Picto Web: A Tool for Complex Model Exploration

Alfa Yohannis, Dimitris Kolovos, Antonio García-Domínguez,
Carlos Javier Fernández Candel

University of York, United Kingdom

Abstract

Exploring and visualising a complex model in a single view can be challenging as it can require a considerable amount of computational resources and make the model difficult to comprehend due to information overload. Picto Web is a tool designed for complex model exploration. Using the Epsilon Generation Language for model-to-text transformation, it can transform domain-specific models into multiple transient web-based views in different formats, such as HTML, Graphviz, and PlantUML. Picto Web uses lazy and incremental model-to-text transformation to (re)generate views efficiently. Moreover, it supports push notifications to clients when views are updated and is packaged as a Docker container for ease of use.

Keywords: complex models, model exploration, picto, web, visualisation

Metadata

This ancillary data table is required for the sub-version of the codebase. Please replace the text in the right column with the correct information about your current code and leave the left column untouched.

1. Motivation and significance

A model is commonly considered complex when it has a large number of heterogeneous elements which are involved in non-trivial relationships [1, 2]. One way to understand and communicate such models is through visualisation. However, displaying all elements and their relationships in a single graphical representation is often undesirable since it can overload users with information and make visualisations challenging to comprehend. In terms of computation, this approach is also inefficient for large models since a potentially very large and complex diagram (or similar visual representation) has to be rendered in one go. Moreover, in a multi-user environment, the visualisation also has to deal with multiple, repeated requests, whether a

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.1.02-SCP
C2	*https* : *//github.com/epsilonlabs/picto-web*	
C3	Permanent link to Reproducible Capsule	
C4	Legal Code License	Eclipse Public License
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Java, Javascript, HTML, CSS, Graphviz
C7	Compilation requirements, operating environments and dependencies	Java 11, Maven
C8	If available, link to developer documentation/manual	For example: *https* : *//github.com/epsilonlabs/picto-web*
C9	Support email for questions	Online forum *https* : *//www.eclipse.org/forums/index.php/f/22/*

Table 1: Code metadata

user is refreshing a view or different users requesting the same view over the network. Thus, novel mechanisms are necessary to allow engineers and other stakeholders to perform contextual exploration of complex models from different viewpoints and at varying levels of detail. The visualisation also should be able to handle repeated requests from multiple users and return the requested views promptly.

In this paper, we present Picto Web, a web-based version of the Picto[1] [3] model visualisation tool which was originally implemented as an extension of the Eclipse IDE. Unlike Picto, Picto Web does not require local software installation, which makes it more suitable for a broader audience of developers and stakeholders who would benefit from access to dynamically-generated graphical views of complex software and system models.

Sprotty [4] is an open-source diagramming framework that uses web technologies to render graphical views. Besides being used on the client-side for fast, scalable SVG rendering with ELK layout, it also provides a headless component that can be used for integration with the Xtext framework, extending Language Server Protocol, and computing diagram layout on the

[1] `https://www.eclipse.org/epsilon/doc/picto`

server side. Sprotty only supports node-edge diagrams and does not support rendering views conforming to different technologies, such as tables in HTML, PlantUML and Graphviz diagrams. Moreover, Sprotty does not support semantic editing on diagrams. As an alternative, the Graphical Language Server Platform (GLSP) [5] allows complete control and access over the Sprotty diagrams and can be considered if diagram editing via web browsers is required.

The KIELER Lightweight Diagram (KLD) framework [6] uses Xtend, Piccolo2D and EMF to allow on-demand model visualisation, and also the KIELER Infrastructure for Meta Layout (KIML) for defining diagram layouts. KLD uses three EMF-based models (KLayoutData, KRendering, and KGraph) for users to describe a diagram. KLD is also extensible via Java/Xtend interfaces, which should be implemented to transform and map a semantic model to KGraph and KRendering models. Similar to Sprotty, KLD only supports rendering views in node-edge diagrams. As a comparison, Picto and Picto Web support rendering views in multiple formats, such as table/form views in HTML, PlatUML and Graphviz diagrams, and views generated using JavaScript graphical libraries, e.g., Three.js.

Picto [3] is an Eclipse plugin that uses lazy model-to-text transformation to generate transient graphical views from heterogeneous models. Figure 1 shows a screenshot of Picto within Eclipse.

In this example, we wish to visualise models conforming to a contrived social network metamodel from [3]. In particular, we use a sample model containing four people (Alice, Bob, Charlie and Dawn) and like/dislike relationships between them. From such social network models we wish to produce one node-edge view for the entire social network, and one view for each member of the network that omits anyone they neither like nor dislike.

