ICS Helper

Ahila, Pawan, Shutong Wu, Yi-Leng Chen
[rameshra,kumarp40,wu867,cheny997]@mcmaster.ca

April 11, 2024

Abstract An ICS (iCalendar) file is a universal calendar format utilized by mainstream email and various calendar apps such as Microsoft Outlook, Google Calendar, and Apple Calendar. Importing an ICS file into calendar apps allows for the addition of new schedules comprising one or more events. Currently, the process of manipulating ICS files involves editing schedules within calendar apps and then exporting them. To address this time-consuming task, we propose ICS Helper, a Domain-specific Language (DSL) designed for handling ICS files. It's a text-based tool kit that allows users to generate schedules outside of calendar applications and provides integration possibility as an Eclipse plugin. With its user-friendly syntax, ICS Helper enables users to manage their schedules more effectively.

1 Introduction

An ICS file is a calendar file saved in a universal calendar format used by mainstream email and calendar apps, including Microsoft Outlook, Google Calendar, and Apple Calendar.

By importing ICS files into calendar apps, a new schedule(composed of one or more events) can be added. People can subscribe to schedules by URL.

Right now the only way to manipulate ICS files (other than writing the protocol by hand) is to edit schedules inside calendar apps/helpers, and then export, which hinders:

- easy creation of schedules outside the app environment
- mass operation (e.g. delay 100 events by an hour)
- programmable access (e.g. workflow automation)
- granular control(e.g. split and merge of events across schedules)

To tackle this, we propose the idea of "ICS Helper", which is a DSL dealing with ICS files. It offers:

- human-friendly syntax for ICS creation (write in plain text and compile to ICS)
- model validation (e.g. time conflict, missing fields)
- integration options with other tools as a plugin

While it doesn't solve all the problems mentioned, it provides a good starting point for further development.

The DSL will be implemented using `Xtext`, and the project will be shipped as a plugin for Eclipse. Our full code and documentation can be found at our Github repo at: `https://github.com/Gudauu/ICS-Helper/`.

2 Metamodel Definition and Analysis

The metamodel defined using Emfatic/Ecore serves as the foundation for the ICS Helper DSL. Below is the metamodel along with an analysis of its components, assumptions, and alternative design considerations.

2.1 Metamodel Overview

The metamodel defines the structure and relationships of entities within the ICS Helper DSL. It comprises five main classes: `Schedule`, `Location`, `Organizer`, `Invitee`, and `Event`. These classes encapsulate various aspects of scheduling and event management.

- **Schedule**: Represents a schedule containing multiple events. It includes attributes for name and description, along with associations with events and the schedule owner.
- **Location**: Represents the physical or virtual location of an event. It contains an attribute for the address.
- **Organizer**: Represents the organizer of a schedule or event. It includes attributes for name and email, as well as an association with a location.
- **Invitee**: Represents an individual invited to an event. It includes attributes for name and email.
- **Event**: Represents a specific event within a schedule. It includes attributes for title, description, start date, end date, and recurrence. It also has associations with attendees, the parent schedule, and the event location.

2.2 Assumptions

The metamodel assumes certain characteristics of schedules, events, organizers, and invitees:

- Events can have multiple attendees.
- Events are part of a schedule.
- Organizers and invitees have associated email addresses.
- Each event has a single location.

2.3 Alternative Design Considerations

While the current metamodel provides a solid foundation, alternative design considerations could enhance its flexibility and completeness:

- Event Recurrence: Consider defining a separate class for recurring events to handle more complex recurrence patterns.
- Polymorphism for Locations: Instead of a single `Location` class, explore the possibility of different location types represented by subclasses.
- Event Timezone: Incorporate timezone information for events, especially when dealing with events across different time zones.

