A Comparison of Different Modelling Tools

CP8309 (MDE)

Urmi Patel
MSc Computer Science
Ryerson University
Toronto, Ontario

Introduction

A language is a set of words, also called strings or sentences, which are composed by a set of symbols of an alphabet. There are two types of languages available, a general-purpose language (GPL) and a domain-specific language (DSL). GPL is a language for describing systems in multiple domains, for example Java, UML and C. DSL is used to describe specific languages into a particular application domain, for example CSS or HTML. Among the many classifications of DSL, one of key importance is notation. Notations can be divided into two types: textual representation and graphical representation. Graphical languages are developed with the help of visual models and its graphical items such as arrows, blocks and symbols.

The primary goal of this project is to utilize and conduct a direct comparison of four different tools to create the domain specific language for a particular domain. Hotel booking application system (HBAS) is a simple domain specific language which was used for this project, where HBAS is used to book a hotel and room by the user. Abstract and concrete syntax are considered to be important additional components of DSL. Abstract syntax, specified by the metamodel, describes the concepts of the language, their relationship and structuring rules. Concrete syntax provides the realization of the abstract syntax of a metamodel as a mapping between the metamodel and their graphical or textual representation.

The tools used for this project were, Sirius, Xtext, MetaEdit+ and AToMPM. The tasks performed by all four tools are as follows: Abstract syntax created for the HBAS language, concrete syntax defined, then the model was created and lastly, verification with some constraints.

1 Sirius Tool

Sirius is a framework based on the eclipse graphical modelling framework (GMF). Although it deals with large metamodels, it is still considered to be very fast. The model defined in Sirius is interpreted, not compiled, hence any change on the concrete syntax specification has an immediate effect on the diagram editor. There are mainly two types of environments available: a specific environment for developers and a runtime environment for end users. Sirius has many editing features which make it user-friendly. The filters feature is used for mapping, where users can hide/show elements of the model according to a specific requirement. A diagram can define any number of filters which can be enabled or disabled instantaneously. Validation rules are used to evaluate the quality of a model, where many different programming languages are used to define the rules, such as Acceleo query language (AQL), java services, using raw OCL and so on. Furthermore, in the properties view section, advanced features such as customizable colors palette, viewpoint selection option, error management and navigation between representation can be found.

For developing the DSL, the first step is to create an ecore modeling project and then open the .aird file where the abstract syntax can be defined. The diagram below is an example of abstract syntax, where the class diagram is defined with attributes: names and types.

To create a concrete syntax, the genmodel file was run and a new runtime eclipse workspace was opened. Inside the workspace, a new modeling project was created and the .odesign file opened in which new elements can be defined for the model. Inside this file, validation rules were defined using the Acceleo query language and one mapping filter was defined.

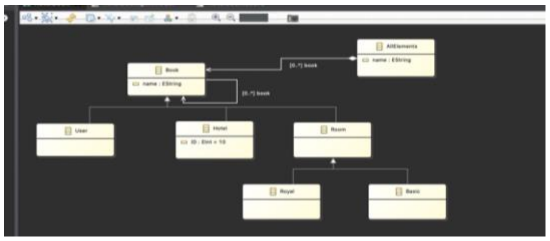
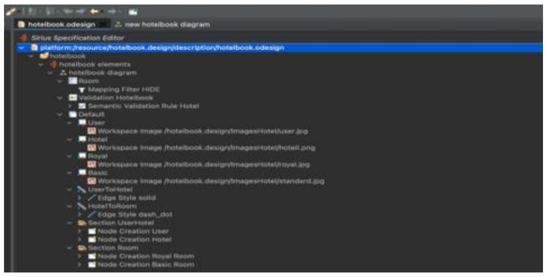


