CAS 703 Project Report

Dong Chen

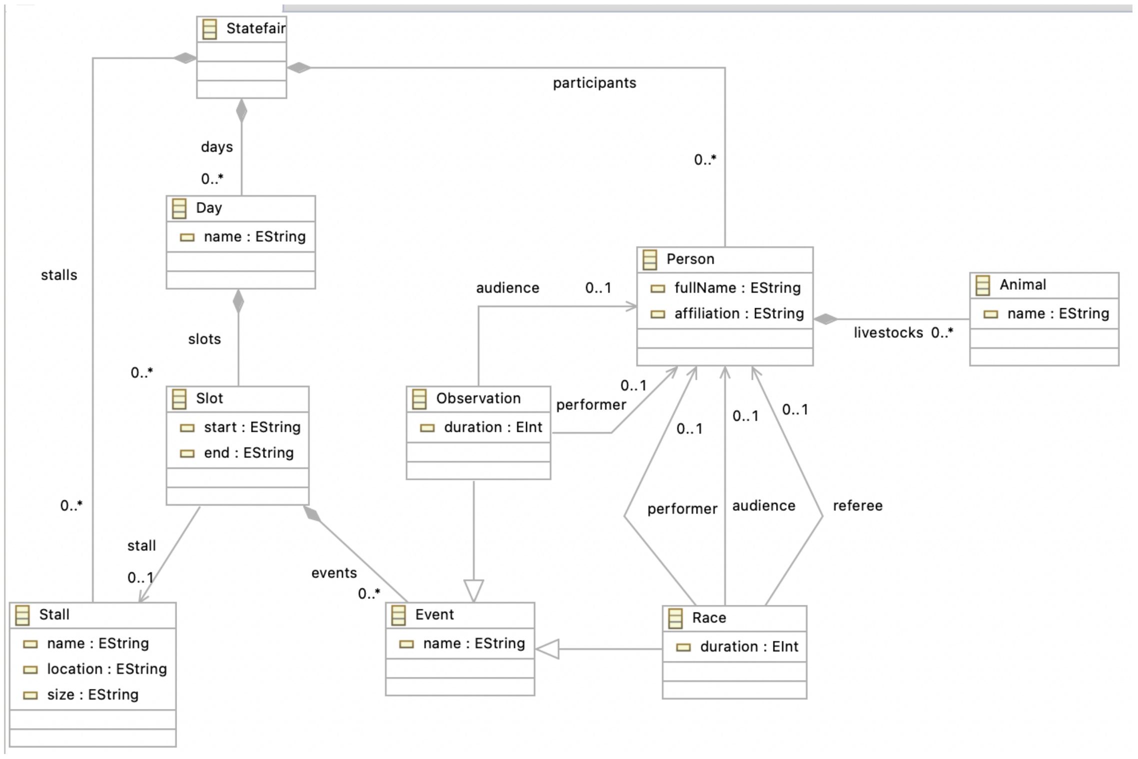
April 14, 2021

This is the report for CAS 703 Software Design. The report will cover the implementation of Emfatic, Ecore, Eugenia, EVL, EOL, and model transformation.

1 Introduction

Typically, the state fair will be hosted in some states in the United States. In some states, like Iowa, Iowans host the event each year during summer. The state fair event usually will last for one day. During the day, participants can attend numerous events, including live shows and livestock competitions. Depend on how much livestock needs to manage, livestock needs a stall to rest. An event may also need some referees, performers, and audiences.

2 Ecore Class Diagram



2.1 Define and Analyze Metamodel

The following classes will be defined in Emfatic and Ecore files.

- Statefair: contains multiple Person, Stall, and Day.
- Person: has a name, company, and multiple livestock.
- Day: has a name and multiple slots.
- Slot: has a start and end time, it also contains multiple events and a reference to Stall
- Event: abstract class contains a name.
- Observation: has a duration, and two references to Person.
- Race: has a duration, and three references to Person.

