance variables
        lI : Interest := new Interest("Requirements Documents");
        lF : Interest := new Interest("GTA V");

    operations

     -- START GETS --
        private testGetName: () ==> ()
        testGetName() ==
        (
        assertEqual(lI.getName(), "Requirements Documents");
        assertEqual(lF.getName(), "GTA V");
        );
     -- END GETS --

    public static main: () ==> ()
        main() ==
        (
           dcl teste : InterestTest := new InterestTest();
   teste.testGetName();
      );

end InterestTest
```

4.5 Class LanguageTest

```
class LanguageTest is subclass of TestCaseLinkedIn

    instance variables
        lI : Language := new Language("Portuguese", 10);
        lF : Language := new Language("English", 8);

    operations

      -- START GETS --
        private testGetName: () ==> ()
        testGetName() ==
     (
       assertEqual(lI.getName(), "Portuguese");
       assertEqual(lF.getName(), "English");
        );
      private testGetProeficiency: () ==> ()
        testGetProeficiency() ==
        (
        assertEqual(lI.getProeficiency(), 10);
        assertEqual(lF.getProeficiency(), 8);
        );
      -- END GETS --

      -- START SETS --
      private testSetProeficiency: () ==> ()
        testSetProeficiency() ==
        (
               assertEqual(lI.getProeficiency(), 10);
        assertEqual(lF.getProeficiency(), 8);

        lI.setProeficiency(8);
        lF.setProeficiency(10);

        assertEqual(lI.getProeficiency(), 8);
        assertEqual(lF.getProeficiency(), 10);
        );
      -- END SETS --

    public static main: () ==> ()
        main() ==
        (
           dcl teste : LanguageTest := new LanguageTest();
   teste.testGetName();
        teste.testGetProeficiency();
        teste.testSetProeficiency();
        );

end LanguageTest
```

4.6 Class LinkedInTest

```
class LinkedInTest is subclass of TestCaseLinkedIn

       instance variables
              l : LinkedIn := new LinkedIn();
              p1 : Person := new Person("Luis Figueiredo");
              p2 : Person := new Person("Diogo Moura");
              p3 : Person := new Person("Antonio Ramadas");
              p4 : Person := new Person("Pedro Castro");
              p5 : Person := new Person("Paulino");
              p6 : Person := new Person("Cigano do Codigo");

              c1 : Company := new Company("Faculdade de Engenharia da Universidade
do Porto", "Formar Pessoas", "Rua Doutor Roberto Frias", <School>);
              c2 : Company := new Company("Autoridade Tributaria e Aduaneira",
"Recolher Impostos", "Avenida da Liberdade, nº40, 4400-200", <School>);

       operations

         -- START ADDS --
              private testAddPerson: () ==> ()
              testAddPerson() ==
              (
                 dcl temp : set of Person := {};
                 temp := temp union {p1};
                 temp := temp union {p2};
                 temp := temp union {p3};
                 temp := temp union {p4};
                 temp := temp union {p5};
                 temp := temp union {p6};

                 l.addPerson(p1);
                 l.addPerson(p2);
                 l.addPerson(p3);
                 l.addPerson(p4);
                 l.addPerson(p5);
                 l.addPerson(p6);

              assertEqual(l.getPersons(), temp)
              );
         private testAddCompany: () ==> ()
              testAddCompany() ==
              (
                 dcl temp : set of Company := {};
                 temp := temp union {c1};
                 temp := temp union {c2};

                 l.addCompany(c1);
                 l.addCompany(c2);

              assertEqual(l.getCompanies(), temp)
              );
```

```
-- END ADDS --

-- START GETS --
private testGetCompanies: () ==> ()
     testGetCompanies() ==
     (
             dcl temp : set of Company := {};
        temp := temp union {c1};
        temp := temp union {c2};
     assertEqual(l.getCompanies(), temp)
     );

private testGetPersons: () ==> ()
     testGetPersons() ==
     (
             dcl temp : set of Person := {};
        temp := temp union {p1};
        temp := temp union {p2};
        temp := temp union {p3};
        temp := temp union {p4};
        temp := temp union {p5};
        temp := temp union {p6};
     assertEqual(l.getPersons(), temp)
     );
-- END GETS --

private testMostConnections: () ==> ()
     testMostConnections() ==
     (
             assertEqual(l.mostConnections(), p1)
     );