Figure 1: Abstract Syntax of Sirius Tool



```
hotelbook.odesign    new hotelbook diagram
Sirius Specification Editor
platform:/resource/hotelbook.design/description/hotelbook.odesign
  hotelbook
    hotelbook elements
      hotelbook diagram
        Room
          Mapping Filter HIDE
        Validation Hotelbook
          Semantic Validation Rule Hotel
        Default
          User
            Workspace Image /hotelbook.design/ImagesHotel/user.jpg
          Hotel
            Workspace Image /hotelbook.design/ImagesHotel/hotell.png
          Royal
            Workspace Image /hotelbook.design/ImagesHotel/royal.jpg
          Basic
            Workspace Image /hotelbook.design/ImagesHotel/standard.jpg
          UserToHotel
            Edge Style solid
          HotelToRoom
            Edge Style dash_dot
          Section UserHotel
            Node Creation User
            Node Creation Hotel
          Section Room
            Node Creation Royal Room
            Node Creation Basic Room
```

Figure 2: Concrete Syntax of Sirius Tool

Next, new presentation option was selected, and a new diagram called valid or invalid model, was created through the palette. This created model can be used by the user side and user can validate the diagram.

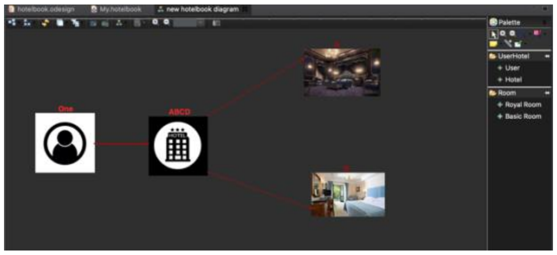
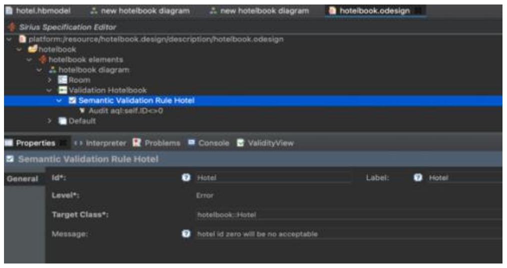


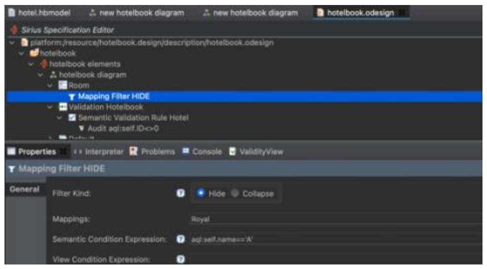
Figure 3: Model diagram of Sirius Tool

Below figure shows defined constraints and mapping filter of the project where constraints were defined using AQL language.



```
hotel.hbmodel    new hotelbook diagram    new hotelbook diagram    hotelbook.odesign
Sirius Specification Editor
platform:/resource/hotelbook.design/description/hotelbook.odesign
  hotelbook
    hotelbook elements
      hotelbook diagram
        Room
        Validation Hotelbook
          Semantic Validation Rule Hotel
            Audit aql:self.ID<>0
        Default

Properties    Interpreter    Problems    Console    ValidityView
Semantic Validation Rule Hotel
General   Id*:            Hotel                       Label:   Hotel
          Level*:         Error
          Target Class*:  hotelbook::Hotel
          Message:        hotel id zero will be no acceptable
```

Figure 4: Constraints in Sirius Tool



```
hotel.hbmodel    new hotelbook diagram    new hotelbook diagram    hotelbook.odesign
Sirius Specification Editor
platform:/resource/hotelbook.design/description/hotelbook.odesign
  hotelbook
    hotelbook elements
      hotelbook diagram
        Room
          Mapping Filter HIDE
        Validation Hotelbook
          Semantic Validation Rule Hotel
            Audit aql:self.ID<>0

Properties    Interpreter    Problems    Console    ValidityView
Mapping Filter HIDE
General   Filter Kind:                    (●) Hide  ( ) Collapse
          Mappings:                       Royal
          Semantic Condition Expression:  aql:self.name=='A'
          View Condition Expression:
```

Figure 5: Mapping Filter of Sirius Tool

After creating all valid or invalid models, the user can validate the diagram or model and if the validation rules don't match, an error message will show. The below figure shows an example of defined constraint with a small red cross symbol mark on hotel symbol which states that the hotel id cannot be zero.