- Stall: has a name, location, and size.
- Animal: has a name.

2.2 Assumptions

None

2.3 Alternative Design

An alternative way is to let the Animal class be a component of the StateFair class rather than a component of the Person class. If this is the case, in the Animal class, there would be a reference to determine who owns the livestock. The Observation class and Race class need to have a dependency/reference on the Animal class because they might require livestock in the event.

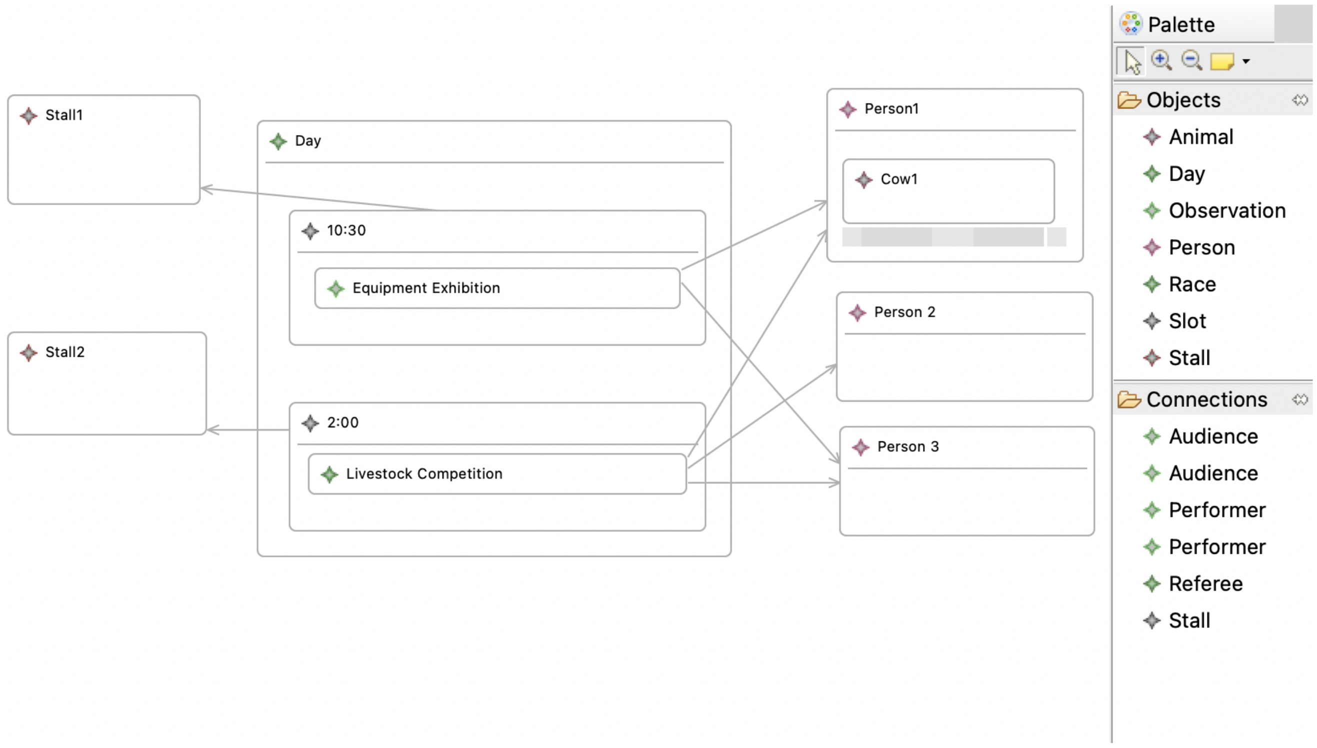
3 Concrete Syntax and Editors

In this section, a tool called Eugenia will be implemented to create a graph editor.

3.1 GMF and Define in Eugenia

Eugenia helps to define the graphical modeling framework graph relationship and a new graph editor can be constructed.

