UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Joonatan Ehlvest

Adding annotation functionality to a web-based viewer for Fractal Enterprise Models

Bachelor's Thesis (9 ECTS)

Supervisor(s): Dr. Ilia Bider

Tartu 2025

Adding annotation functionality to a web-based viewer for Fractal Enterprise Models

Abstract:

The FEM Viewer is a web application for viewing Fractal Enterprise Model diagrams exported from a FEM Toolkit. This thesis aimed to develop the FEM Viewer further by adding the ability to make certain changes to the model for the purposes of annotating the FEM diagrams. It describes the new functionality and how it was added, including how the way the models are displayed was reworked to enable these changes.

Keywords: FEM, SVG, annotation, Fractal Enterprise Model

CERCS: T120 Systems engineering, computer technology

Annoteerimis võimekuse lisamine veebipõhisele fraktaal ettevõtte mudelite kuvajale

Lühikokkuvõte:

FEM *viewer* on veebirakendus FEM *toolkit'is* loodud fraktaal ettevõtte mudeli diagrammide kuvamiseks. Selle lõputöö eesmärk oli FEM *viewer*'it edasi arendada, lisades võimaluse teha mudelites teatud muudatusi FEM diagrammide annoteerimiseks. Antud töö kirjeldab rakenduse uut võimekust ja seda, kuidas see lisati, sealhulgas kuidas mudelite kuvamine nende muudatuste võimaldamiseks ümber töötati.

Võtmesõnad: FEM, SVG, annoteerimine, fraktaal ettevõtte mudel

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia

Table of Contents

1. Introduction

The Fractal Enterprise Model (FEM) is an enterprise modeling solution proposed by Bider et al. [1] for identifying and documenting business processes in an enterprise. To provide a consistent way of creating diagrams that conform to the FEM metamodel, an application called FEM Toolkit [2] was created. This tool was designed to be used primarily by people creating the models, and it was not convenient to use for most business people. In addition, the tool had to be run locally and was inconvenient to install. To address these problems, Siim Langel developed the FEM Viewer web application as a part of their Bachelor's thesis [3], [4]. The FEM Viewer made it simple for users with little technical experience to view and share the models without having to install anything, but did not provide any functionality for editing the models. This means that there is no built-in way for the stakeholders viewing the models to provide feedback to the model developers using the FEM Toolkit.

This thesis aims to make it possible to provide feedback by annotating the models. This will be done by adding functionality for editing descriptions of model elements as well as highlighting them by changing their background and border color, which can be done by assigning elements subclasses defined by the model creators using the FEM Toolkit. It should then be possible to export the edited model from the FEM Viewer and import it back to the FEM Toolkit, where the model creators can make additional changes based on the feedback.

In chapter 2 this thesis gives a simple overview of Fractal Enterprise Models as well as the FEM Toolkit and the existing FEM Viewer application. Chapter 3 describes the primary technologies used to add the new functionality. How the functionality for making changes to the model was implemented is described in chapter 4, while chapter 5 describes how and why the way the models are displayed in the application was reworked. Finally, chapter 6 covers the more minor improvements made to the application.

2. Previous work

2.1 Overview of FEM and FEM Toolkit

The initial version of the FEM consisted of two types of nodes: business processes and assets, as well as relationships (connections) between them [1]. Later, two new types of nodes: external pools and external actors, were added to represent things outside the enterprise that have an influence on the enterprise's processes [5]. Together, the nodes and the relationships between them make up a directed graph (see Figure 1).

