Information Systems Report
A Model of the Olympic Games

Nienke Wessel - s4598350

January 2017

Contents

Chapter 1

Introduction

Modelling (parts of) the real world in such a way that a computer can understand it has been part of research for as long as computers have existed. After all, what use is a computer if you cannot use it in the real world? This research has come a long way and has produced many subfields of modelling and many theories for those subfields. In this report, I will take a look at a specific theory in the information systems modelling.

This closer look will consist of five parts, described in their respective chapters. The main goal is to test the theory against an existing real-life information system and determine what the stronger and weaker parts of the theory are. Our real-world information system will be the Olympic Games as described on `www.olympic.org/sports`. First, I will give the model I made of the Olympic Games. I will describe why I modelled the games this way and will give a small population of the model. After that, I will take a closer look at an application of information modelling, the context-free grammar. After that, I will describe what went well and what difficulties I ran into while modelling the Olympic Games with the theory. In the last two chapters, I will propose a change for the theory to resolve a difficulty and provide an set-up for an implementation for automatizing model-checking.

Chapter 2

The Model

2.1 Introduction

In this section, I will show the models I made. The main model is divided into smaller models to improve readability and clearity. First, I will show the models by using ORM to improve visualisation. Then I will use the formalizatoion with mathematical notations to move away from the constraints of ORM and show in general how this model works. Last, I will explain some choices I made in my model that may seem odd.

2.2 The Models

2.2.1 The Olympic Games

The schema

The schema is shown in 2.1

This is the main model of the Olympic Games. A smaller model of a country is given in the next section.

Formalization

- $\mathcal{P}$ = {hasAt, isFrom, isRepBy, repre, isLoc, heldIn, hasOl, isIn, hasEvThen, isDay, isEvNaOf, hasEvName, hasSe, isSeOf, isOfSp, hasEv, isEvNaOf, hasEvName, isTy, hasTy, hasSNa, isSNaOf, coOf, hasCo, hasNa, isNaOf, bornOn, birthdayOf, inEvSc, athSc, isScbyAtIn, $\in^{\text{e}}_{\text{Team}}, \in^{\text{p}}_{\text{Team}}$}
- $\mathcal{O} = \mathcal{F} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \cup \mathcal{E} \cup \mathcal{L}$
- $\mathcal{F}$ = {citizenship, nationality, location, year, EventOccasion, dayNr, SportEvent, dayNr, typeEv, spoNamed, coach, named, born, evPlacement, $\in_{Team}$}
- $\mathcal{G}$ = {Team}
- $\mathcal{S} = \mathcal{C} = \{\}$
- $\mathcal{E}$ = {Country, Olympic Games, Event, Sport, Type, Placement, Athlete, Coach, Person}
- $\mathcal{L}$ = {Year, Day, EName, SName, Name, Date}