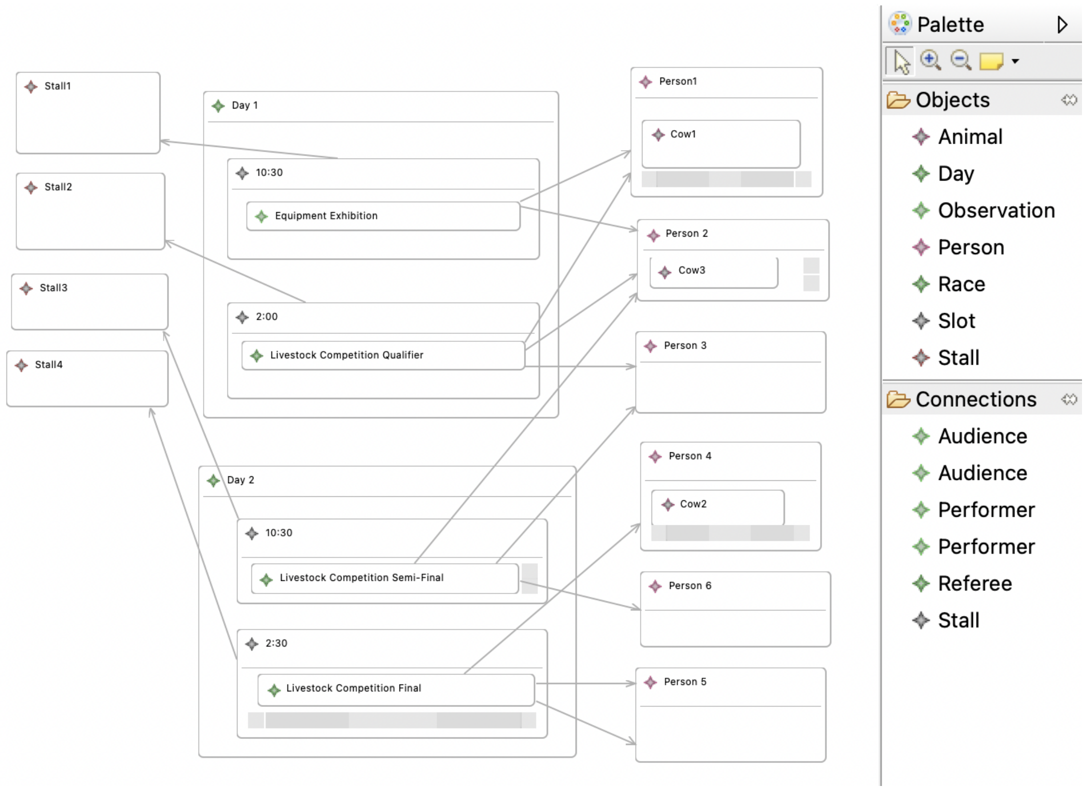
3.2 The Graph Editor



The details of how to define the graph editor in the following list.

- @gmf.diagram: Statefair - (this is the start of diagram)
- @gmf.node: Person, Day, Slot, Event(Observation, Race), Stall, Animal - (they are nodes represent each class)
- @gmf.link: Stall, Person - (they are dependencies/references)
- @gmf.compartment: Animal[*], Slot[*], Event[*] - (compartment allow a shape inside a shape)