2.4 Class Diagram

The following class diagram visually represents the structure and relationships defined in the metamodel:

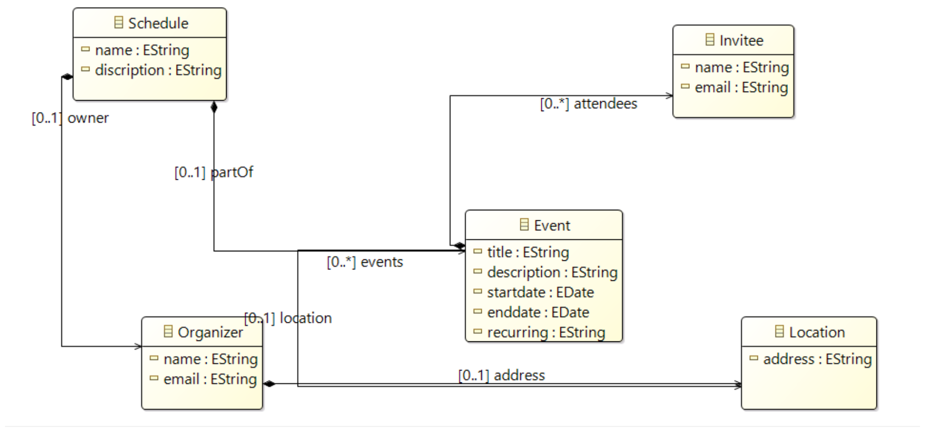


Figure 1: Class Diagram of the ICS Helper DSL Metamodel

The diagram illustrates the associations between the main entities, providing a visual reference for the metamodel.

This metamodel analysis and class diagram form the basis for the subsequent development and implementation of the ICS Helper DSL.

3 Defining and Implementing Syntax

We choose Xtext to implement the syntax and generate an editor accordingly for the following reasons:

- `Xtext`, as a text-based tool, provides flexible syntax control.
- Consider the use case of our DSL, usability in a text-based environment is a must.
- Better integration with other platforms and tools.

3.1 Syntax

Aiming to be user-friendly and intuitive, the syntax of ICS Helper DSL is designed to be concise and natural-language-like. The following sections outline the syntax elements and examples of the DSL.

3.1.1 Basic Syntax

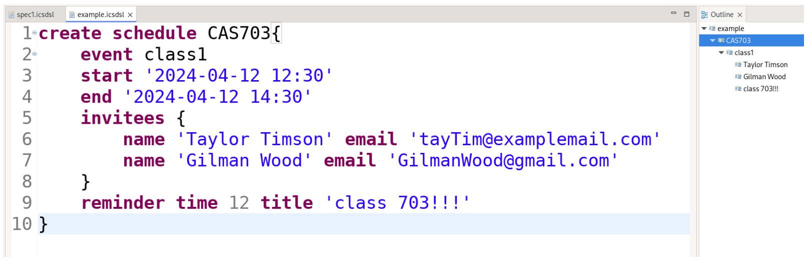
```
'create' 'schedule' UNIQUE_NAME '{'
    one or more Event
'}'
```

Where `Event` is defined as follows:

```
'event' UNIQUE_NAME
'start' YYYY-MM-DD HH:MM:SS
'end' YYYY-MM-DD HH:MM:SS
(OPTIONAL FIELDS)
```

OPTIONAL FIELDS include: `location`, `description`, `organizer`, `invitees`, `recurrence rule`, `reminder rule`, `link`. Please refer to our documentation for more details.

3.1.2 Example



```
create schedule CAS703{
    event class1
    start '2024-04-12 12:30'
    end '2024-04-12 14:30'
    invitees {
        name 'Taylor Timson' email 'tayTim@examplemail.com'
        name 'Gilman Wood' email 'GilmanWood@gmail.com'
    }
    reminder time 12 title 'class 703!!!'
}
```

Figure 2: Example Editor Screenshot

```
create schedule CAS703{
    event class1
    start '2024-04-12 12:30'
    end '2024-04-12 14:30'
    invitees {
        name 'Taylor Timson' email 'tayTim@examplemail.com'
        name 'Gilman Wood' email 'GilmanWood@gmail.com'
    }
    reminder time 15 title 'go to class 703!!!'
}
```

3.2 Discussion

Our syntax design using `Xtext` comes with the following advantages:

- Friendly syntax that is easy to understand and use.
- Editor that provides syntax highlighting, auto-completion, and error checking.
- Portability and integration with other tools (as our project is shipped as a plugin).