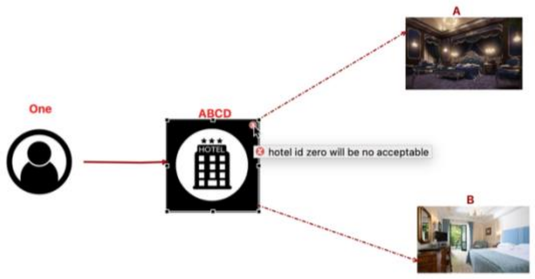


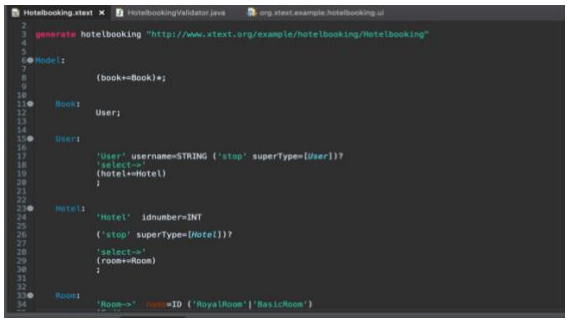
Figure 6: Error message in Sirius Tool

2 Xtext Tool

Xtext is an open-source framework used to define the textual domain specific language. It develops domain specific language based on ANTLR grammars, where grammar is an important part of the DSL and this grammar is defined by the EBNF (it is used to specify lexer and parser). ANTLR generates parser from the grammar after-which the parser can build abstract syntax trees. Xtext automatically generates textual concrete syntax infrastructure, such as linker and parser.

There are many advanced language specific features available for the xtext tool such as, validation, syntax coloring, refactoring, navigability and auto-editing among many others. Outline view of the editor shows the complete picture of the active diagram.

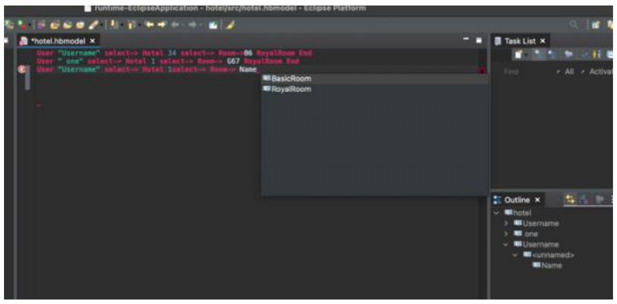
Firstly, xtext project would be created and DSL file extension defined, which will be used for files in the future. The .xtext file of the project is used to define the grammar, which is also known as concrete syntax.



```
Hotelbooking.xtext  HotelbookingValidator.java  org.xtext.example.hotelbooking.ui

generate hotelbooking "http://www.xtext.org/example/hotelbooking/Hotelbooking"

Model:
            (book+=Book)*;

    Book:
            User;

    User:
            'User' username=STRING ('stop' superType=[User])?
            'select->'
            (hotel+=Hotel)
            ;

    Hotel:
            'Hotel'  idnumber=INT
            ('stop' superType=[Hotel])?
            'select->'
            (room+=Room)
            ;

    Room:
            'Room->'  name=ID ('RoyalRoom'|'BasicRoom')
```

Figure 7: Concrete Syntax of Xtext Tool

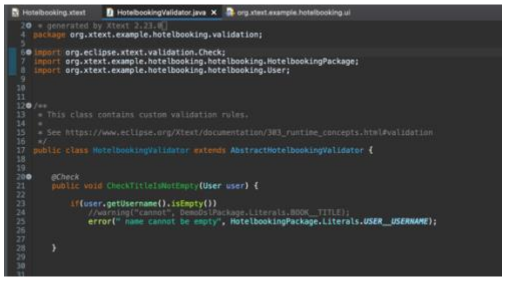
To compile the grammar, right click on the grammar editor and select as generate xtext artifacts, which will generate the parser and text editor. Next, run the project and open a new eclipse workspace. Inside the workspace, a new project will be created and define the model. The figure below shows the model lines which was defined using grammar.



```
runtime-EclipseApplication - hotel/src/hotel.hbmodel - Eclipse Platform
*hotel.hbmodel
User "Username" select-> Hotel 34 select-> Room->B6 RoyalRoom End
User " one" select-> Hotel 1 select-> Room-> G67 RoyalRoom End
User "Username" select-> Hotel 1select-> Room-> Name
```