3.3 More Cases in the Editor



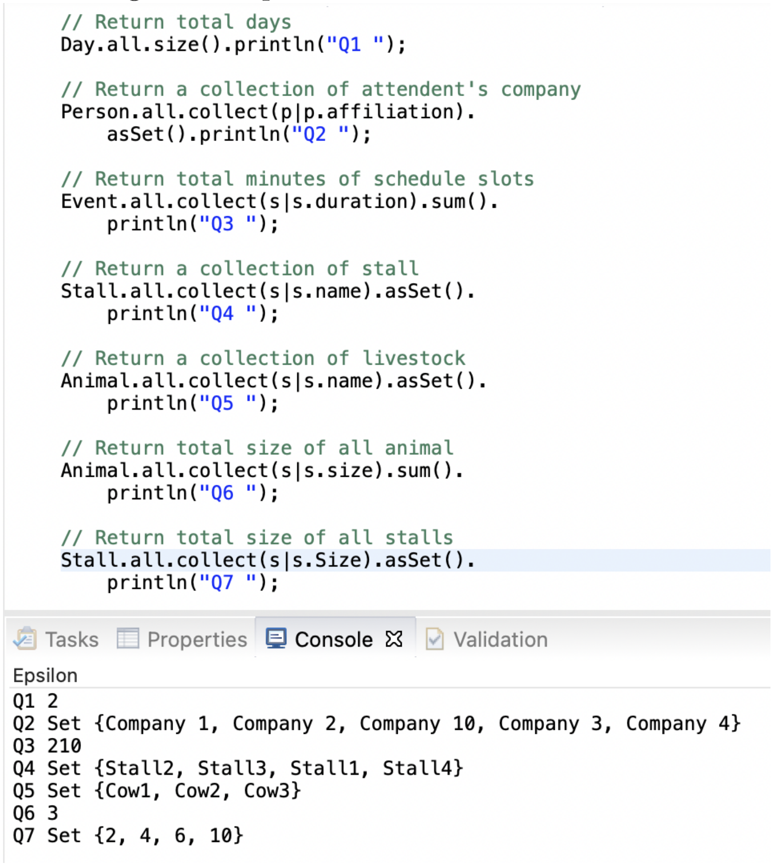
3.4 Strengths and Weaknesses

Strengths: Eugenia helps to visualize the relationship between each class. It is very intuitive and easy to understand.
Weaknesses: Eventually, it can get messy when more cases add to the editor. Increased cases can lead the editor hard to read. Each time, it requires to launch of a new instance of the project. This is an extra step that some developers may think it is tedious. The work between the default eclipse and the new editor is not synchronized, which means data added in the new graph editor are not available in the previous tree editor.
Additionally, developers might feel it is inefficient to use the graph editor. It requires moving shapes and arrows around, and some people may get very picky on the position of each shape. A text editor can avoid this situation.

4 Validation Constraints

4.1 Queries EOL

The following are eol queries in the StateFair.



```
    // Return total days
    Day.all.size().println("Q1 ");

    // Return a collection of attendent's company
    Person.all.collect(p|p.affiliation).
        asSet().println("Q2 ");

    // Return total minutes of schedule slots
    Event.all.collect(s|s.duration).sum().
        println("Q3 ");

    // Return a collection of stall
    Stall.all.collect(s|s.name).asSet().
        println("Q4 ");

    // Return a collection of livestock
    Animal.all.collect(s|s.name).asSet().
        println("Q5 ");

    // Return total size of all animal
    Animal.all.collect(s|s.size).sum().
        println("Q6 ");

    // Return total size of all stalls
    Stall.all.collect(s|s.Size).asSet().
        println("Q7 ");
```

```
Epsilon
Q1 2
Q2 Set {Company 1, Company 2, Company 10, Company 3, Company 4}
Q3 210
Q4 Set {Stall2, Stall3, Stall1, Stall4}
Q5 Set {Cow1, Cow2, Cow3}
Q6 3
Q7 Set {2, 4, 6, 10}
```

4.2 Constraints Analysis

```
context Race{
    constraint C1{
        check: self.performer <> self.referee
        message: "The performer and the referee of race"
            + self.`name` + " are the same person"
    }

    constraint C2{
        check: self.performer <> self.audience
        message: "The performer and the audience of race"
            + self.`name` + " are the same person"
    }

    constraint C3{
        check: self.referee  <> self.audience
        message: "The referee, and the audience of race"
            + self.`name` + " are the same person"
    }

    constraint C4{
        check: self.duration > 0
        message: "The duration of race " + self.`name` +
            "is not a positive number"
    }

    constraint C5{
        check: self.performer.livestocks.size > 0
        message: "The number of livestock of race " + self.`name` +
            "is not a positive number"
    }

}

context Observation{
    constraint C6{
        check: self.performer <> self.audience
        message: "The performer and the audience of race"
            + self.`name` + " are the same person"
    }

    constraint C7{
        check: self.performer.livestocks.size > 0
        message: "The number of livestock of race " + self.`name` +
            "is not a positive number"
    }
}
```