private testMediumDistance : () ==> ()
     testMediumDistance() ==
     (
             assertEqual(l.mediumDistance(), -1)
     );

private testSearchPerson: () ==> ()
     testSearchPerson() ==
     (
             p1.setDescription("Preciso de comprar o GTA V");
             assertEqual(l.searchPerson("Luis Figueiredo"), {p1});
             assertEqual(l.searchPerson("Preciso de comprar o GTA V"), {p1})
     );

private testNoPeople: () ==> ()
     testNoPeople() ==
     (
             assertEqual(l.mostConnections(), nil);
             assertEqual(l.mediumDistance(), -1)
```

```
            );

        private testPopulateConnections: () ==> ()
            testPopulateConnections() ==
            (
              p1.addConnection(p2);
              p1.addConnection(p3);
              p1.addConnection(p6);
              p2.addConnection(p4);
              p2.addConnection(p5);
              p3.addConnection(p6)
            );

      public static main: () ==> ()
            main() ==
            (
               dcl teste : LinkedInTest := new LinkedInTest();

            teste.testNoPeople();
               teste.testPopulateConnections();

      teste.testAddCompany();
            teste.testAddPerson();
            teste.testGetCompanies();
            teste.testGetPersons();
            teste.testSearchPerson();

            teste.testMostConnections();
            teste.testMediumDistance()
            );

end LinkedInTest
```

4.7 Class PersonTest

```
class PersonTest is subclass of TestCaseLinkedIn
      instance variables
            pI : Person := new Person("Jose Miguel Botelho Mendes");
            pF : Person := new Person("Daniel da Silva Reis");
            p1 : Person := new Person("Luis Figueiredo");
            p2 : Person := new Person("Diogo Moura");
            p3 : Person := new Person("Antonio Ramadas");
            p4 : Person := new Person("Pedro Castro");
            p5 : Person := new Person("Paulino");
            p6 : Person := new Person("Cigano do Codigo");
            p7 : Person := new Person("teste123");
            cF : Company := new Company("Faculdade de Engenharia da Universidade
do Porto", "Formar Pessoas", "Rua Doutor Roberto Frias", <School>);
            cI : Company := new Company("Autoridade Tributaria e Aduaneira",
"Recolher Impostos", "Avenida da Liberdade, nº40, 4400-200", <Public>);

            birthdayDate : Date := new Date(1995, 11, 06);
            dF : Date := new Date(2016, 10, 30);
            dI : Date := new Date(2016, 10, 31);

            wF : Work := new Work("Balconista", dF, cF);
            wFF : Work := new Work("CEO", dF, cF);

            wI : Work := new Work("Balconista", dF, cI);
            wII : Work := new Work("CEO", dF, cI);

            s1 : Skill := new Skill("JAVA", 10);
            s2 : Skill := new Skill("Alloy", 2);
            s3 : Skill := new Skill("C", 5);
            s4 : Skill := new Skill("OpenGL", 4);

            c1 : Company := new Company("Faculdade de Engenharia da Universidade
do Porto", "Formar Pessoas", "Rua Doutor Roberto Frias", <School>);
            c2 : Company := new Company("Autoridade Tributaria e Aduaneira",
"Recolher Impostos", "Avenida da Liberdade, nº40, 4400-200", <School>);

            d1 : Date := new Date(2016, 10, 30);
            d2 : Date := new Date(2016, 10, 31);

            eF : Education := new Education(<Graduate>, c1, d2, "Mestrado
Integrado Em Engenharia Informatica e Computacao");
            eI : Education := new Education(<UnderGrad>, c2, d2, "Estagio Final");

            l1 : Language := new Language("Portuguese", 10);
            l2 : Language := new Language("English", 8);
            l3 : Language := new Language("Spanish", 6);
            l4 : Language := new Language("French", 2);

            i1 : Interest := new Interest("Requirements Documents");
            i2 : Interest := new Interest("GTA V");
            i3 : Interest := new Interest("GTA IV");
            i4 : Interest := new Interest("GTA III");
            i5 : Interest := new Interest("GTA II");
            i6 : Interest := new Interest("GTA I");
       operations