Figure 8: Model of Xtext Tool

Xtext automatically derive the ecore model from the grammar file, so here the ecore model is the abstract syntax of the project. Constraints can be defined inside the validator.java class file using java interfaces. Below figure shows the example of validation rule which states, username cannot be empty inside the model, if it is, then it will show the error message.



```
Hotelbooking.xtext  HotelbookingValidator.java  org.xtext.example.hotelbooking.ui
 * generated by Xtext 2.23.0
package org.xtext.example.hotelbooking.validation;

import org.eclipse.xtext.validation.Check;
import org.xtext.example.hotelbooking.hotelbooking.HotelbookingPackage;
import org.xtext.example.hotelbooking.hotelbooking.User;



/**
 * This class contains custom validation rules.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#validation
 */
public class HotelbookingValidator extends AbstractHotelbookingValidator {


    @Check
    public void CheckTitleIsNotEmpty(User user) {

        if(user.getUsername().isEmpty())
            //warning("cannot", DemoDslPackage.Literals.BOOK__TITLE);
            error(" name cannot be empty", HotelbookingPackage.Literals.USER__USERNAME);


    }
```

Figure 9: Validation rule of Xtext Tool

3 AToMPM

It is an open-source framework to design domain specific environment where all tasks can be performed on the cloud via a web browser. Most of the interactions within the AToMPM editor are done with a few mouse clicks whether it is to edit models or to load toolbars in AToMPM canvas. The dock is available at the top of the canvas where formalism toolbars and buttons are available. There are many advanced features available such as, ModelShare, ScreenShare where ModelShare use to share the abstract syntax of a model between clients and ScreenShare used to share the exact same canvas between two or more clients. AToMPM is an online IDE so multiple users can be logged-in at the same time with their individual view of the models. AToMPM relies on specific file extensions to understand model types.

Abstract syntax can be defined using class diagram and that file is saved as a "filename + MM.model". Next, it is compiled and the metamodel form for the class diagram is generated and saved as a "filename.metamodel".

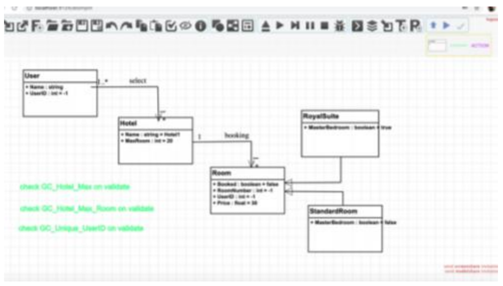

User
select
Hotel
booking
RoyalSuite
Room
StandardRoom
check GC_Hotel_Max on validate
check GC_Hotel_Max_Room on validate
check GC_Unique_UserID on validate

Figure 10: Abstract Syntax of AToMPM Tool

Icon instances, link instances and text instances are used to define the concrete syntax. Next, save this file as a "modelname + .concretesyntaxname + Icons.model" and compile it. Below figure shows the symbol instances for hotelbooking system.

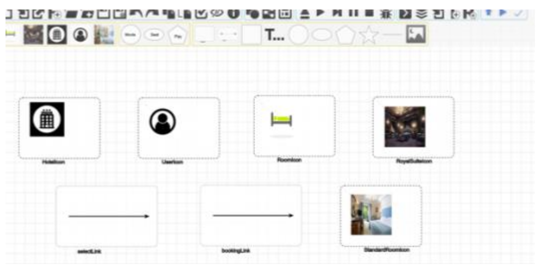


Figure 11: Concrete Syntax of AToMPM Tool

Lastly, the formalism toolbar of the model is loaded and create some valid or invalid models and save them as a “name + .model”. Furthermore, verify button allows to validate the model whether it was correct or not. Below figure shows the valid model, but if the model is invalid then it will show the error or waring message with a popup.