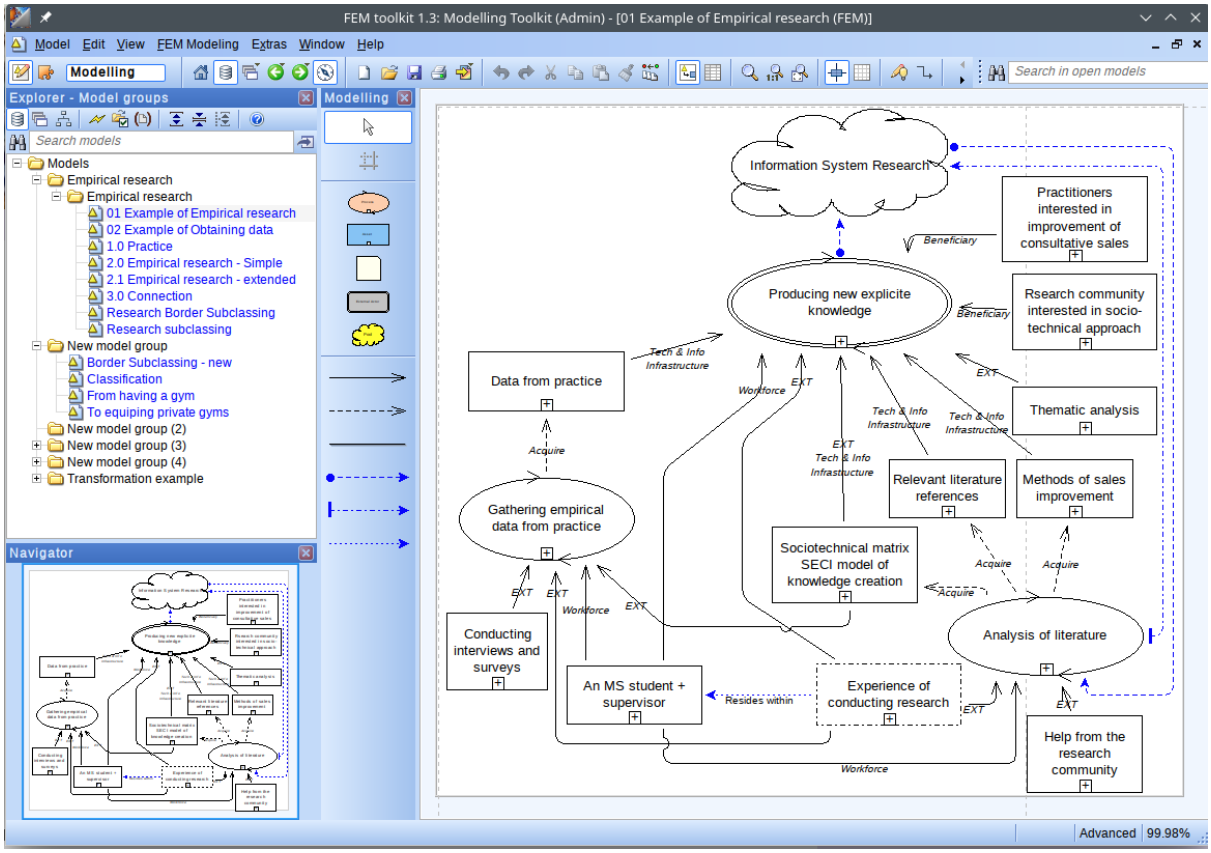

Figure 1: Example of a FEM diagram in FEM Toolkit

The FEM Toolkit [2] is built upon the ADOxx metamodeling platform and allows for the creation of FEMs while preventing the creation of elements that should not exist in a FEM. This limits, for example, which types of connections can be drawn between the model nodes and which labels can be applied to them. There are also additional attributes that the nodes can have that are reflected in how they are displayed, such as a process being a primary process or an

asset being tacit. The FEM Toolkit also provides additional features that make creating and navigating a FEM simpler. These include being able to:

- Add notes to the diagram and connect them to FEM model nodes
- Create copies of model elements (ghosts) which can be used to represent the same element in different models, and allow for easier navigation by being able to navigate to the original element from a ghost element or vice versa
- Assign subclasses with different background colors to highlight differences between nodes of the same class

In FEM Toolkit, multiple models can be grouped and elements can be referenced between them. This is used to implement subclasses and border subclasses. The subclass elements are created in a separate model that only allows for either subclass or border subclass elements. These elements can then be referenced by nodes in other models in the same model group.

The main weakness of FEM Toolkit is that it is too difficult to use for users who just want to view the models, which is why the FEM Viewer [3] was created.

2.2 Overview of the existing FEM Viewer

The FEM Viewer [3] was developed in JavaScript using React.js for the frontend and the Node.js platform with Express.js for server-side functionality. In addition, a MySQL database was used for persisting user data along with Prisma.js for abstracting the database and making writing queries easier. The Passport.js middleware was used for authentication and handling user sessions. These technologies were chosen because they were high-level and widely used, making it easier to find people who would be able and willing to develop the project further. Since this core tech stack still satisfies all its requirements for effectively building the application, it was not changed during this thesis.

To do anything in the FEM viewer, the user needs to authenticate themselves by logging in. Users can have one of three roles: Admin, Developer, or Viewer. The users with the Admin or Developer role can create additional users that they can manage (the initial user or users have to be manually created on the server). Once logged in, the user is taken to the dashboard from where they can access the various functions of the app depending on their role:

- In the register view, admin users can create new users with any role, and users with the developer role can create new users with the viewer role. The user can see users created by them in the dashboard view, where they can also delete them (this will recursively delete all users created by the deleted user).
- In the upload view, users with the admin and developer roles can upload model groups and share them with users they have created. The uploaded model groups consist of an XML file and SVG files for each model in the model group. All of these can be exported from the FEM Toolkit.
- All users can view model groups shared with them or uploaded by them. The list of these model groups is displayed on the dashboard. When viewing a specific model group, the user can navigate between the models by using the menu on the left side of the screen.
- All users can see the model shared with them or uploaded by them on the dashboard.
- All users can change their passwords.

The main use of the FEM Viewer lies in the model view. This view uses the uploaded SVG files to display the model diagram and uses data from the uploaded model XML file to provide an interactive layer on top of the diagram. Clicking on a model node highlights it and opens a pop-up that displays additional information about the element. This pop-up also allows for navigation to referenced elements, for example, to ghost elements created as copies from the selected element or referenced subclass elements. As can be seen in Figure 2, the selected element is highlighted in blue, and the elements that are selected from the navigation options in the popup are highlighted in yellow.