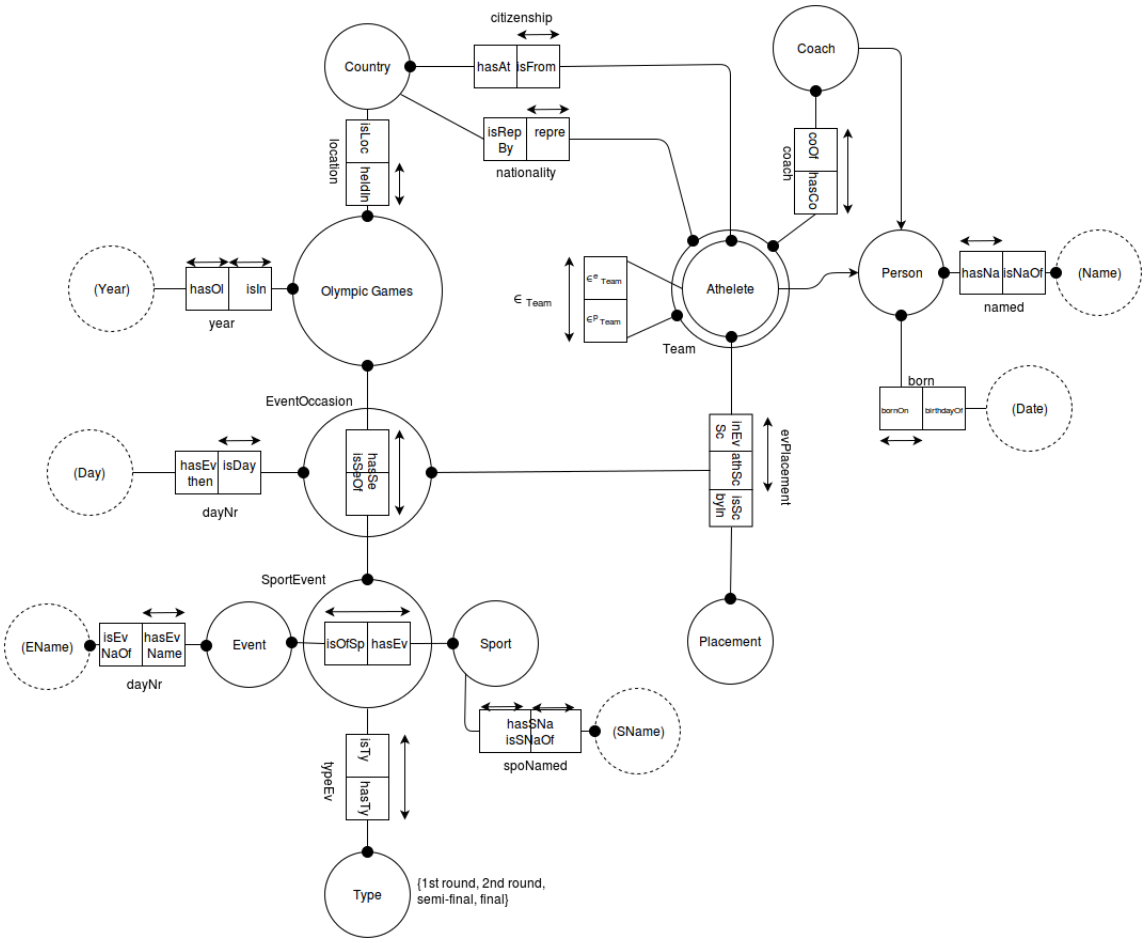


Figure 2.1: Model of the Olympic Games

where location = { isLoc, heldIn }, evPlacement = { inEvSc, athSc, isScbyAth}, etc. Futhermore, **Base**(isLoc) = Country, **Base**(heldIn) = OlympicGames, etc. **Elt**(Team) = Athlete, Coach **Spec** Person and Athlete **Spec** Person. **Gen** is not used in this model. The domains of Year and Day are the natural numbers. The domain of Date is the set of dates as we use them in The Netherlands. The domain of all other labels is the set of strings that can be formed from the Latin alphabet, plus numerical characters.

The following constraints were used:

- **unique**(isFrom), **unique**(heldIn), etc.
- **total**(hasAt), **total**(heldIn), etc.
- **enumeration**(Type, {1st round, 2nd round, semi-final, final})

Explanation of the Model

The model is centralized about the big entity of an Olympic Games. An Olympic Games can be identified by the year it took place in. Sports are modelled in a rather complicated way, so lets

take a closer look at them. A sport (with a SName) has specific events. For example, Hockey has the "12 team tournament women". These events are of specific types. You can have a hockey finale or a hockey semi-finale of this type. Each Olympic Games has an occasion of such an event on a specific day. This is the actual Object the athletes take place in, i.e. a specific event of a specific sport in a specific Olympic Games. An athlete places some number in that event.

In the rest of the model, you can see that athletes can form a team and that a team has a coach. Both athletes and coaches are people that have names and birthdates.

Population

We will provide a small population for the model. We will not provide a population for the complete model, simply because it is not useful.
Pop(Country) = { The Netherlands }
Pop(Olympic Games) = { GamesOf2016 }
Pop(Year) = { 2016 }
Pop(year) = { {hasOl: 2016, isIn: GamesOf2016 } }
Pop(Athlete) = { NaomiAthlete }
Pop(Team) = { { NaomiAthlete } }
Pop(Coach) = { AlysonAnnanCoach }
Pop(Person) = { AlysonAnnanCoach, NaomiAthlete }
Pop(Name) = { Alyson Annan, Naomi van As }
Pop(named) = { {hasNa: NaomiAthlete, isNaOf: Naomi van As }, {hasNa: AlysonAnnanCoach, isNaOf: Alyson Annan } }
Pop(Sport) = { HockeySport }
Pop(Event) = { 12TeamTournamentWomenEv }
Pop(SportEvent) = { { isOfSp: 12TeamTournamentWomenEv, hasEv: HockeySport } }
Pop(EventOccasion) = { { isSeOf: { isOfSp: 12TeamTournamentWomenEv, hasEv: HockeySport }, hasSe: GamesOf2016 } }
Pop(Placement) = { 2 }
Pop(evPlacement) = { { inEvSc: NaomiAthlete, athSc: { isSeOf: { isOfSp: 12TeamTournamentWomenEv, hasEv: HockeySport }, hasSe: GamesOf2016 }, isScbyAtIn: 2 } }
etc.

