SysML 2.0 Viewer in a Browser

Oliver Högberg, olihgb-7@student.ltu.se
Emma Carlsson, emacar-8@student.ltu.se
Axel Kärnebro, axekrn-7@student.ltu.se
Magnus Stenfelt, magset-8@student.ltu.se

March 2020

Contents

1 Introduction **3**
1.1 Background . . . 3
1.2 Problem description . . . 3
1.3 Objectives . . . 3

2 System design **4**

3 Results **7**
3.1 Delivery . . . 7
3.2 Testing . . . 9
3.2.1 Risks of regression . . . 9
3.2.2 Strategy for regression testing . . . 9
3.2.3 Development description . . . 10
3.2.4 Test strategy . . . 10

4 Ethical assessment **11**

5 Conclusion and future work **12**

Appendices **13**

1 Introduction

1.1 Background

When developing large and complex systems it is very important not to lose understanding of the system, however this can be quite a challenge. This is why developers have used some kind of modelling language throughout time. Modelling languages allow for creation of a graphical representation of the system, with varying levels of detail and complexity depending on what the project in question needs. It also allows developers to create and plan the entire system ahead of time, making it easier to avoid messy implementations and improves overall readability. UML [3], Unified Modeling Language, is one such language and SysML [4], Systems Modeling Language, is an extension of UML which lets users practice Model-Based Systems Engineering [5] (MBSE).

The firs iteration of SysML is a complete modelling language with very useful tools for creating and viewing system models. It is also a graphical modeling language meaning it consists of different types of diagrams which contain information useful for conveying system structures. There is now a new iteration of SysML called SysML v2 which is a purely textual modeling language meaning SysML v2 looks more like a programming language. This creates a problem where less tech-literate users have a very hard time understanding SysML v2 as it has a very steep learning curve and the systems are very hard to visualize. To make SysML v2 more palpable for users of varying technical proficiency tools need to be developed to translate the SysML v2 text to diagrams. There are some tools that accomplish this to some degree, such as Eclipse Papyrus [6], but they are very complicated to install and use making them unsuitable for less tech-literate users.

A proof of concept of an interactive SysML v2 web viewer has been developed in a bachelor thesis at LTU where Jesper Nilsson [1] proved that it is possible to render SysML diagrams in a browser. Tommy Andersson [2] would later continue the project by building on the proof of concept as a summer job, however it was discovered that a proof of concept does not necessarily scale and their solution could not be developed any further. Additional work is required to restart the project from scratch to avoid these scalability issues.

This is why this current iteration of the project was started, to find a way around the issues that the prior project iterations arrived at and to create a strong foundation towards a finished product; an interactive SysML v2 viewer in a browser.

1.2 Problem description

The task is to create a web viewer for SysML v2. The viewer is supposed to be used by a system designer to create a clearer and easier to understand description of a complex system. In order to accomplish this the application needs to be able to display the diagrams included in SysML v2 based on a input file in the format of a .sysml file. After creating the .sysml file, it will be uploaded to the application and shown as a SysML v2 diagram in a clear and interactive manner. The final product is supposed to have support for a user to: interact with the diagram views; click SysML objects to get to other diagram views and to view specific SysML object functionality.

The goal of this project is get an understanding of the SysML v2 language to create an easy to use tool to display future SysML diagrams. Another very important goal, if not the main goal itself, is to create the application in such a way that it can be built upon by other developers in the case of this project not completing the first mentioned goal. The application also has to be able to be built upon due to SysML v2 not being finished and it still being updated. To summarize this to one sentence the project is to create a scalable SysML v2 web viewing tool that is easy to use and understand.

1.3 Objectives

The overall objective with this project is to create a tool that allows the user to get a clear understanding of a complex system. This tool should be able to give both an overview of the system and show the small details and how all the parts are connected. The systems provided to the tool should be visualized using the SysML diagrams and written with the syntax of SysML v2 which is still in development. A very important part of this project is to make it open and scalable, our main goal is to lay the groundwork so that others can build upon and perfect it.

This shall be demonstrated at the end of the project by running an example project written in SysML v2 on our application which should be able to parse the code, display at least one diagram and be interactive in such a way that the user can navigate from one diagram view to another. This diagram should be either a block definition diagram, internal block diagram or a sequence diagram.

To reach this objective we need to make sure that we, as developers really understand what we are creating and that we have a sound and well thought through plan to avoid getting stuck. We also need to use the former attempts at this project to learn what worked and what to do differently in order to succeed.