At the same time, it comes with the following limitations:

- Dependence on the Eclipse environment.
- Grammar limitations: like the need to follow the time format exactly, which hinders easy usage.
- Higher learning curve compared to graphical editors.

4 Validation constraints implementation

To better check the correctness, this project separates the validation into two parts.

4.1 Epsilon Validation Language (EVL)

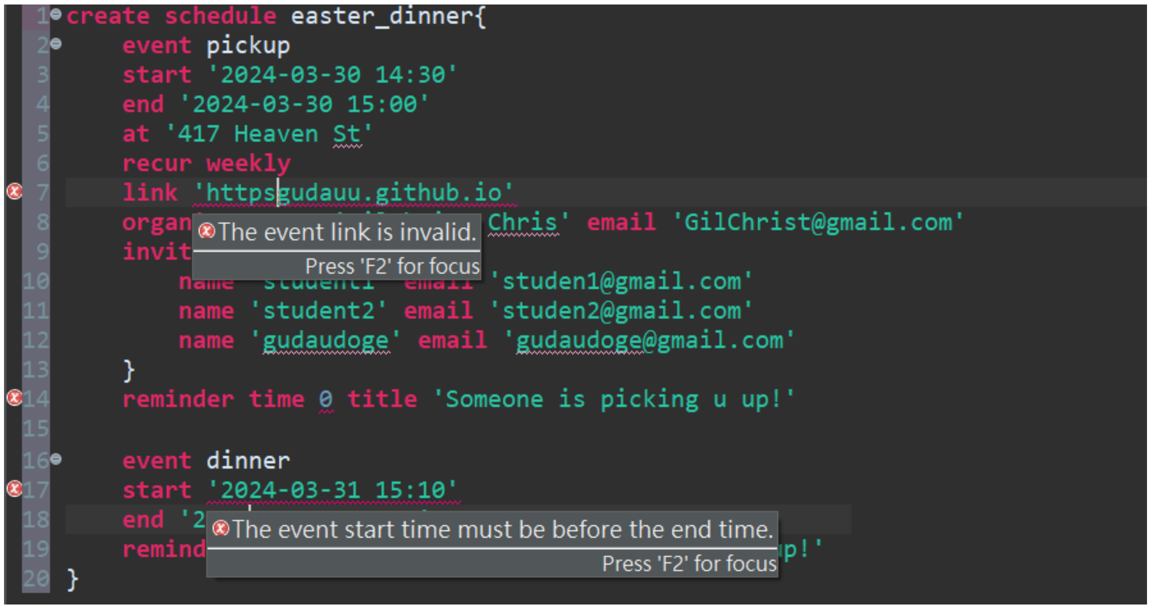
To validate aspects of the model that couldn't be expressed in the metamodel itself, we use EVL (Epsilon Validation Language) constraints to achieve our goal. Below are the constraints we have set up:

1. **Not null or empty constraints:**
    - `Schedule`: The name of the schedule must not be null or empty.
    - `Organizer`: The name of the organizers must not be null or empty.
    - `Invitee`: The name of the invitees must not be null or empty.
    - `Event`: The name of the event must not be null or empty.
2. **Unique constraints:**
    - `Organizer`: The email addresses of the organizers must not be duplicated.
    - `Invitee`: The email addresses of invitees must not be duplicated.
3. **Time constraints:**
    - `Event`: The event's start time must not be later than its end time.