This population does not violate any constraints, except for a few total role constraints (such as **total**(bornOn)), as I am not providing a full population.

2.2.2 The Countries

The schema

The schema is shown in 2.2

Formalization

- $\mathcal{P} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24\}$
- $\mathcal{O}$ = {pre, seg, add, rcd, tle, web, fax, ema, pho, presNa, secNa, couNa, PhoneNumber, President, Name,SecretaryGeneral, Address, RecognitionDate, Date, Title, Website, FaxNumber, Email, Country, CName}
- $\mathcal{F}$ = {pre, seg, add, rcd, tle, web, fax, ema, pho, presNa, secNa, couNa}

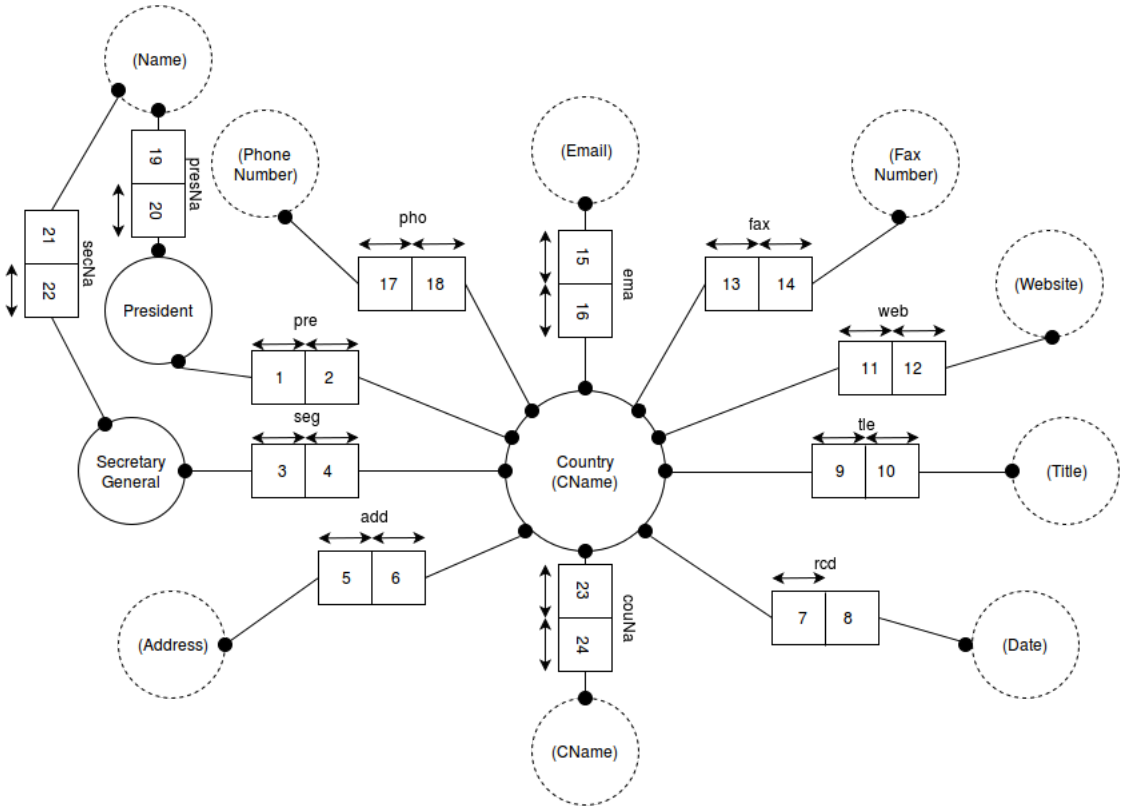


Figure 2.2: Model of a country in the Olympic Games

- $\mathcal{G} = \mathcal{S} = \mathcal{C} = \{\}$
- $\mathcal{E} = \{$President, SecretaryGeneral, Country$\}$
- $\mathcal{L} = \{$PhoneNumber, Email, FaxNumber, Website, Title, RecognitionDate, Address, Name, CName$\}$