Picto's user interface has two main components. On its left-hand side is a tree widget that displays the titles and icons of the views (Social Network, Alice, Bob, Charlie and Dawn) that users can select from. Once a view is selected, its content is generated and rendered in an embedded web browser – through a series of transformations – on the right-hand side of Picto. In Figure 1, the selected view visualises the entire model in the form of a Graphviz-based node-edge diagram. Beyond Graphviz, Picto also supports PlantUML and SVG/HTML as visualisation formats, and also features an extensible architecture that enables adopters to extend it with additional diagram-as-code formats.

A challenge with Picto is that it is implemented as a plugin of the Eclipse IDE, which means that for engineers to use it, they need to install Java, Eclipse and Picto and check out the models they wish to explore (as well as the respective visualisation transformations) locally. While this is not an

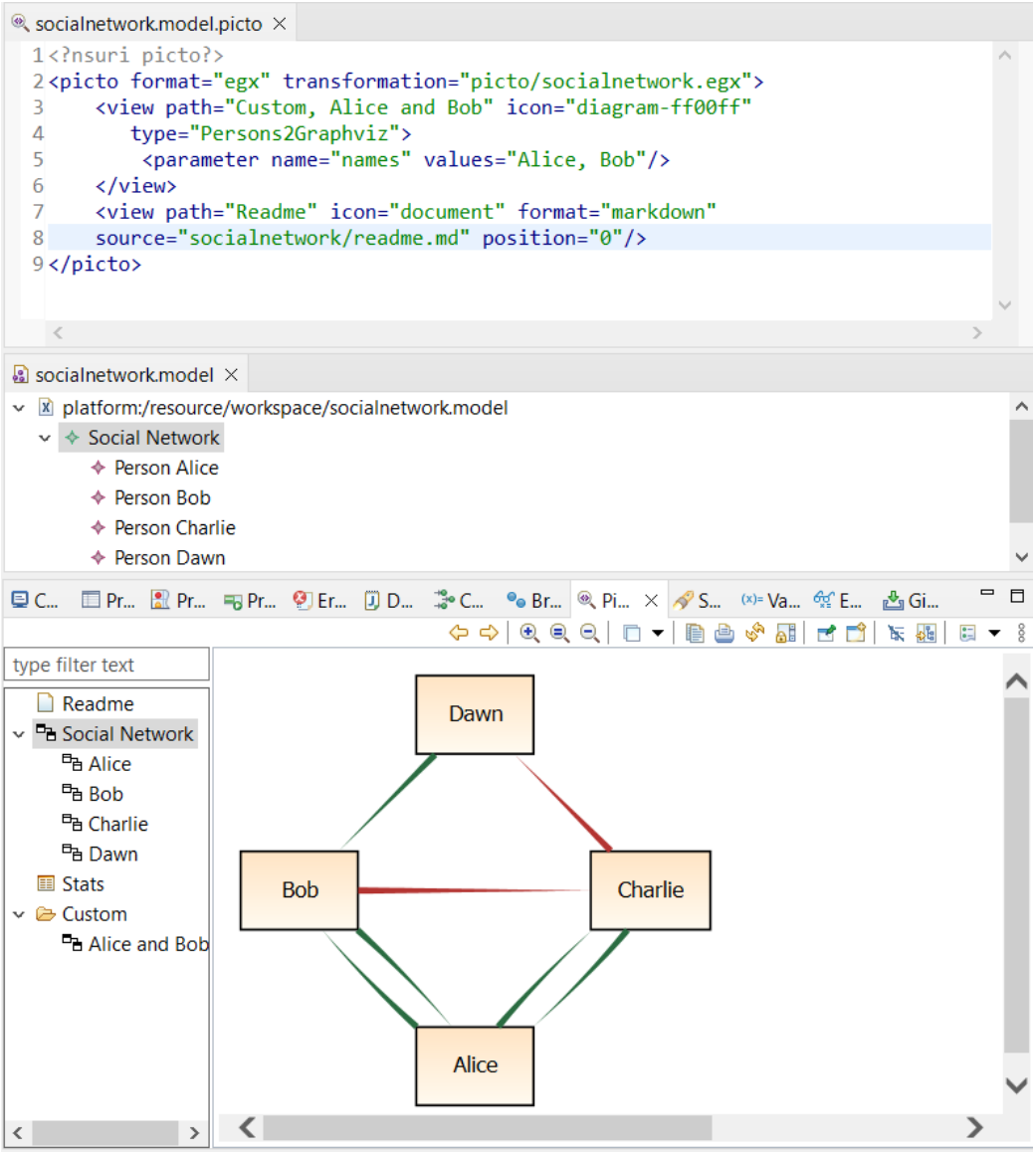


Figure 1: Picto plugin for Eclipse environment.

issue for software developers who are already familiar with Eclipse, it can represent a significant barrier for other engineers and stakeholders.