4.2 Xtext editor validation

To enhance compatibility with the `Xtext` editor, we use Java for runtime validation of events. Below are the constraints we have established:

1. **Time constraints:**
    - The event's start time cannot be later than its end time.
    - The event's reminder time must not be less than 0 minutes.
2. **Email constraints:**
    - The emails of the organizer and invitees must follow the general email format.



```
create schedule easter_dinner{
    event pickup
    start '2024-03-30 14:30'
    end '2024-03-30 15:00'
    at '417 Heaven St'
    recur weekly
    link 'httpsgudauu.github.io'
    organ                        Chris' email 'GilChrist@gmail.com'
    invit
        name 'student1' email 'studen1@gmail.com'
        name 'student2' email 'studen2@gmail.com'
        name 'gudaudoge' email 'gudaudoge@gmail.com'
    }
    reminder time 0 title 'Someone is picking u up!'

    event dinner
    start '2024-03-31 15:10'
    end '2
    remind                                             p!'
}
```

The event link is invalid.
Press 'F2' for focus

The event start time must be before the end time.
Press 'F2' for focus

Figure 3: Xtext validation example

- The emails of the organizer and invitees cannot be null or empty.
- The emails of the organizer and invitees must be unique and not repeated.

3. **Additional constraint:**
    - The event link should resemble a URL, i.e., it should contain "://" to indicate it is a URL.

The figure above displays examples of running the validation code, checking the start and end times of the event, as well as verifying the event link.

5 Model management operation implementation

Since the goal of our DSL is to generate ICS files, we implemented the model-to-text transformation operation.

The operation takes a .icsdsl file (our DSL file) as input and generates a corresponding .ics file as output, which can be further imported into calendar applications.

5.1 Implementation

The operation is implemented using `Xtend`, a statically-typed programming language that compiles to Java. `Xtend` is used in accordance with `Xtext` to provide seamless code translation and execution.

The operation reads the .icsdsl file, then for each `CreateCommand`, reads the values assigned to each mandatory and optional field, and translates them into the corresponding .ics format.

For instance, the below `Xtend` code translates the `organizer` field.

```
if (event.organizer !== null) {
    icsContent.append("ORGANIZER;CN=" +
    event.organizer.name + ":mailto:"
    + event.organizer.email + "\n")
}
```

5.2 Example

The example ICS snippet shown in 3.1.2 will be transformed into CAS703.ics with content below:

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN
BEGIN:VEVENT
SUMMARY:class1
DTSTART:20240412T123000Z
DTEND:20240412T143000Z
ATTENDEE;CN="Taylor Timson":MAILTO:tayTim@examplemail.com
ATTENDEE;CN="Gilman Wood":MAILTO:GilmanWood@gmail.com
RRULE:FREQ=DAILY
```

```
BEGIN:VALARM
TRIGGER:-PT12M
ACTION:DISPLAY
DESCRIPTION:class 703!!!
END:VALARM
END:VEVENT
END:VCALENDAR
```

5.3 Limitations and future work

One limitation in the model transformation (and subsequently affecting the syntax definition) would be the strict syntax requirements.

For instance, the time should be specified in exactly "YYYY-MM-DD HH:MM:SS" format, which might be inconvenient.

This can be easily (but time-consumingly) solved by supporting more time formats in the `Xtend` transformation code.

6 References

1. J. Hooman, *Creating a Domain Specific Language (DSL) with Xtext*. Retrieved from `https://www.cs.ru.nl/J.Hooman/DSL/Creating_a_Domain_Specific_Language_(DSL)_with_Xtext.pdf`

2. Xtext Documentation, *Domain Model Example*. Retrieved from `https://eclipse.dev/Xtext/documentation/102_domainmodelwalkthrough.html`

3. Introduction to Model Validation and Code Generation with *Epsilon* `https://www.youtube.com/watch?v=Z2MJ_oBcfYE`

4. The Epsilon Validation Language (EVL) `https://eclipse.dev/epsilon/doc/evl/`

5. Language Implementation `https://eclipse.dev/Xtext/documentation/303_runtime_concepts.html#validation`