Firstly, there are performers, audience, and referees in the design. In most cases, these three individuals can not be the same person. Secondly, the duration of each event should great than 0 minutes. Thirdly, the size of livestock should great than 0 in each event. Besides implemented constraints, there are some useful constraints that can be added in the future. For example, a certain type of livestock can not stay on a stall that is near other types of livestock. A start time should earlier than the end time. Details will be showed in the following sections.

- C1, C2, and C3: In the race, these three constraints make sure the audience, performer, and referee can not be the same person.

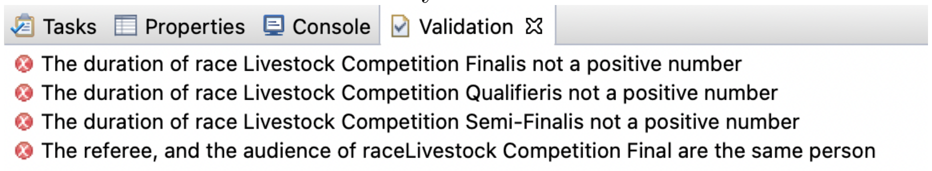
- C4: the duration of each event need to great than 0.
- C5: the size of livestock which attend the race need to great than 0.
- C6: In the observations, the performer and audience can not be the same person.
- C7: the size of livestock which attend the observation need to great than 0.

4.3 Constraints do not know How to Implement

- Time comparison: the start time need to earlier than the end time in an event.
- The size of the stall should great than the total size of livestock attend in the event. This means every livestock will have a stall to rest during the event.

4.4 Run EVL Individually

In the new graph editor, add an evl file. After that right-click it and set the run configure. Click EMF Model, then choose the diagram file as the model file. Then, the author can run the evl constraints individually.



```
The duration of race Livestock Competition Finalis not a positive number
The duration of race Livestock Competition Qualifieris not a positive number
The duration of race Livestock Competition Semi-Finalis not a positive number
The referee, and the audience of raceLivestock Competition Final are the same person
```

4.5 Integrating Constraints in Editor

The author made the constraints integration mainly follow the step to step guide in the Epsilon Website, Live validation and quick-fixes in GMF-based editors with EVL. There are the steps to make constraints integration in the graph editor, it is slightly different than the original post.

- Create a evl file in the original editor.
- Right click StateFair, and select Plugin-in Tools, then open Manifest.
- In the dependencies, add org.eclipse.ui.ide and org.eclipse.epsilon.evl.emf.validation to the list of dependencies.

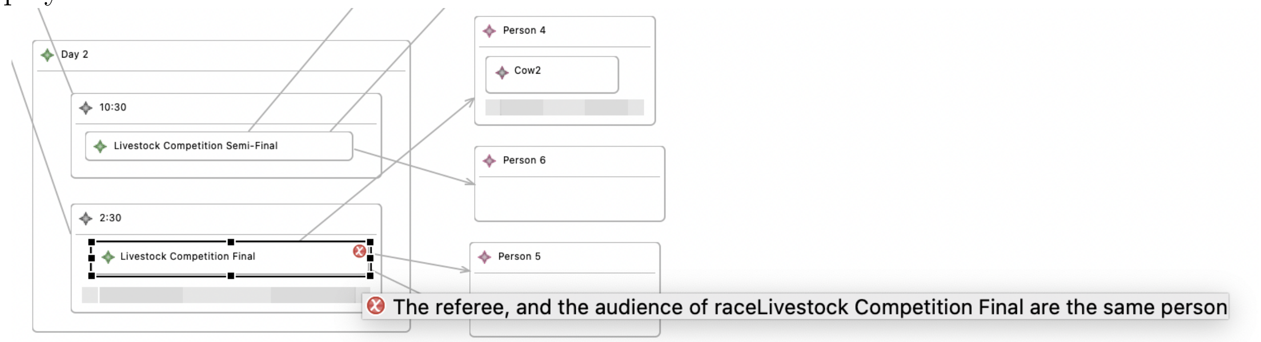
- In the extension, add org.eclipse.epsilon.evl.emf.validation. After that, right-click to add a new constraintBinding. This is the step to tell the program to bind this project with the evl file which contains constraints. In the constraint* field, browse the evl file, and select the directory of it.