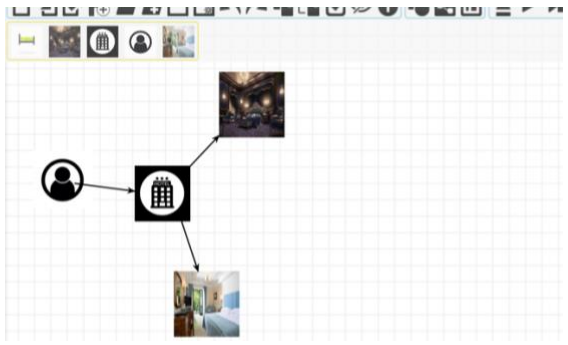
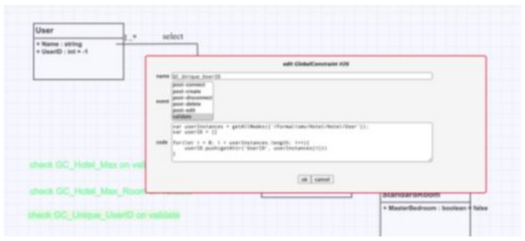


Figure 12: Model of AToMPM Tool

Two type of constraints can be defined within AToMPM; Static constraints and global constraints. These constraints can be expressed using JavaScript API. Below figure shows a global constraint example for the hotelbooking system, it says that user id must be unique.



edit GlobalConstraint #26

name: GC_Unique_UserID

event: post-connect, post-create, post-disconnect, post-delete, post-edit, validate

code:

```
var userInstances = getAllNodes(['/Formalisms/Hotel/Hotel/User']);
var userID = [];

for(let i = 0; i < userInstances.length; i++){
    userID.push(getAttr('UserID', userInstances[i]));
}
```

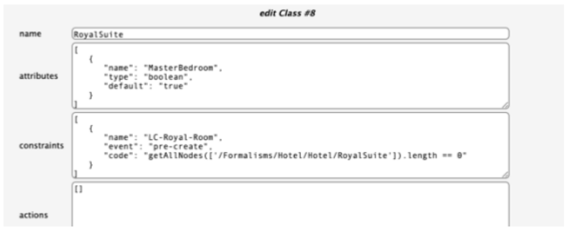
ok | cancel

check GC_Hotel_Max on vali

check GC_Hotel_Max_Room

check GC_Unique_UserID on validate

Figure 13: Global Constraint of AToMPM Tool



edit Class #8

name: RoyalSuite

attributes:

```
[
    {
        "name": "MasterBedroom",
        "type": "boolean",
        "default": "true"
    }
]
```

constraints:

```
[
    {
        "name": "LC-Royal-Room",
        "event": "pre-create",
        "code": "getAllNodes(['/Formalisms/Hotel/Hotel/RoyalSuite']).length == 0"
    }
]
```

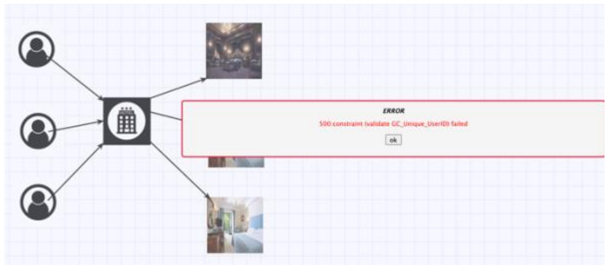
actions:

```
[]
```

Figure 14: Static Constraint of AToMPM Tool

Local constraint can be defined inside a specific class property using events and codes. The figure below shows code of the local constraint for royal room of the hotel.

There is a verify button inside the toolbar, after creating the model where users can click the button and it will check all validation. Global constraints perform checks while verifying the model whereas local constraints perform checks while creating the model. The below figure shows an example of one constraint and how it will provide an error message.