2 System design

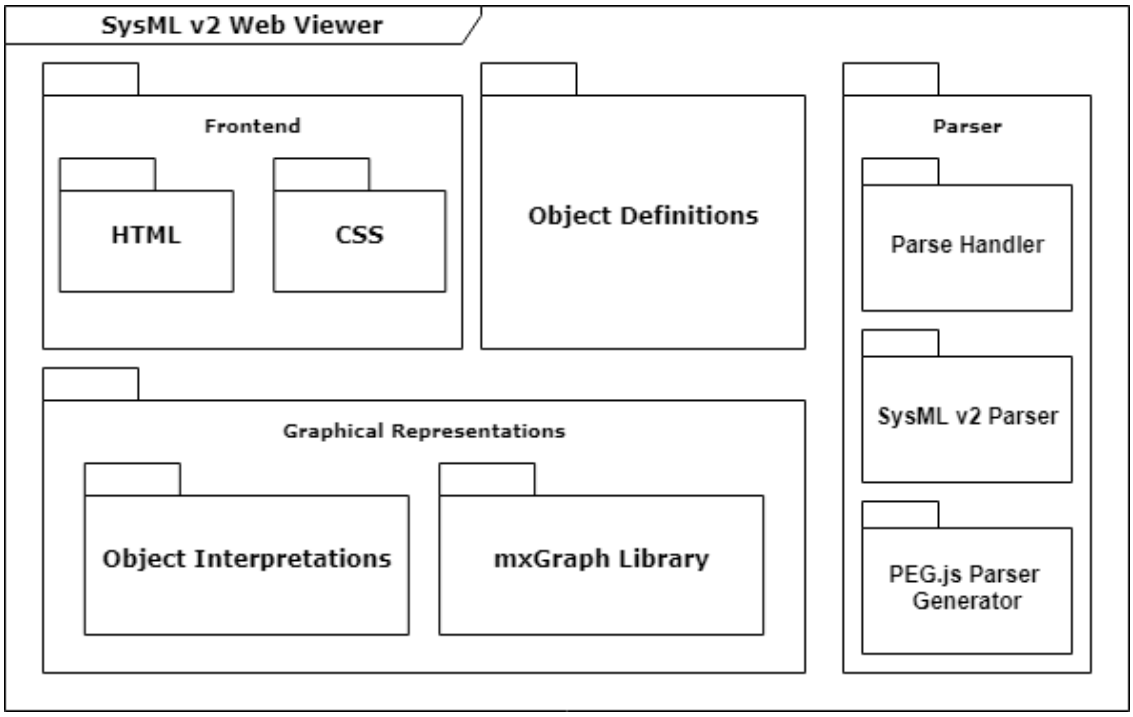

Figure 1: Package diagram describing the basic structure of the implementation

The package diagram in figure 6 shows the overall structure of the application. It consists of a front-end module, an object definitions module, a parser module and a graphical representations module. The front-end module is used to display a website based application using basic HTML [7] and CSS [8] documents to present a graphical view of a system using SysML v2.

The object definitions module consists of predefined SysML v2 object definitions based on the SysML v2 syntax. These are the objects that the entire application utilize when handling the SysML v2 building blocks and diagrams. The object definitions together with the parser module is then used by the graphical representations module to provide a graphical output of the parsed input.

The parser module in itself consists of the PEG.js parser generator [9], parse handler and the SysML v2 parser. The PEG.js parser generator is used together with a grammar rule-set to generate the SysML v2 parser, which is a functional PEG parser [10]. The parse handler connects the SysML v2 parser with the object definitions so that the parse handler can create SysML v2 objects based on the parsed input.

The graphical representations module consists of the object interpretations used to visualize the actual SysML v2 objects created from the parse handler based on the object definitions. The graphical representations module also consists of the mxGraph library [11] used as the library to display graphics in the application.