            -- START GETS --
            private testGetName: () ==> ()
            testGetName() ==
          (
           assertEqual(pI.getName(), "Jose Miguel Botelho Mendes");
           assertEqual(pF.getName(), "Daniel da Silva Reis")
```

```
  );

private testGetCurrentWork: () ==> ()
     testGetCurrentWork() ==
     (
            assertEqual(pI.getCurrentWork(), wII);
            assertEqual(pF.getCurrentWork(), wFF)
     );

private testGetPastWorks : () ==> ()
     testGetPastWorks() ==
     (
            assertEqual(pI.getPastWorks(), {wI})
     );

private testGetStatus : () ==> ()
     testGetStatus() ==
     (
            assertEqual(pI.getStatus(), <Employed>);
            assertEqual(pF.getStatus(), <Unemployed>)
     );

private testGetPastWorksAfterUnemployed : () ==> ()
     testGetPastWorksAfterUnemployed() ==
     (
            dcl temp : set of Work := {};
            temp := temp union {wF};
            temp := temp union {wFF};
            assertEqual(pF.getPastWorks(), temp)
     );

private testGetSkills : () ==> ()
     testGetSkills() ==
     (
            dcl temp : set of Skill := {};
            temp := temp union {s1};
            temp := temp union {s2};
            assertEqual(pI.getSkills(), temp)
     );

private testGetEducation : () ==> ()
     testGetEducation() ==
     (
            assertEqual(pI.getEducation(), {eI})
     );

private testGetDescription : () ==> ()
     testGetDescription() ==
     (
            assertEqual(pI.getDescription(), "Rei do GTA V");
            assertEqual(pF.getDescription(), "Noob do GTA V")
```

```
        );

    private testGetLanguages : () ==> ()
        testGetLanguages() ==
        (
                dcl temp : set of Language := {};
                temp := temp union {l1};
                temp := temp union {l2};
                assertEqual(pI.getLanguages(), temp)
        );

    private testGetInterests : () ==> ()
        testGetInterests() ==
        (
                dcl temp : set of Interest := {};
                temp := temp union {i1};
                temp := temp union {i2};
                temp := temp union {i3};
                assertEqual(pI.getInterests(), temp)
        );

    private testGetCV : () ==> ()
        testGetCV() ==
        (
                assertEqual(pI.getCV(), "Trolhinha Rustico numero 1");
                assertEqual(pF.getCV(), "Trolhinha Rustico numero 2")
        );

    private testGetConnections : () ==> ()
        testGetConnections() ==
        (
                dcl temp : set of Person := {};
                temp := temp union {p1};
                temp := temp union {p2};
                assertEqual(pI.getConnections(), temp)
        );

    private testGets: () ==> ()
        testGets() ==
        (
                assertEqual(pI.getSkillByName("JAVA"),s1);
                assertEqual(pI.getSkillByName("LOLOL"),nil);
                assertEqual(pI.getLanguageByName("English"),l2);
                assertEqual(pI.getLanguageByName("LOLOL"),nil);
                assertEqual(pI.getInterestByName("GTA IV"), i3);
                assertEqual(pI.getInterestByName("LOLOL"), nil);
                assertEqual(pI.getEducationByName("Autoridade Tributaria e
Aduaneira"),eI);
                assertEqual(pI.getEducationByName("LOLOL"),nil);
                assertEqual(pI.getWorkByName("Autoridade Tributaria e
Aduaneira"),wI);
```

```
				assertEqual(pI.getWorkByName("LOLOL"),nil)
		);
	-- END GETS --

	-- START SETS --
	private testSetStatus : () ==> ()
		testSetStatus() ==
		(
				pI.setStatus(<Employed>);
				pF.setStatus(<Employed>);
				pF.setStatus(<Unemployed>)
		);

	private testSetDescription : () ==> ()
		testSetDescription() ==
		(
				pI.setDescription("Rei do GTA V");
				pF.setDescription("Noob do GTA V")
		);

	private testSetCV : () ==> ()
		testSetCV() ==
		(
				pI.setCV("Trolhinha Rustico numero 1");
				pF.setCV("Trolhinha Rustico numero 2")
		);

	private testSetCurrentWork : () ==> ()
		testSetCurrentWork() ==
		(
				pI.setCurrentWork(wII);
				pF.setCurrentWork(wFF)
		);
		--END SETS--