ERROR

500 constraint (validate GC_Unique_UserID) failed

ok

Figure 15: Error message in AToMPM Tool

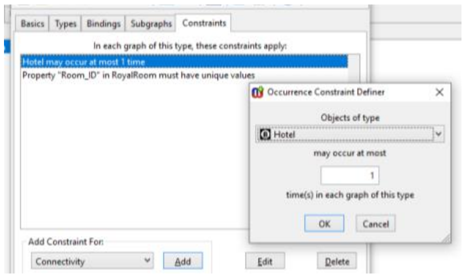
4 MetaEdit+

It is an open environment for using or creating domain specific modelling languages. MetaEdit+ consists of two separate tools; one is MetaEdit+ workbench to design modelling languages and the second is MetaEdit+ modeler to use modelling languages. This tool uses GOPPRR (graph, object, port, property, relationship, role) meta-modelling language for modelling, where graph is used to define one language or diagram techniques, object describes the basic concepts of a modelling language, properties are used to characterize attributes and relationship is defined as a connection of properties or elements. The actual semantics of the graph are defined as the bindings of the object, relationships, roles and ports within the graph. Graph, objects and relationships can have attributes and information which is called as properties, where information can be any type like string, collections, numbers and text.

MetaEdit+ contains some features such as synchronized models, inbuilt constraints, repository connection, meta-model browser, customized symbol editor and so on. The meta-modelling state is synchronized, so if a user changes anything in tool and properties than it will automatically change for current model. Abstract syntax is defined as a GOPPRR language. To create the abstract syntax, firstly create new metamodel and add types to it with the help of object tool. Secondly, define each object with symbol editor and last but not least, use the graph tool to define relationships, roles and objects.

Concrete syntax can be defined using vector tool or importing graphics files; hence developer can specify custom symbols for objects, roles and relationships in their models. Symbols may contain text or property values which are extracted from the abstract syntax objects. MetaEdit+ supports several kinds of constraints; object connectivity with a relationship or role, object occurrence in a model, ports involved in a relationship and property uniqueness.

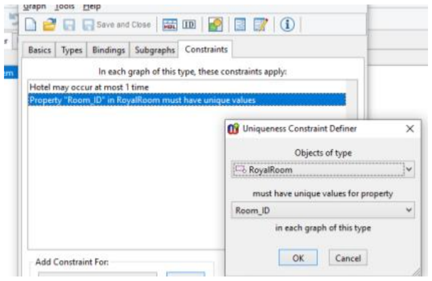
For this project, two types of constraints were defined; one for unique room id values for each room of the system and another one for hotel which is used to define the number of occurrences of hotel object in a system.



In each graph of this type, these constraints apply:
Hotel may occur at most 1 time
Property "Room_ID" in RoyalRoom must have unique values

Occurrence Constraint Definer
Objects of type: Hotel
may occur at most: 1
time(s) in each graph of this type

Figure 16: Constraint define in MetaEdit+

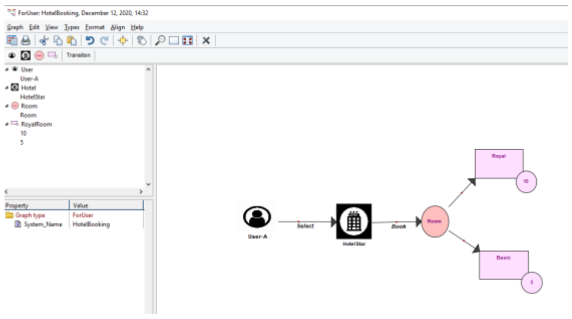


In each graph of this type, these constraints apply:
Hotel may occur at most 1 time
Property "Room_ID" in RoyalRoom must have unique values

Uniqueness Constraint Definer
Objects of type: RoyalRoom
must have unique values for property: Room_ID
in each graph of this type

Figure 17: Constraint define in MetaEdit+

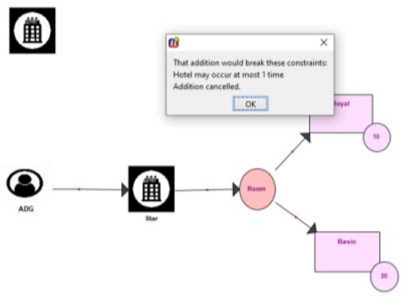
Finally, the model was created using defined symbols. A graphical meta model is described as a set of objects in the GOPPRR language. Images and symbols are representing the objects, and arrows represent the relationship between two objects. Actions or roles are used to form the bindings of different objects. In the below figure, select and book are the actions and user, hotel, room are the objects of the hotelbooking system.