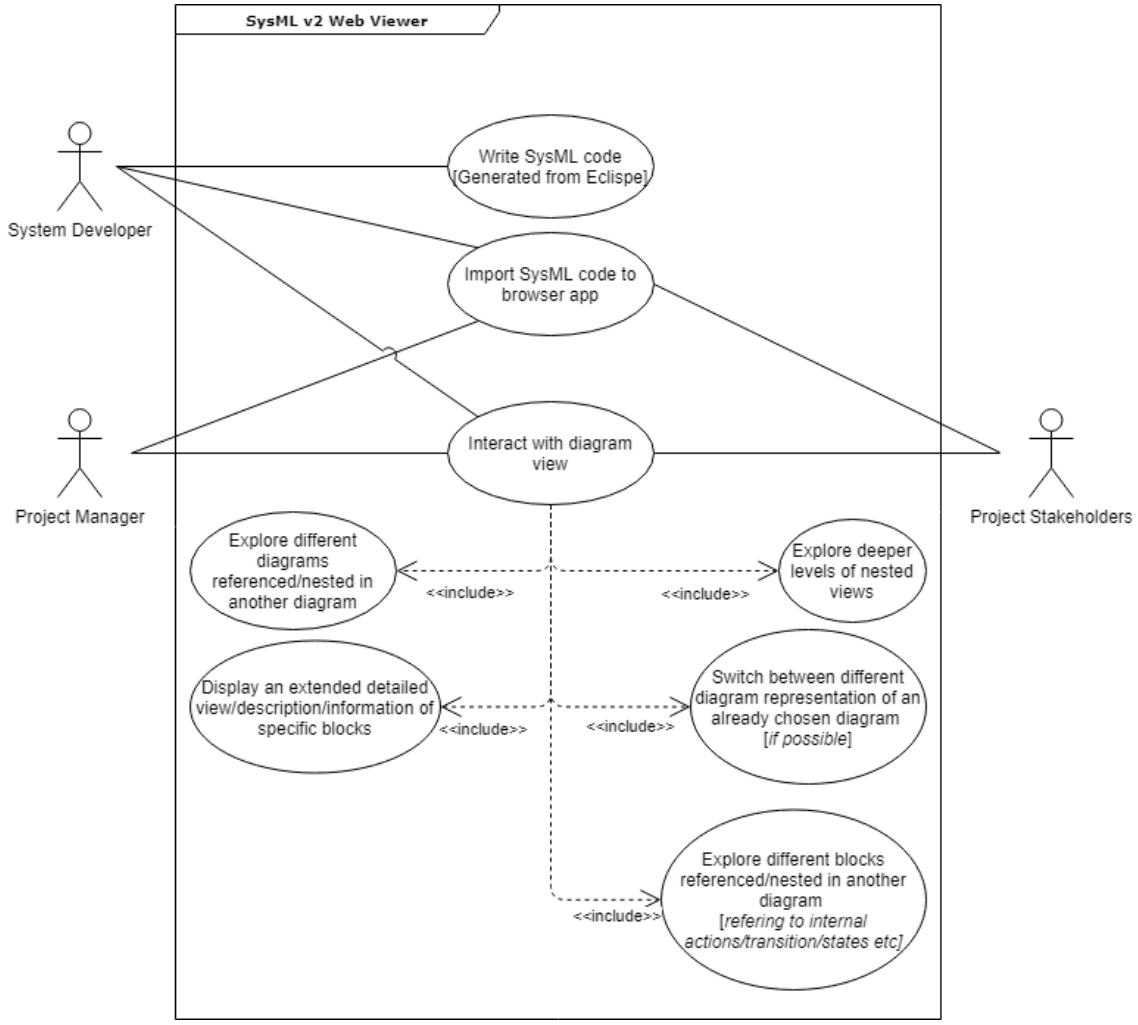

Figure 2: Use-case diagram describing some uses in the the application

The use-case diagram shown in figure 2 describes some of the various uses that the application is planned to have in the future, since these uses unfortunately did not get implemented during this projects time frame. The primary actors for the application are:

- System Developer - The main function of this actor is as a developer and engineer of a particular system.
- Project Manager - This actors main function is as the project lead of the particular system that is being developed. As a project manager this actor oversees the team of system developers and provides the communication channel with the project stakeholders.
- Project Stakeholders - This actors main function is that of an external (or in some cases internal) party which have interests of the particular system being developed. This actor can in a sense be seen as the customer of the particular system being developed.