- Add the org.eclipse.ui.ide.markerResolution in extension, and right-click add generic script with following code.
  class : org.eclipse.epsilon.evl.emf.validation.EvlMarkerResolutionGenerator
  class : markerType : org.eclipse.emf.ecore.diagnostic

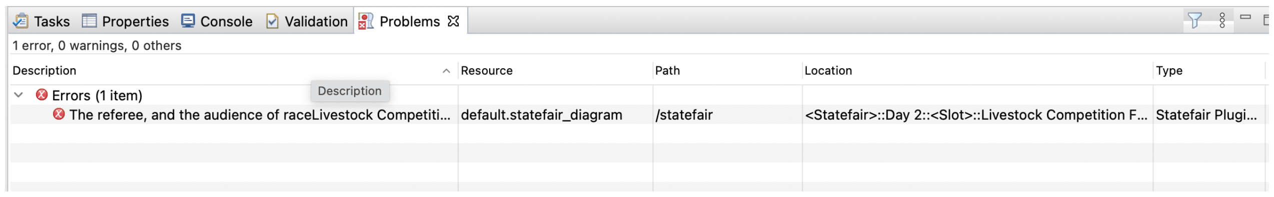
- Launch the project as a new instance, click the Edit, and select Validate

4.6 Integrated Constraints Errors

After integrating constraints into the graph editor, in-editor errors or warnings can be displayed in the editor.



The referee, and the audience of raceLivestock Competition Final are the same person



1 error, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (1 item)				
The referee, and the audience of raceLivestock Competiti...	default.statefair_diagram	/statefair	<Statefair>::Day 2::<Slot>::Livestock Competition F...	Statefair Plugi...

4.7 Difficulties

There are several unsolved problems in this project.

- How to return the sum of the duration of observations and races?

- How to get the total size of livestock in observations and races?

- One type of livestock may not get along with other types. For example, a male hog will act aggressively if another male hog is nearby. In order to make the event more manageable, the stalls need to be assigned base on certain rules. This will make things more complicated. A new attribute, such as type, could be introduced to distinct animal types in the Animal class. The Stall class has an attribute called location, this helps to indicate the stall location.
- The author tried to compare two times and determine which time is earlier time. The logic could be using ":" as a delimiter, then compare the hour and minutes. The author runs some issues with the program syntax, and not sure how and which way is the best to complete this task.

5 Model Transformation

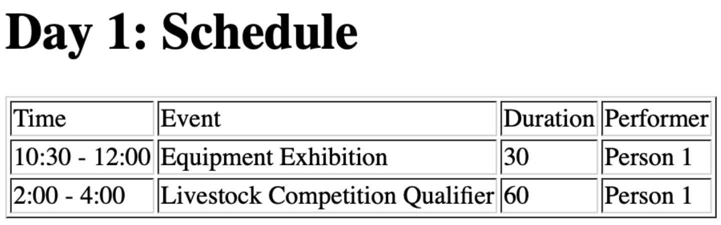
This model to text transformation is similar to the transformation presented in the lecture. However, the author expands the example and produces more complex html files to automate some work tasks. The following is what the html file will be generated.