2. Software description

Picto Web[2] is a web-based version of Picto. It maintains the core features of the Eclipse-based version of Picto, such as supporting the lazy generation of views, drilling-up/down tree views, multi-views, and multi-formats, but it is implemented as a multi-tenant client-server web application that can be accessed through a standard web browser to perform interactive complex model exploration. Picto Web monitors model files and visualisation model-to-text templates for changes and immediately propagates any updates to generated views to the viewers' web browsers, delivering a live user experience.

2.1. Software architecture

The architecture of Picto Web is displayed in Figure 2 and consists of two parts: a server-side part that lazily runs visualisation transformations against models, and a browser-based client that displays the visualisation results.

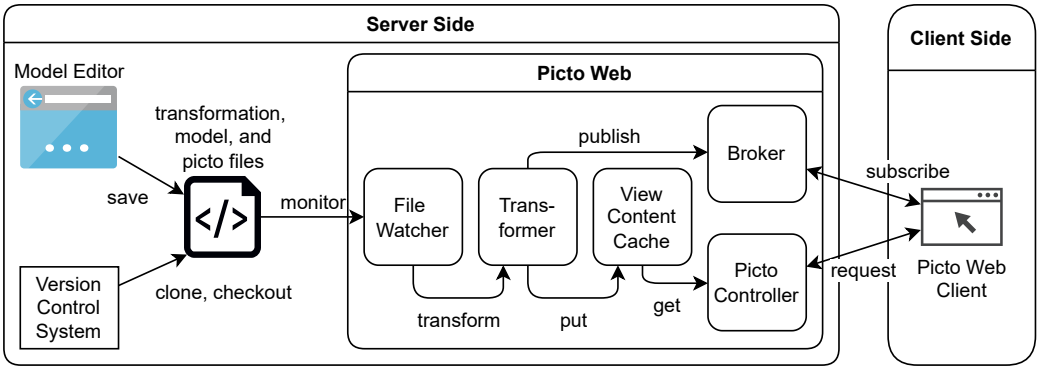


Figure 2: The architecture of Picto Web.

The server-side part of Picto Web consists of the `FileWatcher`, `Transformer`, `ViewContentCache`, `PictoController`, and `Broker` components shown on the left hand side of the figure. The `FileWatcher` component is responsible for monitoring a directory that contains the models that Picto Web needs to visualise (see Section **??**). When any of the files is modified (added, deleted, or updated), `FileWatcher` detects the change and notifies

[2] `https://github.com/epsilonlabs/picto-web`

the `Transformer` component, which then generates, refreshes, or invalidates the contents of the respective views[3].

After that, Picto Web puts the generated views into a cache (`ViewContentCache`) that maps the paths of the views in the tree view to their actual contents. The cache is used to avoid wastefully re-generating views which cannot have been impacted since the last time a client requested them. Therefore, Picto Web can respond promptly to multiple, repeated requests for the same views.

Every time a client requests the content of a specific view, `PictoController` receives and handles the request by retrieving the view content from `ViewContentCache` using the view's path sent in the request as the key for the map. The retrieved content is then returned to the client to display.

`Broker` is a component responsible for pushing updates to Picto Web clients. Every time a Picto Web client starts visualising a model, the client subscribes to the `Broker` so that it can receive further updates from the Picto Web server-side component if any of the model files are subsequently modified. After transformation, the `Transformer` publishes new content views to the `Broker`.

We have packaged the Picto Web server in the form of a Docker image[4] so that users can easily deploy and test it using the following command.

```
docker run --rm -it -v $PWD:/workspace \
-p 8080:8080 picto-web
```

The Picto Web server can then be accessed using a web browser via `http://localhost:8080`. The `-v $PWD:/workspace` option is used to allow the Docker image to access the transformation and model files in the current directory from its internal `/workspace` directory. This way, the internal `FileWatcher` in the Picto Web Docker image can monitor the file system for any relevant changes that should trigger view generation or invalidation.

2.2. Software functionalities

Present the major functionalities of the software.

2.3. Sample code snippets analysis or use cases of the software (optional)

2.3.1. Transformation

Picto requires at least three types of files: model files (any model representation format supported by Epsilon[5] is also supported by Picto Web),

[3]The mechanisms used to implement incremental view invalidation and re-generation upon changes in models and templates are based on the work in [7], but a detailed description is beyond the scope of this paper.