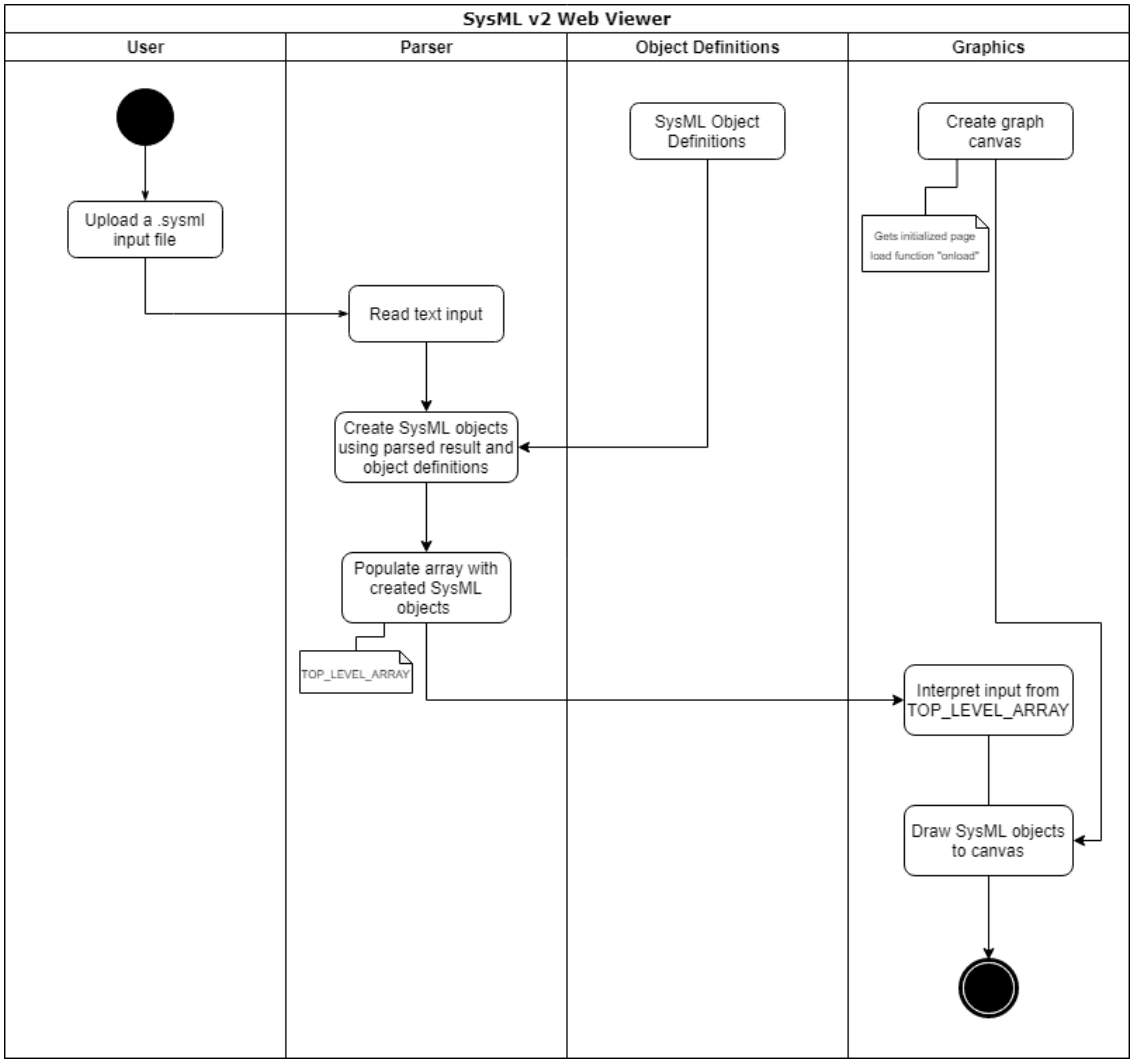

Figure 3: Activity diagram showing the use of the application

The activity diagram in figure 3 shows the general functionality of the application as one of the actors presented earlier uses the application. The process describes the general actions taken in the application from the point of a user uploading a .sysml file. As shown in the figure the basic process of handling an input file is rather simple in that the actions taken by the application does not involve to many steps to produce a complete result.