where pre $= \{1, 2\}$, seg $= \{3, 4\}$, add $= \{5, 6\}$, rcd $= \{7, 8\}$, tle $= \{9, 10\}$, web $= \{11, 12\}$, fax $= \{13, 14\}$, ema $= \{15, 16\}$, pho $= \{17, 18\}$, presNa $= \{19, 20\}$, secNa $= \{21, 22\}$ and couNa $= \{23, 24\}$
Furthermore, **Base**(1) = President, **Base**(2) = Country, etc. **Elt**, **Gen**, **Spec** are not used in this model. The domain of all labels is the set of strings that can be formed from the Latin alphabet, plus numerical characters and characters like "@".

The following constraints were used:

- **unique**(1), **unique**(2), **unique**(3), **unique**(4), **unique**(5), **unique**(6), **unique**(7), **unique**(9), **unique**(10), **unique**(11), **unique**(12), **unique**(13), **unique**(14), **unique**(15), **unique**(16), **unique**(17), **unique**(18), **unique**(20), **unique**(22), **unique**(23), **unique**(24)
- **total**(1), **total**(2), **total**(3), **total**(4), **total**(5), **total**(6), **total**(7), **total**(8), **total**(9), **total**(10), **total**(11), **total**(12), **total**(13), **total**(14), **total**(15), **total**(16), **total**(17), **total**(18), **total**(19), **total**(20), **total**(21), **total**(22), **total**(23), **total**(24)

Explanation of the Model

The model is centered around a country, and a country has a lot of labels, most of it what one would call contact information. Most labels are pretty self-explanatory, maybe except for rcd which means recognition date. The population as supplied in the previous part is from the page `https://www.olympic.org/cyprus`

Population

I've chosen to use only one country to show the population, as there is not much use in showing more.
Pop(Country) = {CyprusCountry}
Pop(President) = {CyprusPres}
Pop(SecretaryGeneral) = {CyprusSec}
Pop(Name) = {General Charalambos Lottas, Mr Dinos Michaelides}
Pop(CName) = {Cyprus}
Pop(Address) = {Olympic House 21 Amfipoleos St. P.O. Box 23931 1687 Nicosia Cyprus}
Pop(Date) = {01/01/1978}
Pop(Title) = {The Cyprus National Olympic Committee}
Pop(Website) = {http://www.olympic.org.cy}
Pop(FaxNumber) = {+357 22 449 890}
Pop(Email) = {cypnoc@cytanet.com.cy}
Pop(PhoneNumber) = {+357 22 449 880}
Pop(pre) = {{1: CyprusPres, 2: CyprusCountry}}
Pop(couNa) = {{23: CyprusCountry, 24: Cyprus}}
Pop(tle) = {{9: Cyprus, 10: The Cyprus National Olympic Committee}}
etc.

2.3 Special Choices

2.3.1 Identify Olympic Games by Year

I chose to identify a specific Olympic Game by the year. This may seem odd, as the year an Olympic game took place can also be modelled by another object with the text " ... took place in ... ". However, this leaves the problem of what then to identify a specific Olympic Games with. The country it took place in is not unique. The sets of sports or athletes that were in that games probably is unique, but not very convenient or intuitive. The year is unique (there is only one or zero Olympic Games in each year) and it is actually intuitive and commonly used when people talk about the Olympic Games (people also tend to use the place where it took place, but like said, this is not unique). A combination of place and year might be more intuitive, but produces redundancy.

2.3.2 Address as one object

When taking the course Modelleren, we were learned different ways to display an address. However, in my model, I have chosen to use none of those and simply went with a general label "Address". The choice for this is based on the fact that the website of the Olympic Games also lists this as a single thing. Also, an address has already proven to be modelled with the theory, so there is not much point in doing so again.

2.3.3 Leaving out the Torch

On the Olymic Games website, there was also information about the torch for each Olympic Games. However, I have chosen not to include this information in the my model. This is because including the information would give a model pretty similar to the model of the countries, i.e. one main object with a lot of labels. Since the structure of this would be mainly the same as the structure of another model I already made, I did not think it would be of much help to make this second model in testing the theory. Therefore, this model is excluded from this report.