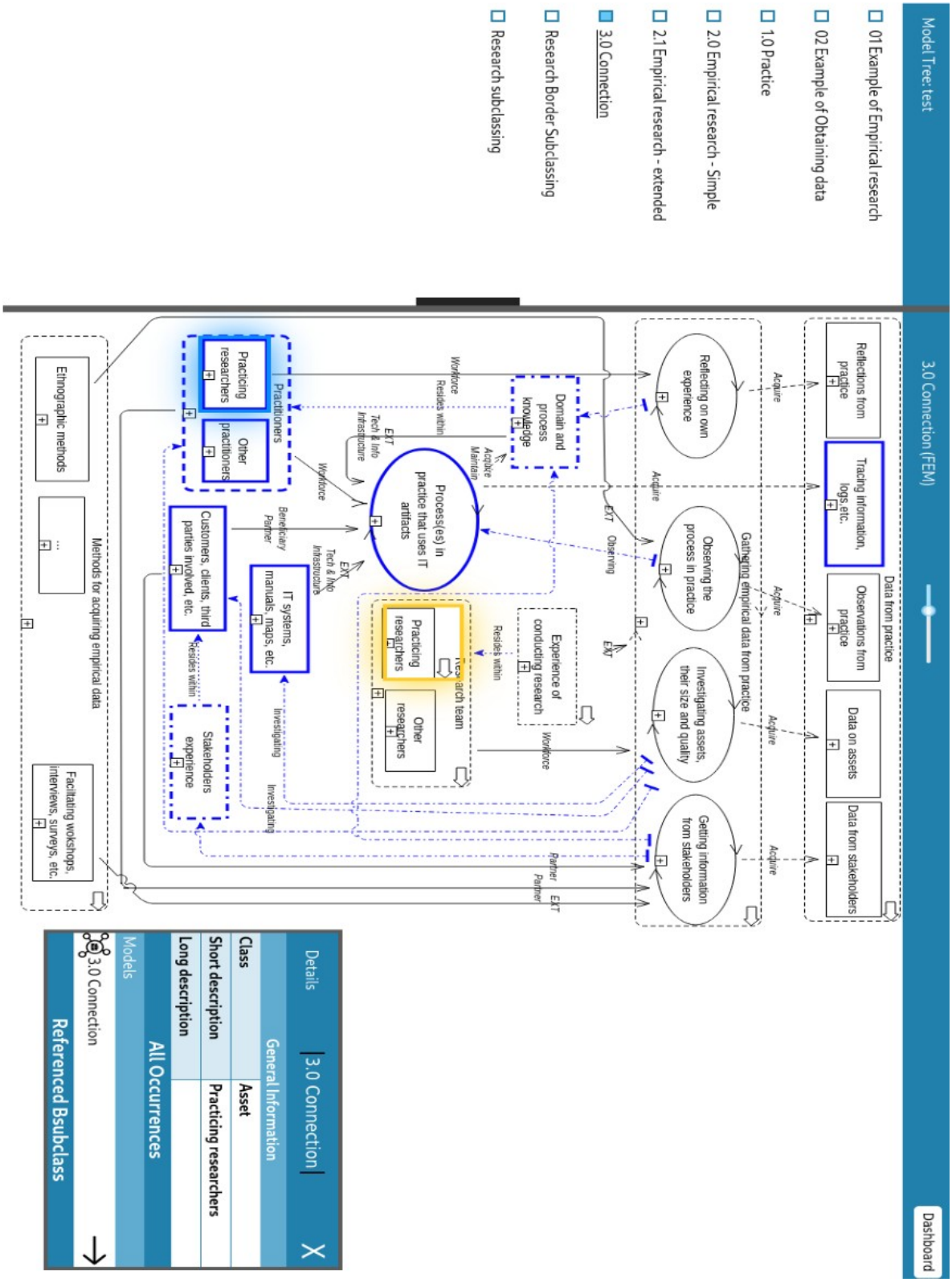

Figure 2: Model view in FEM Viewer

3. Used technologies

An overview of the core technologies used in the FEM Viewer application are given in [3]. This chapter will describe the technologies most relevant to the changes made in the application.

3.1 XML

Extensible Markup Language (XML) is a versatile standard for defining structured text documents. Its structure is similar to HTML, but instead of using predefined tags such as <p> or <header>, XML tags for elements can be defined based on the needs of the users, as long as their syntax conforms to the XML standard. This is what makes XML *Extensible* and useful for storing all kinds of structured data. [6]

3.2 SVG

Scalable Vector Graphics (SVG) is a document format that uses geometric shapes to describe images. SVG is an extension of XML, which means it can work well with other XML data. An SVG document consists of a root <svg> element with a defined width and height, which can contain other SVG elements that are positioned using x- and y-coordinates relative to the upper left corner of the <svg> element. The most versatile element that can be used to draw all other SVG shapes is <path>. The path element takes a set of instructions as arguments that describe how to draw a sequence of lines and curves, which can be used to draw various shapes. In addition, SVGs can also display text inside <text> elements. [7]

3.3 Generative AI

The help of generative AI, mainly Claude 3.5 Sonnet and Claude 3.7 Sonnet [8], was used for writing and editing the application code. It was most helpful for finding relevant parts of the code that needed to be changed and for implementing the rendering logic for the SVG elements. Since the author had little experience with JavaScript and TypeScript and no prior experience with React, these tools were also helpful for explaining the existing code and language-specific syntax. One of the consistent weaknesses of these models was that they tended to be verbose and tried to do more than was prompted. Sometimes the proposed solutions would handle relevant edge cases, but often they would be needlessly complicated and would need to be simplified.

4. Editing models

This thesis had the goal of adding two primary improvements to the FEM Viewer: being able to edit model node descriptions and being able to assign subclasses and border subclasses to model nodes. To achieve this, two main things needed to be implemented: the functionality for editing the models XML file in a way that the XML file could be imported back to FEM toolkit while reflecting the changes made to it and the ability to display the changes made in the FEM Viewer.