3 Results

3.1 Delivery

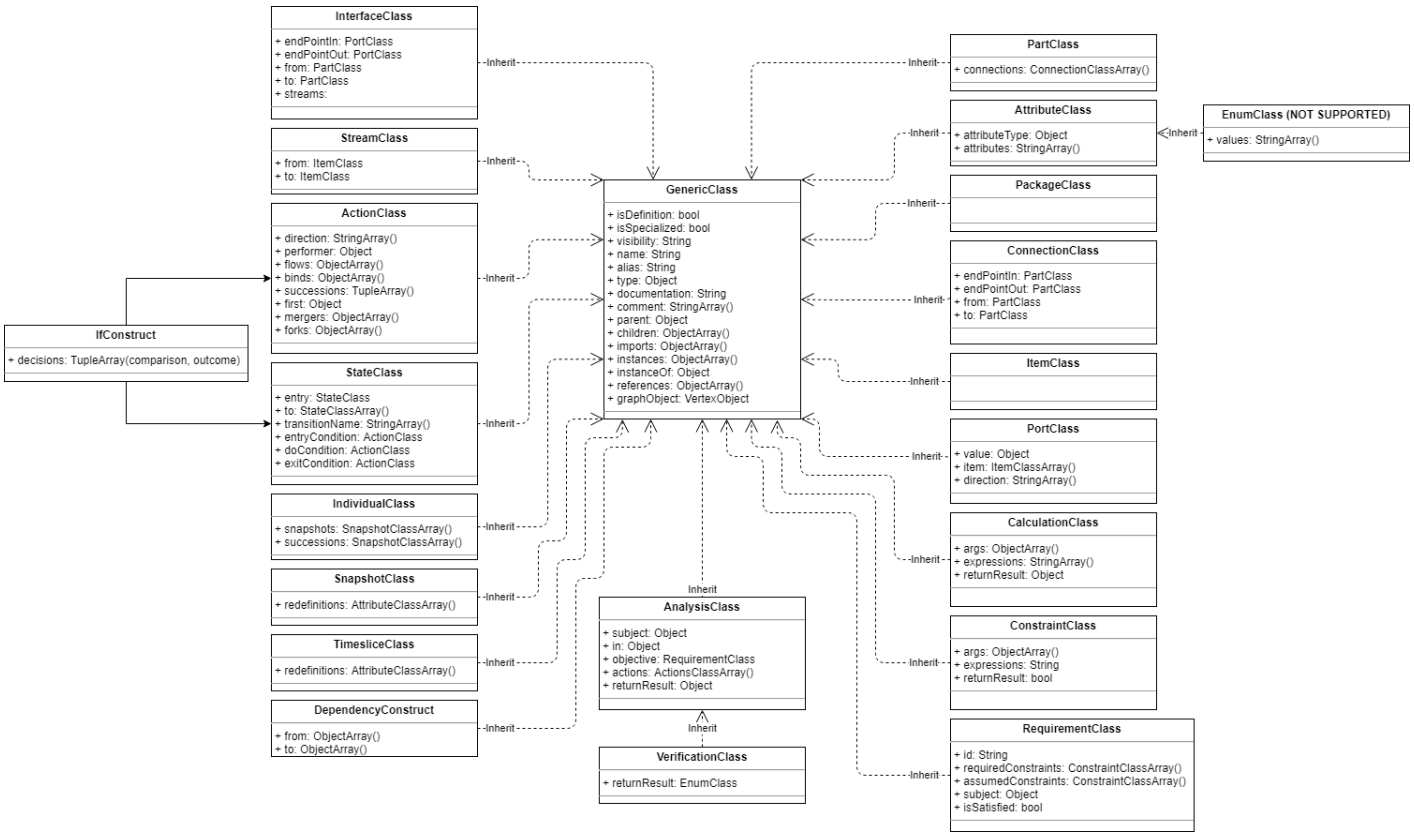

Figure 4: The planned class diagram for the current and future SysML v2 object definitions

The first part of the project was all about planning as far forward as possible. The entire first sprint and part of the second sprint was spent planning for implementation as far as we could. We read through the available SysML v2 documents and tried to fit all of the keywords used in SysML v2 into a class diagram. This diagram, shown in figure 4, contains our understanding of SysML v2 and how we would need classes to look to implement SysML v2 functionality in JavaScript. This was created to try to make our development future proof and make it easy to continue for someone else after this project has ended.