Property	Value
Graph type	ForUser
System_Name	HotelBooking

Figure 18: Model of MetaEdit+ Tool

While creating the model, defined constraints were checked and if created model element doesn't match with the carted rules, then it will show an error. Below figure shows an example of constraints error while the user tries to create another hotel within a system.



That addition would break these constraints:
Hotel may occur at most 1 time
Addition cancelled.

Figure 19: Error message in MetaEdit+ Tool

5 Practical Comparison of modeling tools

Below table shows the comparison of different modelling tools from a practical basis. AToMPM tool was hard to understand and use because of the lack of documentation and videos, and so it took time to understand the flow. In furthering this point, the customization level was not good as compared to other tools. It was a very useful tool for academic purposes and as such, it gave a basic idea about the modeling language. Navigability of the tool was low because the tool requires specific name and extension and there is no auto completion option, so manual checking is required. In the end, it was very easy to use after fully understanding it.

The customization level of the Sirius tool was very high because of the auto completion option, color palette and quick fix options. Graphical expressiveness and navigability were high as compared to other tools because it uses specific symbols for a particular task and one can easily create files using different navigation options. If one is familiar with the eclipse environment, it will be easy to use Sirius tool as the modelling tool of choice. Sirius is mostly used for industry purposes.

Xtext is a textual editor as the concrete syntax defined using grammar. Developing domain specific language using Xtext is comparatively easy for developers.

On the other hand, users can easily understand graphical language than graphical editors are more user friendly. Again, this tool is based on the eclipse framework and is free of cost. The main disadvantages of this tool are low graphical completeness and low navigability as compared to other tools. Large amounts of documentation and videos are available, to assist in understanding the flow of modeling. Often times, it was easy to deal with textual data as compared to graphical data because graphical data has many properties like height, name, color and side but textual data has only text so one can easily deal with those data.

MetaEdit+ is a commercial tool (free for first 30 days) so it is mainly used in industry. Customization level and efficiency are low; hence it is hard to develop modelling language. There is no auto completion option, so the user has to remember each word and add attributes and objects with correct naming. Learning time for this tool is less as compared to other tools but at the same time the efficiency is not the best. All tools have their own specialty and uniqueness, and users have to choose the modelling tool as per their requirements, for example a student can use AToMPM to learn the DSL.

The given details (speed of development, ratings) of each tool defined as per developers' point of view, it may be different for each user as per them ability.

Table 1: Practical Comparison of different tools

	AToMPM	**Sirius**	**Xtext**	**MetaEdit+**
Editors	Graphical	Graphical	Textual	Graphical
Speed of development *	10 days	6 days	4 days	3 days
Documentation/ Videos	3/5	4/5	5/5	3/5
Cost	Free	Free	Free	Commercial
Customization level	3/5	5/5	4/5	3/5
Efficiency	80%	85%	87%	40%
Graphical Expressiveness	High	High	Low	Low
Navigability	Low	High	Low	Low
Update time	Sometimes	Regularly	Regularly	Sometimes
Target audience	Academic	Industry	Industry	Industry

6 Technical Comparisons of Tools

Currently, Mac os version of MetaEdit+ is not available, hence users have to use Windows or Linux version of the tool. The developer's workspace is maintained online through a repository, which means all your work is kept safe. Constraints specification for MetaEdit+ is limited, but if the developer wants to specify more advanced constraints, the MERL generator should be used. MetaEdit+ also offers an extensive Java API to communicate with models.

Sirius and xtext are an eclipse-based framework while other two have their own frameworks. In AToMPM, code generation can be done manually while for MetaEdit+, it uses MERL language. Xtext uses the XTend which is an extension of java and Sirius uses genmodel to generate editor file which is used to run the project. Use of MetaEdit+ is quite straightforward but it can be time consuming when the number of objects and relationships increases. Sirius used ecore model as an abstract syntax and at the same time Xtext used ecore model as a concrete syntax.