4.1 Overview of the XML files

The models created in the FEM toolkit can be exported and imported as XML files. This means that it is possible to make changes to the models by editing the XML files outside the FEM Toolkit, as long as the changes made follow the same logic used by the Toolkit. The existing FEM Viewer already stored the model data as an XML file, but it did not have any functionality for making changes to it. Therefore, to enable updating model node descriptions and subclasses, it was necessary to identify which XML attributes needed to be changed and add the ability to change these attributes without making any unintended changes to the file.

Finding the right attributes to change was done primarily by looking at examples of XML files generated by the FEM Toolkit. The general structure of a FEM XML file is as follows: a single MODELS element contains any number of MODEL elements, which further contain a MODELATTRIBUTES element as well as any number of INSTANCE and CONNECTOR elements. The MODELATTRIBUTES contain the general configuration of a FEM diagram. For this thesis, the important attributes here are the World Area, which represents the size of the diagram's canvas, as well as the attributes defining the default colors for different kinds of nodes.

The most significant elements for us are instances and connectors, with instances representing the nodes of the model and connectors representing the relationships between them. All instances contain a position attribute, which defines its coordinates on the canvas as well as its width, height, and stack order. There are also attributes that determine how the instance should look, such as font size and denomination, which define the size and content of the text displayed for most types of instances. In addition to the attributes that are common among all instances,

there can also be additional attributes for different classes of instances that can influence how it is displayed, such as Process instances with the isprimary attribute having a double border. Most importantly for this thesis, the instances also contain a field for referencing subclasses and border subclasses as well as attributes that define their individual and referenced color and border color, along with which one of those should be used when displaying the Instance.

The connector elements also contain position data and attributes that help define the appearance of the connection arrow and its label. The position data includes the connectors' stack order, optional coordinate points that define the connection path, and a "middle" point that determines where the label should be placed (unless specified, this is in the middle of the connection). The connector also includes references to the instances it is drawn between, which help determine its start and end points.

4.2 Implementing the XML editor

While it would be possible to edit the XML file directly, this would require complicated logic for finding the right attributes and be prone to errors. To make editing the XML easier, the fast-xml-parser [9] library was used to parse the XML into a JavaScript object (and later back to XML). This library was already used in the original FEM Viewer for creating the model objects that power the applications core functionality, but since the existing implementation simplified the data when parsing it, it was impossible to recreate the original XML without making major changes to how the parsing worked. For that reason, the XML editing logic was built separately from the existing parser.

The implemented editor parses the XML data and preserves the original XML structure. It then provides functions for finding model instances by their IDs and changing the attribute values of these instances. In the simplest case, such as changing the instance's description, the value can be set to a simple string. Otherwise, appropriate formatting is applied to the attribute value depending on its type. Once the changes are made, the XML is written to the file. The fast-xml-parser was configured to maintain all the original structure and data of the XML file. The one thing that it was not able to preserve was the DOCTYPE metadata. In order not to lose this data when making edits, it was parsed separately using regex and manually inserted back in when building the XML.

4.3 Update Service

Following patterns established in the original FEM Viewer [3], the service layer was used to handle the logic of making changes to the XML using the XML editor based on the request received from a HTTP PATCH route. Since all the update operations contain similar logic, an abstract BaseInstanceUpdateService class was created to handle the common operations needed to make changes to an instance. The logic for updating instance subclasses and instance border subclasses was further abstracted into the BaseSubclassUpdateService, since most of the logic, aside from the specific attributes that needed to be updated, was shared between the two. The class hierarchy can be seen in Figure 3.