		--START ADDS--
		private testAddPastWorks : () ==> ()
				testAddPastWorks() ==
				(
						pI.addPastWorks(wI);
						pF.addPastWorks(wF)
				);

		private testAddSkill : () ==> ()
				testAddSkill() ==
				(
						pI.addSkill(s1);
						pI.addSkill(s2);
						pF.addSkill(s3);
						pF.addSkill(s4)
				);
```

```
        private testAddEducation : () ==> ()
                testAddEducation() ==
                (
                        pI.addEducation(eI);
                        pF.addEducation(eF)
                );

        private testAddLanguage : () ==> ()
                testAddLanguage() ==
                (
                        pI.addLanguage(l1);
                        pI.addLanguage(l2);
                        pF.addLanguage(l3);
                        pF.addLanguage(l4)
                );

        private testAddInterest : () ==> ()
                testAddInterest() ==
                (
                        pI.addInterest(i1);
                        pI.addInterest(i2);
                        pI.addInterest(i3);
                        pF.addInterest(i4);
                        pF.addInterest(i5);
                        pF.addInterest(i6)
                );

        private testAddConnection : () ==> ()
                testAddConnection() ==
                (
                        pI.addConnection(p1);
                        pI.addConnection(p2);
                        pI.addConnection(p7);
                        pF.addConnection(p3);
                        pF.addConnection(p4);
                        p1.addConnection(p5);
                        p1.addConnection(p6);
                        p4.addConnection(p6)
                );
        -- END ADDS --

-- START REMOVES --

private testRemovePastWork : () ==> ()
        testRemovePastWork() ==
        (
                pI.removePastWork(wI);
                assertEqual(pI.getPastWorks(), {})
        );
```

```
private testRemoveSkill : () ==> ()
	testRemoveSkill() ==
	(
		pI.removeSkill(s2);
		assertEqual(pI.getSkills(), {s1})
	);

	private testRemoveEducation : () ==> ()
		testRemoveEducation() ==
		(
			pI.removeEducation(eI);
			assertEqual(pI.getEducation(), {})
		);

	private testRemoveLanguage : () ==> ()
		testRemoveLanguage() ==
		(
			pI.removeLanguage(l1);
			assertEqual(pI.getLanguages(), {l2})
		);

	private testRemoveInterest : () ==> ()
		testRemoveInterest() ==
		(
			pI.removeInterest(i1);
			pI.removeInterest(i2);
			assertEqual(pI.getInterests(), {i3})
		);

	private testRemoveConnection : () ==> ()
		testRemoveConnection() ==
		(
			dcl temp : set of Person := {};
			pI.removeConnection(p7);
			temp := temp union {p1};
			temp := temp union {p2};
			assertEqual(pI.getConnections(), temp);
		);
	--END REMOVES

	private testCommonContacts : () ==> ()
		testCommonContacts() ==
		(
			assertEqual(p1.commonContacts(p4), {p6})
		);

	private testDistance : () ==> ()
		testDistance() ==
		(
			assertEqual(p1.distance(p2), -1);
			assertEqual(pI.distance(p6), 2)
```

```
                );

        public static main: () ==> ()
            main() ==
            (
               dcl teste : PersonTest := new PersonTest();
              teste.testGetName();
              teste.testSetDescription();
              teste.testSetCV();
              teste.testSetCurrentWork();
              teste.testAddPastWorks();
              teste.testAddSkill();
              teste.testAddEducation();
              teste.testAddLanguage();
              teste.testAddInterest();
              teste.testAddConnection();
              teste.testGets();
              teste.testGetDescription();
              teste.testGetCV();
              teste.testGetCurrentWork();
              teste.testGetPastWorks();
              teste.testGetSkills();
              teste.testGetEducation();
              teste.testGetLanguages();
              teste.testGetInterests();
              teste.testRemoveConnection();
              teste.testGetConnections();
              teste.testSetStatus();
              teste.testGetStatus();
              teste.testGetPastWorksAfterUnemployed();
              teste.testCommonContacts();
              teste.testRemovePastWork();
              teste.testRemoveSkill();
              teste.testRemoveEducation();
              teste.testRemoveLanguage();
              teste.testRemoveInterest();
              teste.testDistance()
            );
end PersonTest
```

4.8 Class SkillTest