Chapter 3

A Context-Free Grammar

3.1 Introduction

In this section we will take a closer look at one of the applications of the theory discussed in the syllabus, specifically the application mentioned in chapter 5, the context-free grammar. In this section, we are going to use a context-free grammar to form a string of an event-name. This is of course a very simple context-free grammar, but shows the possibility of combining information modelling and context-free grammars. We will start by giving the grammar and the model and then briefly discuss it.

3.2 The Grammar

I developed the following grammar that produces strings like "Swimming 100m backstroke women"

$$\langle SportEvent\rangle \rightarrow \langle Sport\rangle \text{ " " } \langle Distance\rangle \text{ "m " } \langle Event\rangle \text{ " " } \langle Gender\rangle$$
$$\|\langle Sport\rangle \text{ " " } \langle Event\rangle \text{ " " } \langle Gender\rangle$$
$$\langle Sport\rangle \rightarrow \langle String\rangle$$
$$\langle Distance\rangle \rightarrow \langle Integer\rangle$$
$$\langle Event\rangle \rightarrow \langle String\rangle$$
$$\langle Gender\rangle \rightarrow \text{"women"}\|\text{"men"}$$
$$\langle String\rangle \rightarrow \langle Char\rangle^{+}$$

Of course, this grammar is overly simple and too simple to accurately describe the Olympic Games, but it gives a general view of how a context-free grammar can be used. There is a distinction between sports that do use a distance to describe their different elements and between sports that do not. There is also a distinction between the female and the male event of the sport.

3.2.1 Generate a sentence

Let's generate the example sentence "Swimming 100m backstroke women". The first time $\langle String \rangle \rightarrow \langle Char \rangle^+$ is shown in its completeness. After that, this step is not shown anymore.

$$\langle SportEvent \rangle \rightarrow \langle Sport \rangle \text{ " " } \langle Distance \rangle \text{ "m " } \langle Event \rangle \text{ " " } \langle Gender \rangle$$
$$\rightarrow \langle String \rangle \text{ " " } \langle Distance \rangle \text{ "m " } \langle Event \rangle \text{ " " } \langle Gender \rangle$$
$$\rightarrow \langle Char \rangle\langle Char \rangle\langle Char \rangle\langle Char \rangle\langle Char \rangle\langle Char \rangle\langle Char \rangle\langle Char \rangle \text{ " " } \langle Distance \rangle \text{ "m " }$$
$$\langle Event \rangle \text{ " " } \langle Gender \rangle$$
$$\rightarrow \text{"Swimming "} \langle Distance \rangle \text{ "m " } \langle Event \rangle \text{ " " } \langle Gender \rangle$$
$$\rightarrow \text{"Swimming 100m " } \langle Event \rangle \text{ " " } \langle Gender \rangle$$
$$\rightarrow \text{"Swimming 100m backstroke " } \langle Gender \rangle$$
$$\rightarrow \text{"Swimming 100m backstroke women"}$$

3.3 The Model

See 3.1

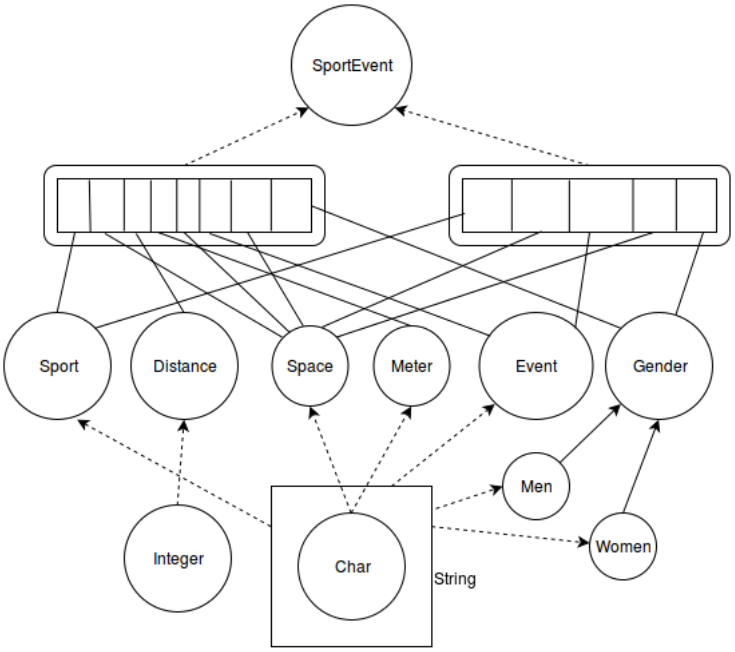


Figure 3.1: Model of the context-free grammar to describe an event at the Olympic Games

Chapter 4

Suitability of the Theory

4.1 Introduction

In this section, I will discuss the suitability of the theory for modelling the Olympic Games. In order to do this, I will first describe the strong parts of theory that go well with modelling this specific Universe of Discourse. Then I will describe the parts where the theory is less suited. Last, I will briefly summarize the findings.