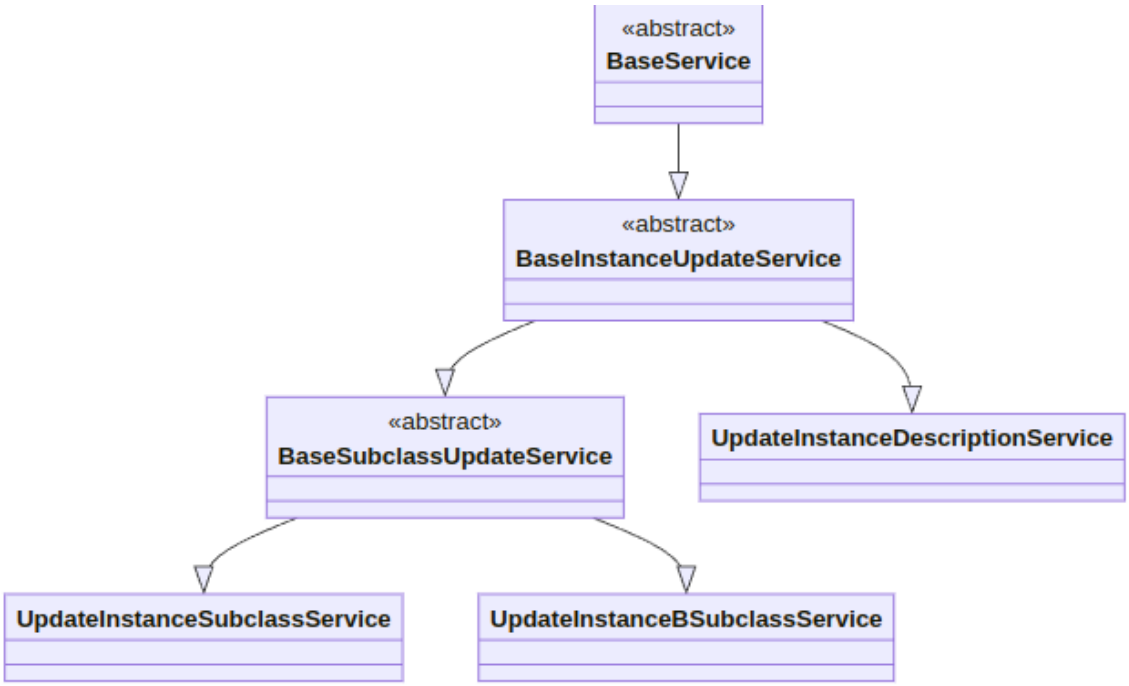

Figure 3: Update service structure

Updating instance descriptions was simple, since only the description attribute needed to be changed. Changing the instance background and border color required changing its color picker attribute, the referenced colors fields for the default and the ghost colors, as well as a reference to the subclass object. The color picker attribute can be set to ‘Default’, ‘Individual’, or ‘Subclass’ for the background color subclass or to ‘Individual’ or ‘Subclass’ for the border color subclass. The subclass update service takes the color picker mode and optionally the referenced subclass ID as attributes and sets the referenced fields appropriately.

5. Changes to the model view

This chapter describes the changes made to the way the models are displayed in FEM Viewer.

5.1 Reasons for the new implementations

The original FEM Viewer used imported SVG files for displaying the models and used the model XML position data to overlay an invisible interactive layer on top of the displayed SVG image. Thanks to this, there was no need to implement logic for displaying different types of instances and connectors, however, this approach posed a problem for adding subclass and border subclass annotation functionality, since it required changing the color and border color of the instances.

There were three approaches considered for solving this. The first was to utilize the interactive layer on top of the displayed model and use it to highlight instances with subclasses. The main problem with this approach was that it would only be possible to add visual elements on top of the original model without changing the appearance of the underlying image. This would make it impossible to visually remove subclasses previously applied to the model in the FEM Toolkit, since even with the highlighting removed, the underlying asset would still have the color or border color of the subclass.

The second solution considered was editing the uploaded SVG files when changes were made to the subclass or border subclass. This would avoid the problems with the first approach and still have the advantage of not having to implement custom SVG rendering, since the only things that would need to be changed in the SVG files would be the fill and stroke colors of the instance elements. The difficulty with this approach would be that the SVG elements do not contain any metadata that would make it simple to link the instances in the SVG file to the instances in the XML they represent. While it should be possible to map the instances to their SVG representations using their position data, this might not end up being significantly easier than rendering the model using only the XML data.

The solution that was implemented was rendering the models from just the XML without using uploaded SVG files. This approach had the added benefit of simplifying the model upload process by not requiring the SVG files and allowing for potential visual improvements to the

model, such as removing a ‘+’ button on the instances that serves a purpose in the FEM Toolkit, but is not actually used in the FEM Viewer.

5.2 New model display implementation

The author of the original FEM Viewer tried displaying the models by just using the XML data, however he ended up using uploaded SVG files due to the latter solution being simpler to implement [3]. Since the SVG rendering implementation was incomplete and undocumented, the SVG rendering had to be implemented mostly from the ground up.

The rendering uses an object-oriented approach, with abstract base classes for instances and connectors implementing the common rendering logic and the inheriting subclasses implementing the logic specific to each instance and connector type. The logic for how to display elements was gathered by observing how the elements were displayed in the FEM Toolkit and by incorporating feedback from the thesis supervisor, who had helped build the FEM Toolkit. Figure 4 shows how a model looks in FEM Toolkit and Figure 5 shows how the same model is displayed in the FEM Viewer with the new implementation that just relies on the XML data.