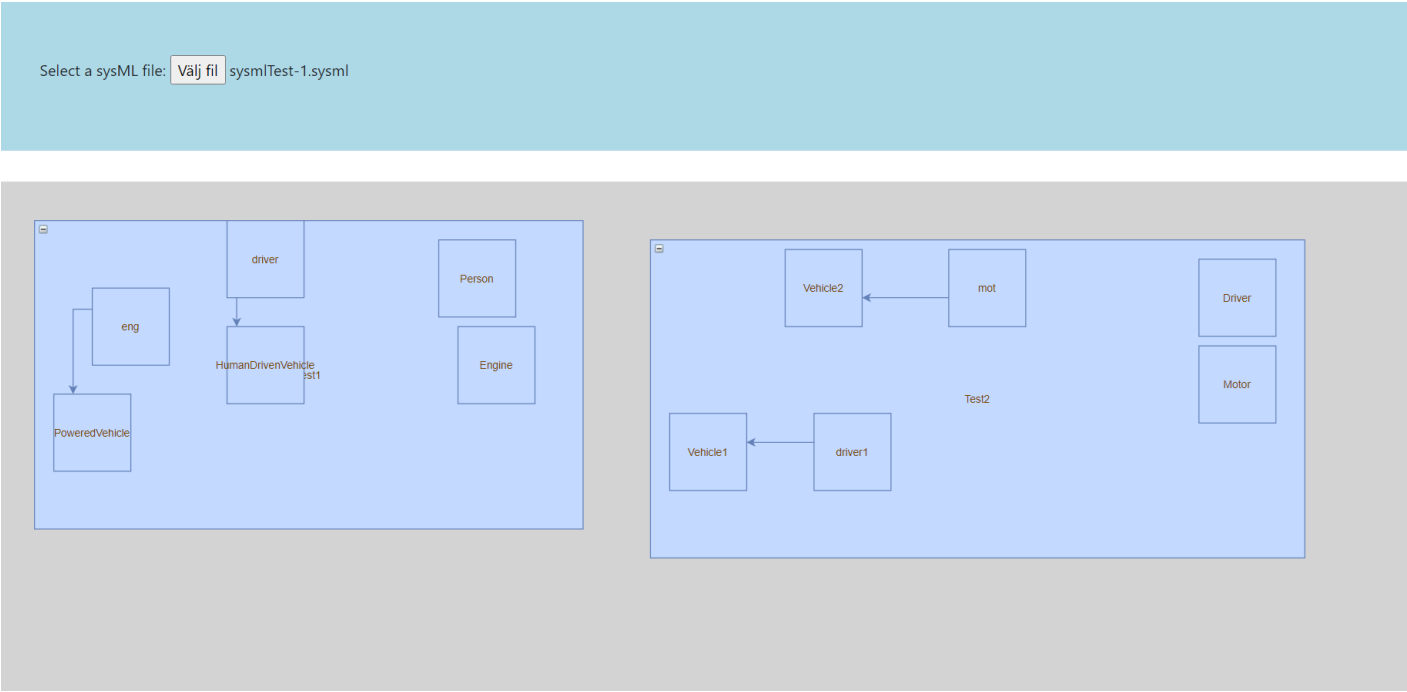

Figure 5: The current version of the application

The end product that we ended up with was a proof of concept rather than a fully fledged interactive SysML v2 web viewer. At its current state it is only possible to upload a .sysml file containing packages, part definitions and part usages and display them as crude diagrams in a restricted manner. The layout algorithm and placing of the different SysML objects is not implemented yet. There are also no official SysML diagrams that can be shown yet. However there are things that have been implemented that are very important and useful for future work. A parser generator, PEG.js mentioned earlier in section 2, has been used to create a functional PEG parser which provides the basic groundwork regarding parsing for future development on this project and testing of the generator has shown that it is suitable for the task. A generalized foundation of object definitions used to create the SysML objects that can be used for graphical representation has also been created.

There are a few things that have not been incorporated into the parser at the moment. Specifically the things in the parser that are not yet fully realized and mature are the missing rule-sets for SysML objects other than packages, part definitions and part usages. There are several different functions, rules and more that has to be added for a functioning SysML parser. There is one issue that has been purposely ignored due to an assumption made on our part. What we chose to ignore was checking for errors in the code and assuming that it is correctly written. We can assume this because we expect the .sysml files to be written in the Eclipse IDE [12] where there is a syntax checker already included for the SysML v2 language.

The graphical representation was created using a library called mxGraph [11]. This part of the product is not as developed as most other parts, there were many functions that we wanted to implement but didn't have time for. What we did implement is the very basic ability to draw objects based on information from the parser. It can draw nestled objects within each other but it can only draw the same looking rectangles for all objects, meaning that parts and packages look the same which they shouldn't. Parts within parts are represented using arrows while parts within packages are nestled. When a file is uploaded all the objects are just drawn in the upper left corner stacked on each other instead of placed using a layout algorithm like we planned. While it is possible to move objects and arrows, it doesn't look good. Also it was supposed to be interactive which it isn't at this point.

Most of the code and work put into the project has been in the effort to create a solid starting point for future attempts at this web viewer. That has been the goal of this project the entire time. In the end, the goal of this project was not a fully functioning SysML v2 web viewer. As such we did manage to succeed in delivering what was asked of us to deliver; a solid foundation to continue developing on.

3.2 Testing

3.2.1 Risks of regression

Based off of an interview with Tommy Andersson who previously worked on this project, our design is aiming to be as modular as possible with very distinct and separated classes and methods to offer as much scalability as possible. This, of course, raises the risk for remote regression as a small change in a method may ripple across the entire program. For example we might need to change a method to make it more efficient or if we discover a bug and this seemingly small change could cause much bigger problems since other parts of the program depends on this method.

Due to the fact that the planned implementation of the diagrams involves implementing the diagram one at a time, a big risk of both unmasked and remote regression appears. For example, modifying the parsing code to include new definitions of SysML v2 functions may unmask bugs that was previously harmless and in turn, the graphical representation might break when implementing these new diagram definitions as well. Creating new definitions can create remote regression in both the graphical representation and the parser as well as creating local regression risks and as such extra care will have to be put into handling new definitions.

Every time we add something new there is of course also the risk of local regression, and when modifying existing classes and methods to account for new diagrams the risk of local regression also increases. The risk of unmasked regression also increases in these situations.

Our design choices open up the possibility of many regression risks, however the trade off for us seems worth it due to the benefits of scalability and the ability to salvage code in the case of failure. Regression risks can be worked around through proper testing and vetting but scalability issues are not so easily patched at a later date.

3.2.2 Strategy for regression testing

Traceability From our use case diagram and requirement specifications we can decide what to test using traceability. For example the use case diagram describes that a user should be able to import SysML code to browser app and then interact with the diagrams. This could be tested to make sure that the functions works as intended and similarly this should be done for every use-case described. One approach to make sure that the tests are traceable is to use branches in the development of different parts of the application. This provides a way to easily trace tests to specific branches of code containing specific functionality.

Change analysis To ensure that no errors manages to sneak it is way past development, a close eye needs to be kept on any rippling effects after changes. What we mean by this is that a highly modular program becomes very hard to keep track of when the program scales to larger sizes and as such, remote regression becomes very hard to notice and patch. When we want to add or change a part we need to triple check all the parts that inherits what we changed to make sure that behaviour has not changed. We need to trace the changes all the way through all the other modules that it can affect to make sure our program works as intended.