4.2 Conveniences

I will briefly describe several occasions on which the theory proved useful. I will also try to generalize these occasions to more general occasions where the theory most likely will be useful.

4.2.1 Objects with a lot of roles

It is easy to model objects with a lot of roles. In the model described above, this is the case with the Country and its many roles. These can all be represented in a organized and clear manner. This makes the theory well-suited for databases with many simple relations.

4.3 Problems

I will briefly describe problems I ran into while applying the theory to the Olympic Games. Doing this, I will describe the several attempts I did to resolve the problem and why they failed to do so. In the next chapter, I will propose changes in the theory that should solve most of the problems mentioned here.

4.3.1 Modelling Types of Events

When modelling the different type of events, I chose to make a objectification of the sport and a specific event. For example, Athletics combined with 100m Male makes a `SportEvent`. This `SportEvent` can occur multiple times on different dates in the same Olympic Games, as there are differrent rounds. For example, there could be a finale and a semi-finale, but no first and second round of a particular sport. However, there is no simple way of defining those kinds of constraints with the current theory.

4.3.2 Constraints on Power Types

While working on the model, I wanted to include a certain constraint, but I could not find a suitable constraint in the current theory. The problem: a group of athletes can form a team, but only if they are of the same nationality. Also, the nationality of that team should be the same nationality as the athletes have. I could not find constraints that would make this work. Uniqueness constraints do not have anything to do with this problem, so they are not useful to solve it. Enumeration constraints, total role constraints and occurence frequency constraints are also not relevant here. The interesting constraints are set constraints and power type constraints. The problem of set constraints is that they require the two sets to be compared of the same type and the sets we want to compare are not. We want to compare athletes with **sets** of athletes. These two types are not comparable. For power type constraints, these are only usable on a power type and its **Elt**(). This is not the case here.

The theory states that "it is not possible, nor even desirable, to specify all static constraints graphically." This is probably true, but it does limit us in modelling the real world.

4.3.3 Complex Context-Free Grammars

When modelling a context-free grammar, the model can become complex very quickly. The model of 3.1 is already complex with all the lines crossing each other and it is hard to maintain view of the structure of the model, even though it is a relatively simple model. Modeling the real world with a context-free grammar will prove challenging.

Chapter 5

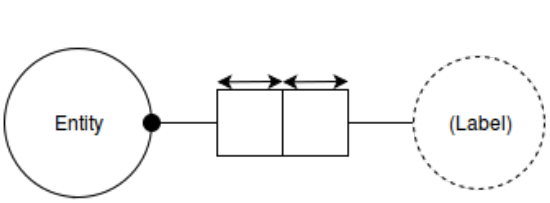
Suggestion for Improvement

5.1 Introduction

In this section, I will propose an improvements for the theory. I will first generally explain in what situations this is interesting and then discuss my proposal in greater depth.

5.2 Labels and Entities

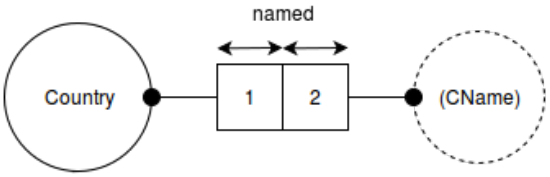
In the theory, there is a strict distinction between labels and entities. And while I do recognize the importance of the distinction between the two, it can be meaningful to merge the two into one new type. For example, consider figure 2.2 from chapter 2, in which countries are modelled. In this figure, a country has a president. A president is an entity. This president has the label Name, by which a president can be identified. One could argue that you could make President the label, but this clearly violates the distinction between labels and entities. But yet it is counter-intuitive to have an 'extra' entity type. Because, when populating the President entity, with what do you populate it? This is especially relevant when making relational databases. When one populates a relational database for the schema as given in 2.2, the entity of President and the label Name will be merged into one entry.