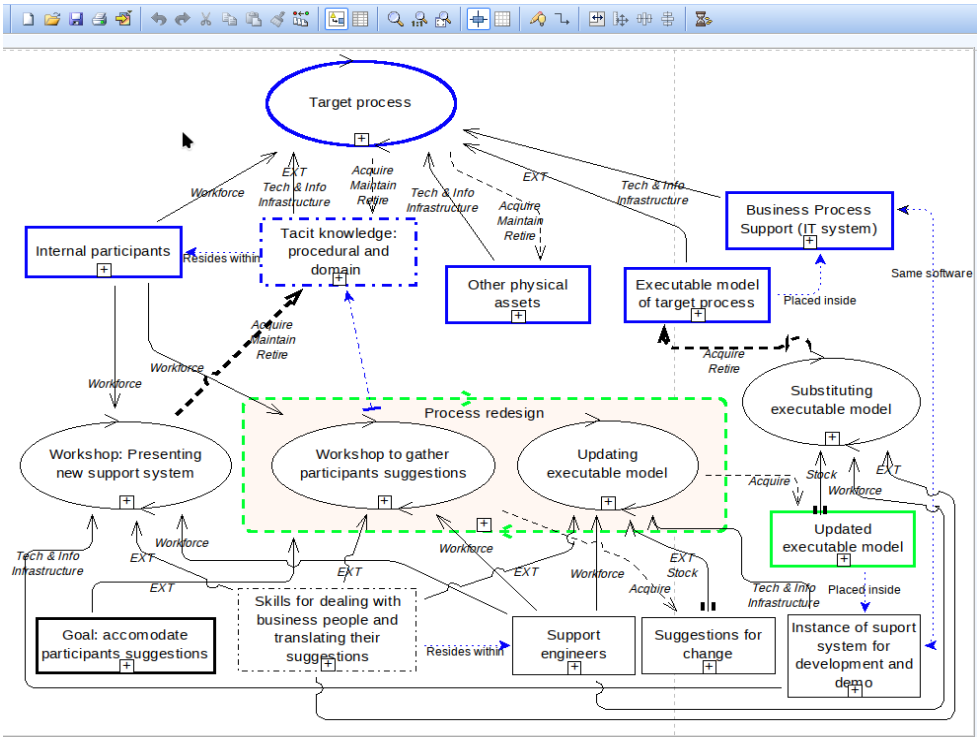

Figure 4: Model in FEM Toolkit

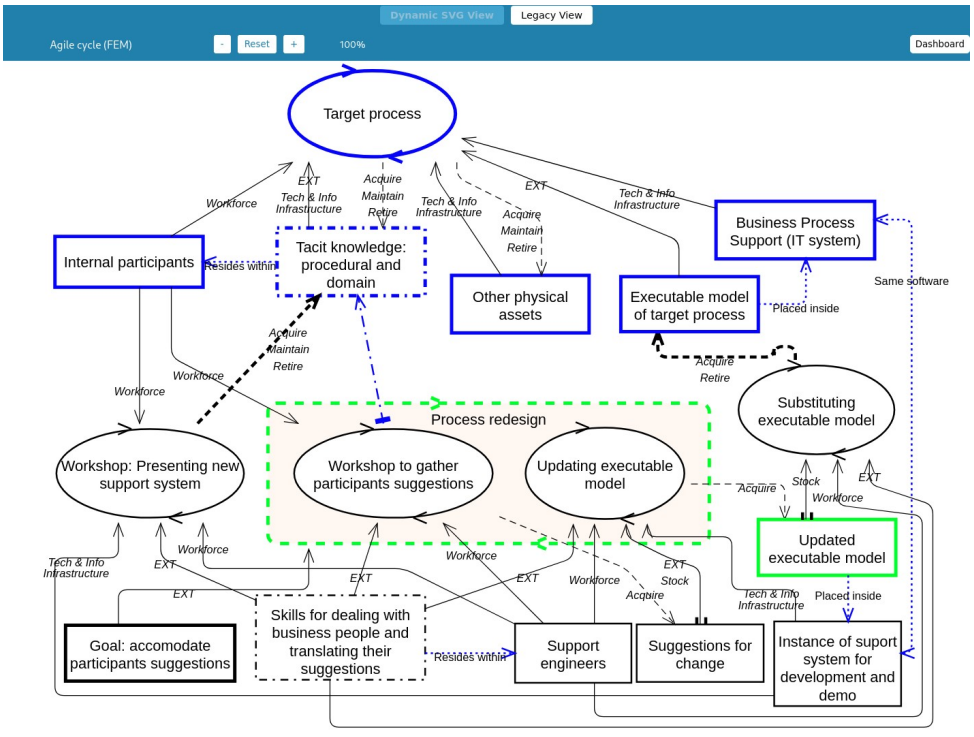

Figure 5: Model in FEM Viewer

5.2.1 Instances

The way an instance should be displayed is determined by the instance's class, subclass, border subclass, whether the instance represents a group of instances or is a copy of another instance (a ghost), and the default or individual colors defined for the instance or the class of instances. The following is an overview of how the different classes of instances are displayed (examples of these can be seen in Figure 6):

- Processes have two arrows at the top and bottom of their border and are, by default, ovals. If a process is the primary process in a model, the instance has a double border.
- Assets have a rectangular shape with sharp corners. Tacit assets have their border made of alternating dots and stripes, as opposed to the default continuous line.
- External actors have a rectangular shape with rounded corners and a double border.
- Pools are shaped like a cloud.
- Notes are rectangles with the top right corner cut off and have a yellow background color by default.

All instances except notes can have two modifiers that change the appearance of the instance in a major way:

- Instances with the group attribute are rectangles with rounded corners and a dashed border. By default, they also have an instance-class-specific background color. The text on the instance is also displayed starting from the top border of the instance instead of the middle, like with non-group instances. Process class instances can additionally be subprocess groups, in which case they will have a dotted border instead of a dashed one.
- Instances that are ghosts have an arrow at the top right corner to indicate that they are copies of another instance.