```
class SkillTest is subclass of TestCaseLinkedIn
```

```
        instance variables
                sI : Skill := new Skill("JAVA", 10);
                sF : Skill := new Skill("Alloy", 2);

        operations

                -- START GETS --
                private testGetName: () ==> ()
                testGetName() ==
                (
                assertEqual(sI.getName(), "JAVA");
                assertEqual(sF.getName(), "Alloy");
                );
          private testGetAbility: () ==> ()
                testGetAbility() ==
                (
                assertEqual(sI.getAbility(), 10);
                assertEqual(sF.getAbility(), 2);
                );
          -- END GETS --

          -- START SETS --
          private testSetAbility: () ==> ()
                testSetAbility() ==
                (
                        assertEqual(sI.getAbility(), 10);
                assertEqual(sF.getAbility(), 2);

                sI.setAbility(8);
                sF.setAbility(1);

                assertEqual(sI.getAbility(), 8);
                assertEqual(sF.getAbility(), 1);
                );
          -- END SETS --

        public static main: () ==> ()
                main() ==
                (
                   dcl teste : SkillTest := new SkillTest();
                teste.testGetName();
                teste.testGetAbility();
                teste.testSetAbility();
                );

end SkillTest
```

4.9 Class WorkTest

```
class WorkTest is subclass of TestCaseLinkedIn

	instance variables

		cF : Company := new Company("Faculdade de Engenharia da Universidade 
do Porto", "Formar Pessoas", "Rua Doutor Roberto Frias", <School>);
		cI : Company := new Company("Autoridade Tributaria e Aduaneira", 
"Recolher Impostos", "Avenida da Liberdade, nº40, 4400-200", <Public>);

		dF : Date := new Date(2016, 10, 30);
		dI : Date := new Date(2016, 10, 31);

		wF : Work := new Work("Balconista", dF, cF);
		wI : Work := new Work("CEO", dF, cI);

	operations

	  -- START GETS --
		private testGetTitle: () ==> ()
		testGetTitle() ==
		(
		assertEqual(wI.getTitle(), "CEO");
		assertEqual(wF.getTitle(), "Balconista");
		);
	  private testGetStartDate: () ==> ()
		testGetStartDate() ==
		(
		assertEqual(wI.getStartDate(), dF);
		assertEqual(wF.getStartDate(), dF);
		);
	  private testGetEndDate: () ==> ()
		testGetEndDate() ==
		(
		assertEqual(wI.getEndDate(), dI);
		assertEqual(wF.getEndDate(), dI);
		);
	  private testGetCompany: () ==> ()
		testGetCompany() ==
		(
		assertEqual(wI.getCompany(), cI);
		assertEqual(wF.getCompany(), cF);
		);
	  -- END GETS --

	  -- START SETS --
	  private testSetEndDate: () ==> ()
		testSetEndDate() ==
            (
            wI.setEndDate(dI);
            wF.setEndDate(dI);

            assertEqual(wI.getEndDate(), dI);
            assertEqual(wF.getEndDate(), dI);
            );
       -- END SETS --

     public static main: () ==> ()
            main() ==
            (
               dcl teste : WorkTest := new WorkTest();
      teste.testGetTitle();
      teste.testGetStartDate();
      teste.testGetCompany();
      teste.testSetEndDate();
     teste.testGetEndDate();
          );

end WorkTest
```

4.10 Class CompleteTestLinkedIn

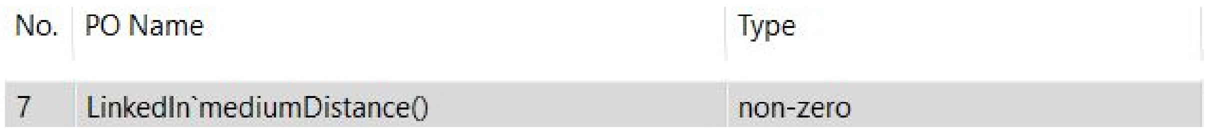
This class is used to properly simulate LinkedIn, creating a full-fledged network and populating it. It executes all other tests.