Quality risk analysis The biggest factor in the success of this project is whether or not we can interpret the SysML v2 language and documentation correctly. A failure to understand the SysML v2 language and documentation would lead to a poor implementation of the application which would lead to an application that is unable to be built upon in the future. Therefor a strategy is to make up a solid plan of how the solution is supposed to be implemented and make it possible for others to build upon this plan in the future.

Cross functional testing Using cross functional testing we can find bugs in unexpected places. This is done by testing functions that should have no effect on each other both together and in isolation. If the results are different, there is a bug. At first we could do a wide search to cover large areas of the system in order to find common areas with bugs. Then when we know these problem areas, we can focus the testing on them to uncover more bugs.

3.2.3 Development description

The following table describes our plan for automated unit and system level testing for regression. Our plan contains two stories, one for unit level and one for system level, on how to create automated testing.

Story ID	Story	Task ID	Task *[Risk from 1-10 (low to high)]*	Time estimate (story)	Time estimate (task)	Dependencies	Risk	Priority
1	**Title:** Test of .sysml file upload *(Automated unit testing)* **Intended use:** Automated test for uploading .sysml files to the web application which are generated from Eclipse **Desired properties:** Will test the upload functionality **Test case:** Tests with proper SysML v2 syntax, inproper SysML v2 syntax, with .sysml file type, with other incorrect filetypes	1.1	**To do:** Make a script to check if uploading several files converts correctly to a directory structure **Risk:** 8	19 h	6 h	N/A	8	High
		1.2	**To do:** Make a script to check if correct filetype provides correct output **Risk:** 2		2 h	N/A		
		1.3	**To do:** Make a script to check if uncorrupt/correct syntax provides correct output **Risk:** 5		4 h	1.2		
		1.4	**To do:** Make a script to check if incorrect filetype provides correct error message **Risk:** 2		2 h	1.2		
		1.5	**To do:** Make a script to check if corrupt/incorrect syntax provides correct error message **Risk:** 7		5 h	1.2		
2	**Title:** Test of graphical rendering *(Automated system testing)* **Intended use:** Automated tests rendering of graphics of the SysML v2 digarams **Desired properties:** Will test all diagram types **Test case:** Tests with graphical rendering of the different diagrams, relationships/references links etc,	2.1	**To do:** Make a script to check if file reader interprets the file correctly **Risk:** 5	22 h	6 h	N/A	8	Medium
		2.2	**To do:** Make a script to check if syntax reader interprets the syntax correctly **Risk:** 5		6 h	2.1		
		2.3	**To do:** Make a script to check if parser handles the file input and object definitions correctly **Risk:** 7		4 h	2.1, 2.2		
		2.4	**To do:** Make a script to check if rendering of different objects behaves correctly **Risk:** 8		3 h	2.1, 2.2, 2.3		
		2.5	**To do:** Make a script to check if rendering of references/relationships/links etc behaves correctly **Risk:** 8		3 h	2.1, 2.2, 2.3, 2.4		

Figure 6: Table describing automated test stories

3.2.4 Test strategy

Our general strategy for testing has been to try every new function out locally and separate during development. Then try it out with other parts when merging to see if they are compatible. Due to our extensive planning this has worked quite well as everyone had a good understanding of the whole picture and how the parts were supposed to fit together. Also since we coded in pairs there was always someone to discuss with and who could help look for errors and bugs.

Our strategy for traceability was implemented right before we started to code by creating branches in our Github, which can be found in the appendix. The code wasn't pushed to main branch until the very end when we had a finished product. During development our group split into pairs which each had a branch. The pairs worked on different parts and when ready the different parts where merged into a development branch. This worked really well as it made it easy to see progress and to solve any problems that arose.

Regarding the quality risk analysis our solution was to create a class diagram for object definitions by discussing our understanding of SysML v2 in group. By doing this we have made sure that others can understand how we interpreted SysML v2 and how we implemented it. The code has been kept as modular as possible for the same reason.

Since our product is quite small in terms of lines of code, we didn't have to worry about rippling effects created by a modular design as we feared in our change analysis. It might be a bigger problem in the future when more functionality is added but for this project it didn't pose a problem.

With the help of our branches we could test functions and parts of the code either separately or together to find bugs in unexpected places. This was described in the cross functional testing description and while we didn't experience many bugs due to our relatively small product it was still a well functioning strategy.

4 Ethical assessment

One of the ethical concerns of the application is based on how to handle potential data that might be confidential when uploading .sysml files to present a particular system. The concern that might exist and the potential problem that arise is mainly the leak of this confidential data pertaining to the systems presented in the .sysml files and letting that data get into the hands of bad actors. However at the current state that the application is in, this should not be too much of a concern application wise. This is due to the local nature of the application since it is only located on the users own local computer and not hosted on a server. Another fact that voids this particular concern is the fact that the application does not save the parsed .sysml file when the application is closed. If the application however was to be hosted on a server and/or the .sysml files were to be saved on every new file upload some other precautions should be taken. To be able to counteract potential data leaks or data generally getting in the hands of bad actors by accident is to include more security measures into the application. This can be done on the server hosting side with encrypting communication with the server and on the application side to encrypt the .sysml files when they are uploaded.