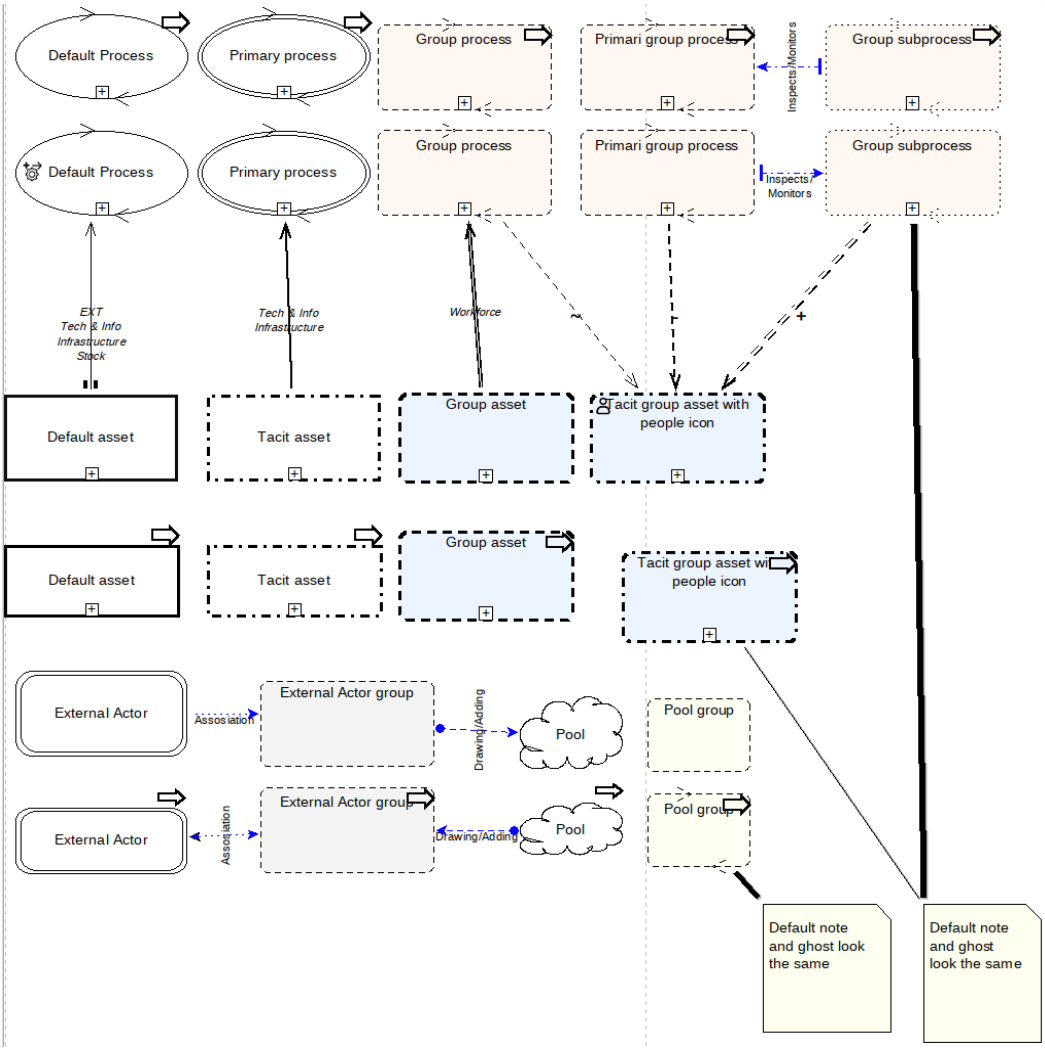

Figure 6: Example of FEM elements in FEM Toolkit

Observing these rules, rendering for the instance elements was implemented using primarily the SVG rectangle, ellipse, line, and path shapes.

In the FEM Toolkit, process and asset class instances can be set to display one of a predefined set of icons in the top left corner of the instance. These are the only elements in the model that use raster image (PNG) files instead of being rendered using SVG. The icon's images are stored in a public icons directory on the server, and which icon to display is determined by the icon name defined for the instance in the XML.

5.2.2 Connectors

The connectors' start and end points are defined by the positions of their start and end instances. A connector can also have additional points that define the path the connection takes. The actual

point where the drawn connector starts and ends is calculated based on where the line drawn from the instance's coordinates to the next connector point intersects with the instance border (see Figure 7). The exception to this is when the connector point the line will be drawn towards is within the bounds of the Instance in either the x- or y-direction. In that case, either a horizontal or a vertical line will be drawn from the Instance border to the connector point (the shortest possible path). The connectors can have descriptive labels, which, unless the label point is explicitly defined, are positioned in the middle of the connector path.

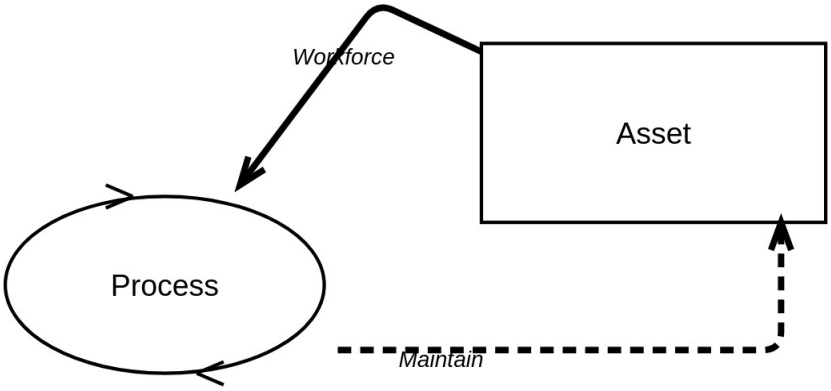

Figure 7: Example of connectors

Similarly to the instances, how the connectors are displayed is also determined by their class, along with additional optional attributes:

- Connectors of the *Used In* class are drawn from assets to processes as a continuous black line. They have a predefined set of labels, of which one or more can be applied (for example, 'Attraction' or 'Stock'). The S*tock* label additionally adds two decorative lines to the base of the connector to emphasize it. Additionally, the *Used In* connector can have highlighted and/or transitive properties, with the former making the line thicker and the latter being represented by having a double line for the connection.
- *Manages* connectors are drawn from processes to assets as a dashed black line. Like with *Used In* connectors, these can have highlighted or transitive properties. These connectors can have three types of labels: 'Acquire', 'Maintain', and 'Retire'. Additionally, how the label is displayed can be changed from the default text representation to a symbol representation of '+', '~', or '-', respectively. If more than one label is applied, all the labels are displayed as text.