Models can be defined using any symbols or images because the developer had to make it more user friendly. Refactoring means when a developer renames a language element then it will automatically update all references to it, hence AToMPM doesn't support this feature. AToMPM uses class diagram as a domain model while Sirius uses ecore diagram as a domain model.

AToMPM is more sensitive for bugs and errors as compared to MetaEdit+ tool. JavaScript is used to define constraints inside the AToMPM, AQL query language used to define validation rules inside the Sirius tool and Java interfaces used to define the validations in Xtext tool. Modified symbols are used to define concrete syntax while xtext grammar are used to define concrete syntax of Xtext tool. Table 3 describes the structure for each tool, how to define abstract syntax, concrete syntax and model for domain specific language.

Table 2: Technical Comparison of different tools

	AToMPM	**Siris**	**Xtext**	**MetaEdit+**
Code generation	Manual	Genmodel	XTend	MERL language
Domain model	Class diagram	Ecore diagram	Textual	Graph model
Operating	Windows, Linux, Mac	Windows, Linux, Mac	Windows, Linux, Mac	Windows, Linux
Language	Python	Java	Java	Java API
Framework	Own	Eclipse	Eclipse	Own
Refactoring	No	Yes	Yes	Yes
Abstract syntax	Class diagram model	Ecore model	Ecore model	GOPPRR
Concrete syntax	Concrete syntax model	Odesign	Xtext grammar	Vector tool
Hierarchy	Yes	Yes	Yes	No
Constraints	JavaScript	AQL query	Java class	Limited (inbuilt)

Table 3: Modelling Process comparison of different modelling tools

Tools	Abstract Syntax	Concrete Syntax	Use DSL
AToMPM	• Load class diagram toolbar • Develop class diagram model • Save it • Load compile menu toolbar • Compile abstract syntax	• Open new tab • Load concrete syntax toolbar • Develop concrete syntax model • Save it • Load compile menu toolbar • Generate toolbar	• Open new tab • Load new toolbar • Develop models and save • Can verify models
Sirius	• Create ecore modelling project • Open .aird file and create class diagram of language • Generate .editor file from genmodel • Run the genmodel file	• Open runtime eclipse workspace • Create new modelling project • Make it viewpoint specification project • Open .odesign file and define new elements of model	• Select the new presentation option and generate new diagram where you have palette to create any model
Xtext	• Automatically derive the ecore model from a grammar • To understand ecore model easily, visualize it as a ecore diagram (.ecore file)	• Create the xtext project • Open .xtext file and define grammar • Save and compile it • Generate xtext artifacts for the language infrastructure	• Open runtime workspace • Create new java project • Add new file to src folder • Continue by editing that file and use as a model
MetaEdit+	• After login, select repository and create new project • Create objects • Open graph tool menu and define name and graph type • Add property and save • Open types page and develop types	• For each type, open symbol browser menu and develop symbol • Save • commit	• Create graph menu and add parameters • Develop model • commit

7 Conclusion

Hotelbooking system is a simple domain specific language used for this project. The developer attempted to make it simpler because the main goal of this project is to analyze different tools and compare those tools with each other. Comparison done within this project can be different from others point of view, hence version of the tools changes as time passes.

Clearly, every tool has its own advantages and disadvantages; this project gives an idea of each tool. Individual users can choose any tool of their choice based on their requirements and usability.

REFERENCES

[1] J. C. M. W. Marco Brambilla, Model-Driven Software Engineering in Practice: Second Edition, Morgan & Claypool , 2017.

[2] "Eclipse-Sirius," [Online]. Available: https://www.eclipse.org/sirius/doc/.

[3] "Eclipse-Xtext," [Online]. Available: https://www.eclipse.org/Xtext/documentation/.

[4] "Metacase," [Online]. Available: https://www.metacase.com/support/40/manuals/. [Accessed 2020].

[5] "AToMPM," [Online]. Available: https://msdl.uantwerpen.be/documentation/AToMPM/. [Accessed October 2020].

[6] F. Biryukov, "Textual languages with Xtext," University of Antwerp.

[7] R. P. S. K. Juha-Pekka Tolvanen, "Advanced Tooling for Domain-Specific Modeling: MetaEdit+," Finland.