There are also things that can be done to counteract future attacks from known sources. If this product goes public, then bad actors could possibly use it to create systems that are made to harm others. This is of course an exaggeration of the ethical problems with the application but it needs to be addressed either way.

Another possible ethical concern that is a potential problem in the future is the fact that the application could, if implemented poorly, make the displayed systems be misunderstood. This could mean that projects that need to showcase a particular system, to perhaps get funding or to get approved, are not able to be shown in a fair and positive light and therefore never get approved to continue. This problem could also extend to an even more serious concern were projects that develops a complex system could be made incorrectly and cause problems or failures down the line. For example if the application was to parse the SysML v2 syntax incorrectly and the resulting diagrams are used for displaying the system of a space rocket it could mean that the diagrams are not showing certain parts correctly or certain object connections could be wrong etc. In this case there is a possibility to lead to devastating effects for everyone involved.

A positive side from an ethical point of view is however the complete opposite of the problems that have already been brought up, provided that the applications future development continues without any of the mentioned setbacks. These are that this application will help make it more clear and easier to use SysML v2 diagrams that will lead to better developed systems for projects. It has the potential to avoid unnecessary miscommunication due to the fact that the application is based on .sysml input that can be sent around with the diagram descriptions and not diagrams drawn by hand. This avoids simple mistakes and creates a better, safer and easier to understand system. Finally the most important thing this application has the potential to achieve, is to provide less tech-literate people with the ability to understand the systems that are being presented. This will make way for more projects and progress for entire industries. It is according to our opinion that this will do more good than the possible bad in an ethical point of view.

5 Conclusion and future work

As stated earlier the goal of this project wasn't a finished product, but rather to lay the groundwork for future attempts, and ensure that whoever takes up this project after us has a real chance to continue our work and actually finish it. Therefore there is a lot of work left to do before the product is finished according to specifications. First of all since SysML v2 is a newly released language still under development, modifications and additions to our plan will probably be necessary. For example our object definitions as described in a class diagram, shown in figure 4, will need to be updated if their specifications are changed.

In the graphical representation one of the first steps for improvement would be to add a layout algorithm so that all objects are placed visibly and correct in relation to each other. Since the product is supposed to show different diagrams, the ability to place objects like these diagrams needs to be added as well. It would also be preferable if different objects had distinct designs, for example if packages and parts looked different from each other. Another valuable requirement for this project is interactivity. As such, focus needs to be put on this area to make sure the requirements are met. For example this could mean: adding zooming to see different levels or views; being able to double click on a component for further information or to see connections between components in different parts of a system. Also as more object definitions are added, parsing rules will need to be added as well.

As we knew that changes would be necessary in the future, we have tried to make the program as modular as possible and created an easy to read documentation of our work. This way it will be easier to pick up right where we left off and to add new functionality. Even though our product doesn't hold much practical value in its current state, it has a lot of potential if developed upon further. Then it could prove a valuable tool for system developers in a wide range of fields.

References

[1] Jesper Nilsson. "Interactive SysML Diagrams using a Web Browser". 2020, pp. 1–53.

[2] Tommy Andersson. "A software to display Sysml graphically". Oct. 2020, pp. 1–16.

[3] Object Management Group. *UML*. URL: https://www.uml.org/. (2020-12-17).

[4] Object Management Group. *SysML*. URL: http://www.omgsysml.org/. (2020-12-17).

[5] Object Management Group. *MBSE*. URL: https://www.omgwiki.org/MBSE/doku.php. (2020-12-17).

[6] The Eclipse Foundation. *Eclipse Papyrus*. URL: https://www.eclipse.org/papyrus/. (2020-12-17).

[7] MDN contributors. *HTML: HyperText Markup Language | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/HTML. (2021-03-17).

[8] MDN contributors. *CSS: Cascading Style Sheets | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS. (2021-03-17).

[9] Futago-za Ryuu, David Majda. *PEG.js – Parser Generator for JavaScript*. URL: https://pegjs.org/. (2021-03-15).

[10] Wikipedia, The Free Encyclopedia. *Parsing expression grammar — Wikipedia*. URL: https://en.wikipedia.org/wiki/Parsing_expression_grammar. (2021-03-15).

[11] JGraph Ltd. *mxGraph 4.2.2*. URL: https://jgraph.github.io/mxgraph/. (2021-03-15).

[12] Eclipse Foundation. *Eclipse desktop web IDEs | The Eclipse Foundation*. URL: https://www.eclipse.org/ide/. (2021-03-17).

Appendices

Link to project GitHub repository: https://github.com/olihgb-7/D0020E
Link to project documentation: https://github.com/olihgb-7/D0020E/blob/main/README.md