(a) Entity and label according to theory

(b) Entity and label as proposed

Figure 5.1: Entities and labels graphically depicted



(a) Entity and label according to theory

(b) Entity and label as proposed

Figure 5.2: Example of merged entity and label objects

5.2.1 The Proposal

The proposal is to make a new type of object that merges the entity and its label. Graphically, this is depicted in 5.1. In 5.1a one can see the graphical notation as in the current theory. In 5.1b, one can see the new graphical notation.

As can be seen in the images, this merge requires several constraints. First of all, the merge is not possible if not all entities have a label. Furthermore, it makes the most sense to perform such a merge when the label is unique for the entity and the entity unique for the label. That means the label can be used as a key for the entity, in terms of relational databases. Further research could be done into finding whether one of the two uniqueness constraints is necessary.

Formalization

We introduce a new object type $\mathcal{E}_\mathcal{L} \subseteq \mathcal{O}$, which consists of a set of *entity-label types*. These are entity types with an label. For this new type, the same laws hold, i.e.

$$|\mathcal{L} \cup \mathcal{E} \cup \mathcal{F} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \cup \mathcal{E}_\mathcal{L}| = |\mathcal{L}| + |\mathcal{E}| + |\mathcal{F}| + |\mathcal{G}| + |\mathcal{S}| + |\mathcal{C}| + |\mathcal{E}_\mathcal{L}|$$

and

$$\mathcal{L} \cup \mathcal{E} \cup \mathcal{F} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \cup \mathcal{E}_\mathcal{L} = \mathcal{O}$$

Example

As an example, we will take a look at a country and its name, CName in the model in chapter 2. We will merge these two objects into one object of the new type. Both before and after the merge are shown in 5.2. We will also take a look at the formalization in this example.

Before merge:

- $\mathcal{P} = \{1, 2\}$
- $\mathcal{O} = \{$named, Country, CName$\}$
- $\mathcal{F} = \{$named$\}$
- $\mathcal{G} = \mathcal{S} = \mathcal{C} = \{\}$
- $\mathcal{E} = \{$Country$\}$
- $\mathcal{L} = \{$CName$\}$
- named $= \{1, 2\}$

- **Base**(1) = Country, **Base**(2) = CName
- **unique**(1), **unique**(2)
- **total**(1), **total**(2)

After merge:

- $\mathcal{P} = \mathcal{O} = \mathcal{F} = \mathcal{G} = \mathcal{S} = \mathcal{C} = \mathcal{E} = \mathcal{L} = \{\}$
- $\mathcal{E}_{\mathcal{L}} = \{\text{Country}\}$

We will also provide a sample population of the given example.
Before merge:

- **Pop**(Country) = {CyprusCountry, NetherlandsCountry}
- **Pop**(CName) = {Cyprus, The Netherlands}
- **Pop**(named) = { {1: CyprusCountry, 2: Cyprus}, {1: NetherlandsCountry, 2: The Netherlands} }

After merge:

- **Pop**(Country) = { Cyprus, The Netherlands }

This example population shows that the idea is to use the Entity-name to indentiy the new structure, but the population of the labels as the new population of the whole structure.

Chapter 6

Attempt for Automatization

6.1 Introduction

Automatizing processes is becoming more and more important as the size of a project grows. This also holds for information systems and their models. There is a lot of interest in developing ways to automatize the proces of model checking and population. In this chapter, I provide a small set-up for automatizing the checking of models. This is done in the object orientated programming language Java, but it could be done in any language. The choice for object orientation is because the structure of the theory lends itself for object orientation.

6.2 The Code

Object Interface

We start with an interface for an object $\mathcal{O}$.

```
import java.util.ArrayList;

public interface Object {
    public String getName();
    public ArrayList<String> Pop();
    public void setPred(Predicate pred);
    public ArrayList<Predicate> getPred();
    public Type getType();
}
```