```
class CompleteTestLinkedIn
     instance variables
            s : SkillTest := new SkillTest();
            d : DateTest := new DateTest();
            l : LanguageTest := new LanguageTest();
            i : InterestTest := new InterestTest();
            c : CompanyTest := new CompanyTest();
            e : EducationTest := new EducationTest();
            w : WorkTest := new WorkTest();
            p : PersonTest := new PersonTest();
            linkedIn : LinkedInTest := new LinkedInTest();

     operations
            public main: () ==> ()
            main() ==
            (
               s.main();
               d.main();
               l.main();
               i.main();
               c.main();
               e.main();
               w.main();
               p.main();
             linkedIn.main()
         );

 end CompleteTestLinkedIn
```

5 Model Verification

5.1 Example of Domain Verification

One of the proof obligations generated by Overture is:



No.	PO Name	Type
7	LinkedIn`mediumDistance()	non-zero

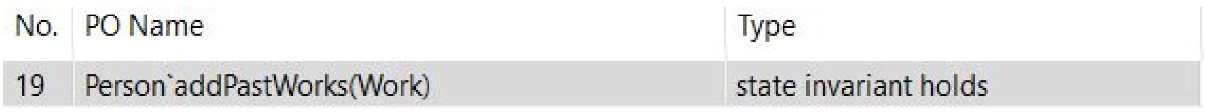
The relevant code under analysis is:

```
public mediumDistance: () ==> int
     mediumDistance() ==
          (
               dcl num : int := 0;
               dcl b : bool := false;
               dcl den : nat := 0;
               for all p in set persons do
                    (
                    for all p1 in set p.getConnections() do
                         (
                              b := true;
                              num := num + p1.distance(p);
                              den := den + 1
                         );
                    );
                    if (not b)
                         then return -1
                    else return num / den;
               );
```

The problem here, as correctly pointed out by overture, is that the expression *(den + 1) <> 0* is not always true (case: *den* = *-1*) and thus would originate a division by 0. However, the domain of den is the set of all natural numbers, meaning it could never be negative, and thus, the expression would always be true. The case where *den* is never incremented and stays with its first assignment of *den := 0* is verified by the boolean *b* and causes a return value of *-1*, so that instance is covered as well. This guarantees that the value of *den* is always in the domain of the positive integers when reaching the division instruction, guaranteeing the integrity of the model.

5.2 Example of Invariant Verification

Another proof obligation generated by Overture is:



No.	PO Name	Type
19	Person`addPastWorks(Work)	state invariant holds

The relevant code under analysis is:

```
public addPastWorks : Work ==> ()
       addPastWorks(w) == (pastWorks := pastWorks union {w}; return)
       pre w not in set pastWorks and w <> currentWork
       post pastWorks = pastWorks~ union {w};
```

The relevant invariant under analysis is:

```
inv {currentWork} inter pastWorks = {}
```

We have to prove that the invariant still holds after the execution of the instruction. Since we have a pre-condition that imposes some restrictions, it means that we have to take into consideration the execution of the instruction only if the pre-condition holds. This means we can write the following:

```
(w not in set pastWorks) and (w <> currentWork) => {currentWork} inter (pastWorks
union {w}) = {}
```

The pre-condition is only ensuring that new works to be added are never the currentWork and that it is never a duplicate in the set. It doesn't, however, ensure that the current work is also in the pastWorks set. This scenario would cause the invariant to break. However, this will never happen, because that is ensured by the pre-condition of the "setCurrentWork()" method, which is the only way to change the currentWork. The code where this is defined is:

```
       public setCurrentWork : Work ==> ()
       setCurrentWork(w) == (currentWork := w; status := <Employed>; return)
       pre status <> <Employed> and w not in set pastWorks;
```

Because the currentWork cannot be added in any other way, the invariant will always hold.

6 Conclusions

The developed LinkedIn model successfully implements all of the requirements and is test-covered in its entirety. As future validation options, we could create Combinatorial Tests to ensure that the model holds in every situation up until a set ammount of interactions.

The project took an aproximated 45 hours to complete.

7 References

- http://mse.isri.cmu.edu/software-engineering/documents/faculty-publications/miranda/kuhnintroductioncombinatorialtesting.pdf - Introduction to Combinatorial Testing - Rick Kuhn National Institute of Standards and Technology Gaithersburg, MD
- Vending Machine Report example
- http://overturetool.org/documentation/ - Official VDM++ overture documentation
- https://docs.google.com/document/d/1cf1cn2qbELCMaHZcBut__0dQL9HGnHlk-l7dmM5q5kw/pub - Overture Quickstart guide