- The *relates-to* connector can be drawn between a note and another instance and is a continuous black line. Unlike the other connectors, it does not have an arrow at the end to signify direction. Its appearance can be set to either ‘Default’, ‘Important’, or ‘Very Important’, which will determine its width. Since, unlike with other instances, the note’s position attribute defines its top left point instead of the center point, the *relates-to* connectors also draw the connection towards the note’s top left corner instead of the center of the instance.

- The *Inspects/Monitors* connector can be drawn from processes to other instances besides notes. These connectors are blue and have a decorative rectangle at the base. Their line consists of alternating dashes and dots, and they can have a highlighted appearance. Unlike the previous connectors, the label set for these can be any text, and the label can be oriented either horizontally or vertically.

- The *Drawing/Adding* connector can be drawn between processes and pools, pools and external actors, and from assets to processes. It has a dashed blue line with a circle at the base of the connector. Like with *Inspects/Monitors* connectors, the label can be set freely and positioned vertically, and the appearance can be set to highlighted.

- The *Association* connector is the most general connector type as it can be drawn between any instances besides notes. It has a dotted blue line, which can have a highlighted appearance. Like the other blue connectors, it can have a highlighted appearance, and its label can be set to be vertical or horizontal with freely chosen text. It also has a unique direction attribute, which can be set to unidirectional or symmetric. If set to the former, it will only have an arrow at the end of the connector to indicate direction; if set to the latter, it will have an arrow at both ends.

6. Additional improvements

In addition to adding the annotation functionality, a few additional improvements were made to the application.

6.1 User role changes

The Viewer role is the role with the least privileges that can be created for the FEM Viewer. To avoid having to create many different accounts when wanting to allow access to the models for multiple people, it was convenient to have a single Viewer user shared between them. The problem was, that in the original FEM Viewer, the users with the Viewer role were able to change their passwords. This made it possible for one person using the account to lock out the others. To prevent this, this functionality was disabled for the user with the Viewer role. Instead, it was made possible for the Developer and Admin users to change the passwords for users they have created.

Additionally, a new Expert user role was added. This role has the same privileges as the Viewer role, except that users with this role can use the new annotation functionality implemented during this thesis. This role is intended for business people who can use their expertise to provide feedback to the model but who do not need to be able to upload models or create users themselves.

6.2 Model export

To be able to import the modified XML file back into the FEM Toolkit, we need to be able to export it from FEM Viewer. To enable this, a new model download route was added, together with download buttons for each of the models in the dashboard view next to the view model button.

7. Conclusion

This thesis set out to improve the FEM Viewer web application by enabling users to annotate the Fractal Enterprise Models. It gave an overview of FEMs and the existing FEM Viewer application and described the new functionality and its implementation. The application was improved by adding the ability to edit parts of the uploaded models. This involved adding new parsing functionality that enabled making changes to the XML files exported from FEM Toolkit. To properly display the annotations, the way the models are displayed was reworked to only use XML data instead of relying on uploaded SVG files. Additionally, a way to export the changed models was added. Finally, a new role was created for users who need to be able the annotate the models.

Reference

[1] I. Bider, E. Perjons, M. Elias, and P. Johannesson, ‘A fractal enterprise model and its application for business development’, *Softw Syst Model*, vol. 16, no. 3, pp. 663–689, Jul. 2017, doi: 10.1007/s10270-016-0554-9.

[2] I. Bider, E. Perjons, and V. Klyukina, ‘Tool Support for Fractal Enterprise Modeling’, in *Domain-Specific Conceptual Modeling: Concepts, Methods and ADOxx Tools*, D. Karagiannis, M. Lee, K. Hinkelmann, and W. Utz, Eds., Cham: Springer International Publishing, 2022, pp. 205–229. doi: 10.1007/978-3-030-93547-4_10.

[3] S. Langel, ‘Web-based viewer for Fractal Enterprise models’, University of Tartu, Tartu, 2022. Accessed: Apr. 14, 2025. [Online]. Available: https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=74422

[4] I. Bider and S. Langel, ‘Creating a Web-Based Viewer for an ADOxx-Based Modeling Toolkit’, 2024, pp. 65–73. doi: 10.1007/978-3-031-59468-7_8.

[5] I. Bider, ‘Structural Coupling, Strategy and Fractal Enterprise Modeling’, 2020, pp. 95–111. doi: 10.1007/978-3-030-50316-1_6.

[6] E. R. Harold and W. S. Means, *XML in a Nutshell*. O’Reilly Media, Inc., 2004.

[7] J. D. Eisenberg and A. Bellamy-Royds, *SVG Essentials*, 2nd ed. Beijing Köln: O’Reilly, 2014.

[8] ‘Claude 3.7 Sonnet’. Accessed: May 09, 2025. [Online]. Available: https://www.anthropic.com/claude/sonnet

[9] ‘fast-xml-parser’, npm. Accessed: Apr. 30, 2025. [Online]. Available: https://www.npmjs.com/package/fast-xml-parser

Appendices

The source code for the improved FEM Viewer is available at https://github.com/JoonatanEhlvest/FEM. The installation instructions are included in the main README.md.

License

I, Joonatan Ehlvest

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis Adding annotation functionality to a web-based viewer for Fractal Enterptise models, supervised by Dr. Ilia Bider;

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Joonatan Ehlvest

14/05/2025