- The event schedule on each day. It contains the start time and end time, event name, duration, and who is the performer.
- An overall stall list. It contains the name of the stall, the size, and the location tag.
- The name badge of all participants in the state fair event.

The implementation follows closely on the instruction of this tutorial - Code Generation Tutorial with EGL

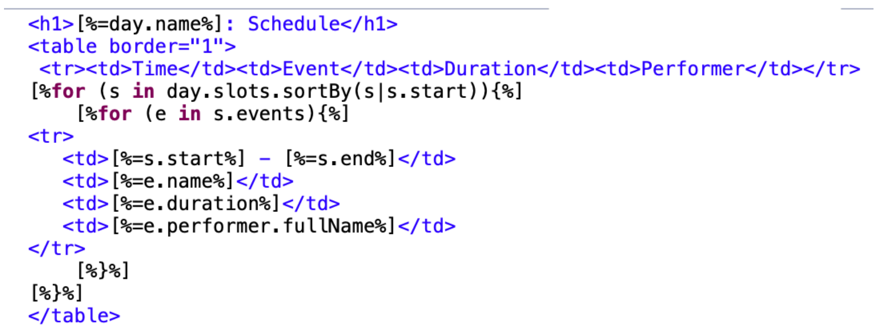
5.1 Event Schedule

An example of day 1 schedule.



Day 1: Schedule

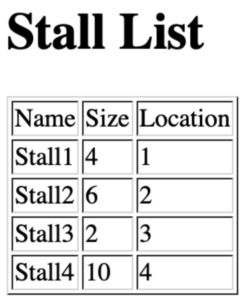
Time	Event	Duration	Performer
10:30 - 12:00	Equipment Exhibition	30	Person 1
2:00 - 4:00	Livestock Competition Qualifier	60	Person 1



```
<h1>[%=day.name%]: Schedule</h1>
<table border="1">
 <tr><td>Time</td><td>Event</td><td>Duration</td><td>Performer</td></tr>
[%for (s in day.slots.sortBy(s|s.start)){%]
    [%for (e in s.events){%]
<tr>
   <td>[%=s.start%] - [%=s.end%]</td>
   <td>[%=e.name%]</td>
   <td>[%=e.duration%]</td>
   <td>[%=e.performer.fullName%]</td>
</tr>
    [%}%]
[%}%]
</table>
```

5.2 Whole Stall List

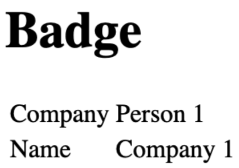
An example of a list of all stalls.


Stall List

Name	Size	Location
Stall1	4	1
Stall2	6	2
Stall3	2	3
Stall4	10	4

5.3 Badge for each Individuals

An example of a badge of a participant.


Badge

Company	Person 1
Name	Company 1

5.4 Analysis

The benefit of creating all different types of html files are enormous. From the practical view, it helps to keep the website, event booklets, and badge consistent. Moreover, it could save more time for a programmer to focus on other tasks, and save the company's resources in the long term.
The author also thinks there should be some integration between the web server and Epsilon. For example, how the web server automatically picks up the right html file which generated by Epsilon, and updates it on the web page. Another integration is sending the name badge to each participant via email. Right now, the EGL model only can produce the name badge for all participants. An email bot may be needed to automatically send the badge html file to each individual.

5.5 Difficulties

There are some difficulties the author faced in the implementation. For example, how to print a list that only contains referees? Only Race class has the attribute of the referee, and the Race class is a child class of Event. When looping through the Event list, some instances can be Observation class which does not have an attribute of the referee.

6 Summary

Creating a DSL is an extremely interesting topic. The project demonstrates how to use Emfatic/Ecore to generate a class diagram of the metamodel, integrating a graph editor in Eugenia, adding validation constraints in the editor, and transforming the model to other forms. The report also provides opinions/insights on each implementation, include advantages, disadvantages, and difficulties.

7 Reference

1. Live validation and quick-fixes in GMF-based editors with EVL

2. Code Generation Tutorial with EGL