I only implemented classes for Entity, Fact, Label and Predicate. Those are the following:

```
import java.util.ArrayList;

public class Entity implements Object {
    private String name;
    private ArrayList<String> population;
    private ArrayList<Predicate> preds;

    public Entity(String name, ArrayList<String> pop){
```

```
        this.name = name;
        population = pop;
        population = new ArrayList<String>();
        preds = new ArrayList<Predicate>();
    }

    @Override
    public String getName(){
        return name;
    }

    @Override
    public ArrayList<String> Pop() {
        return population;
    }

    @Override
    public void setPred(Predicate pred) {
        preds.add(pred);
    }

    @Override
    public ArrayList<Predicate> getPred() {
        return preds;
    }

    @Override
    public Type getType() {
        return Type.Ent;
    }

}

public class Label implements Object {
    private String name;
    private ArrayList<String> population;
    private ArrayList<Predicate> preds;

    public Label(String name, ArrayList<String> pop){
        this.name = name;
        population = pop;
        population = new ArrayList<String>();
        preds = new ArrayList<Predicate>();
    }

    public void setPred(Predicate pred) {
        this.preds.add(pred);
    }
```

```
    @Override
    public String getName(){
        return name;
    }

    @Override
    public ArrayList<String> Pop() {
        return population;
    }

    public ArrayList<Predicate> BaseInv() {
        return this.preds;
    }

    @Override
    public ArrayList<Predicate> getPred() {
        return preds;
    }

    @Override
    public Type getType() {
        return Type.Lab;
    }

}

public class Fact implements Object {
    private String name;
    private ArrayList<Predicate> preds;

    public Fact (String name ){
        this.name = name;
        preds = new ArrayList<Predicate>();
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public ArrayList<String> Pop() {
        return null;
    }

    @Override
    public void setPred(Predicate pred) {
        preds.add(pred);
    }
```

```
    public ArrayList<Predicate> getPreds(){
        return this.preds;
    }

    @Override
    public ArrayList<Predicate> getPred() {
        return preds;
    }

    @Override
    public Type getType() {
        return Type.Fac;
    }

}

public class Predicate implements Object {
    private String name;
    private Object object;
    private Fact fact;

public Predicate(String name, Object object, Fact fact){
        this.name = name;
        this.object = object;
        this.fact = fact;
        object.setPred(this);
        fact.setPred(this);
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public ArrayList<String> Pop() {
        return null;
    }

    @Override
    public void setPred(Predicate pred) {
    }

    public Object Base() {
        return object;
    }

    @Override
```

```
    public ArrayList<Predicate> getPred() {
        return null;
    }

    @Override
    public Type getType() {
        return Type.Pred;
    }

}
```

I have not yet been able to correctly form a list of strings for the output of Pop for predicates. However, this should be easy when one puts more time in actually completing this automatization.

I added an enumeration type to keep track of of what type each Object was.

```
public enum Type {
    Ent, Lab, Pred, Fac;
}
```

Population

I tried the following sample:

```
ArrayList<Object> list = new ArrayList<Object>();
ArrayList<String> names = new ArrayList<String>();
ArrayList<String> names2 = new ArrayList<String>();

names.add("NaomiAthlete");
names2.add("Naomi van As");

Entity athlete = new Entity("Athlete", names);
Label name = new Label("Name", names2);
Fact named = new Fact ("named");
Predicate hasNa = new Predicate ("hasNa", athlete, named);
Predicate isNaOf = new Predicate ("isNaOf", name, named);

list.add(name);
list.add(athlete);
list.add(named);
list.add(hasNa);
list.add(isNaOf);
```

Testing

There are many different things that can be tested with such a model. As a proof of concept, I only give a test that checks whether every Fact type has one or more predicates.

```
public boolean FactHasPred(ArrayList<Object> objects) {
            for (Object obj : objects) {
                if (obj.getType() == Type.Fac) {
                    if (obj.getPred().size()<=0)
                        return false;
            }
        }
    return true;
}
```

Further Work

It could be interesting to complete this model, i.e. include all object types. Also, more tests should be defined other than just this simple, almost trivial, test.

Chapter 7

Conclusion

In general, the theory is pretty complete and solid, as can be expected of a field that has been around this long. But as always, there are some aspects that can be improved and there are always new fields to apply the theory on. This report only captures a tiny, tiny bit of both of those things, as the theory is way to big to consider all of it in a small report.

However, we tried to take a look at the field of information systems and its applications and found some interesting things. For example, as usual there are certain parts of the theory that do not apply well in every situation. Defining all possible natural world constraints for a model is known to be very hard and time-consuming, if not impossible, which is why we found some missing constraints. There are so many domains to which we could apply the theory of information modelling, but we tried to briefly touch a few of them.

7.1 Suggestions for Further Research

Like said, there are so many more fields to consider, but a few that I personally would find interesting are mentioned in this list.

- Modelling natural languages. Try to capture the structure of a certain natural languages in a information system structure.
- Develop a program that can draw a schema based on a formalization.
- Further work on developing programs that can automatize model checking and increase efficiency of those programs.