[4]`https://hub.docker.com/r/alfayohannisyorkacuk/picto-web`

[5]`https://eclipse.org/epsilon/doc/emc`

Listing 1: A social network model as the input file for the lazy transformation. The format of the ids is simplified.

```
<?xml version="1.0" encoding="ASCII"?>
<SocialNetwork xmlns="socialnetwork" xmi:id="sn">
<people xmi:id="a" name="Alice" likes="b c"/>
<people xmi:id="b" name="Bob" likes="a" dislikes="c"/>
<people xmi:id="c" name="Charlie" likes="a" dislikes="d"/>
<people xmi:id="d" name="Dawn" likes="b"/>
</SocialNetwork>
```

Listing 2: The Network2Graphviz EGX rule

```
rule Network2Graphviz
transform n : socialnetwork::SocialNetwork {
        template : "socialnetwork2graphviz.egl"
        parameters : Map{
                "path" = Sequence{"Social Network"},
                "format" = "graphviz-circo",
                "people" = n.people
        }
}
```

visualisation transformation files in Epsilon's EGL template-based language (*.egl, *.egx), and Picto files (*.picto) which bind models to visualisation transformations and specify custom views (EGL, EGX, and Picto files are explained briefly later in this section). Listing 1 shows the XMI-based representation of the social network model of our running example.

The visualisation transformation consists of EGL (Epsilon Generation Language) templates [8] that produce individual views, and an EGX (EGL Co-Ordination Language)[6] program that specifies how the templates should be executed against different elements in the model(s). Listing 2 shows an excerpt of the EGX transformation coordination program of this example. Rule `Network2Graphviz` runs the `socialnetwork2 graphviz.egl` template of Listing 3 for every `SocialNetwork` element in the source model and produces a Graphviz-based view for it. The EGL template of Listing 3 defines the logic for generating the content of the view. An actual generated view in Graphviz is shown in Listing 4.

Picto files bind the transformation coordination and model files to generate the contents of the target views. In Listing 5, Picto Web uses the `socialnetwork.egx` transformation coordination file that contains the rule in Listing 2 to transform the social network model in Listing 1 into the social network and individual person views shown in Figure 3. It also includes two

[6] `https://www.eclipse.org/epsilon/doc/egx`

Listing 3: An EGL template that generates a Graphviz representation of a social network.

```
1          digraph G {
2                  node[shape=rectangle, fontname=Tahoma, fontsize=10,
                       style="filled", gradientangle="270", fillcolor="
                       bisque:floralwhite"]
3                  edge[penwidth=3, style=tapered, arrowhead=none]
4                  [%for (p in people){%]
5                          [%=p.name%]
6                          [%for (l in p.likes){%]
7                                  [%=p.name%] -> [%=l.name%] [color=green]
8                                  [%}%]
9                          [%for (l in p.dislikes){%]
10                                 [%=p.name%] -> [%=l.name%] [color=red]
11                                 [%}%]
12                         [%}%]
13         }
```

Listing 4: A view generated by the EGL template in Listing 3.

```
1          digraph G {
2                  node[shape=rectangle, fontname=Tahoma, fontsize=10,
                       style="filled", gradientangle="270", fillcolor="
                       bisque:floralwhite"]
3                  ...
4                  Alice
5                  Alice -> Bob [color=green]
6                  ...
7                  Dawn
8                  Dawn -> Bob [color=green]
9          }
```

Listing 5: The Picto file that binds the model and the visualisation transformation.

```
1  <?nsuri picto?>
2  <picto format="egx" transformation="picto/socialnetwork.egx">
3  <view path="Custom, Alice and Bob" icon="diagram-ff00ff" type="
       Persons2Graphviz">
4  <parameter name="names" values="Alice, Bob"/>
5  </view>
6  <view path="Readme" icon="document" format="markdown" source="
       socialnetwork/readme.md" position="0"/>
7  </picto>
```

additional views: (1) a custom view that includes only Alice and Bob's social network (Figure 3d) and (2) a view that displays a Markdown readme file.

3. Illustrative examples

This paper uses the same contrived social network example as in the original Picto documentation[7] to explain Picto Web. As a brief refresher, a social network is a graph that contains a number of people as its nodes linked with *like* and *dislike* relationships. In a view, as shown in Figure 3a, a person is depicted as a rectangle while the *like* and *dislike* relationships are presented as green and red edges respectively. This node-edge view is the visualisation of the social network model defined in Listing 1.

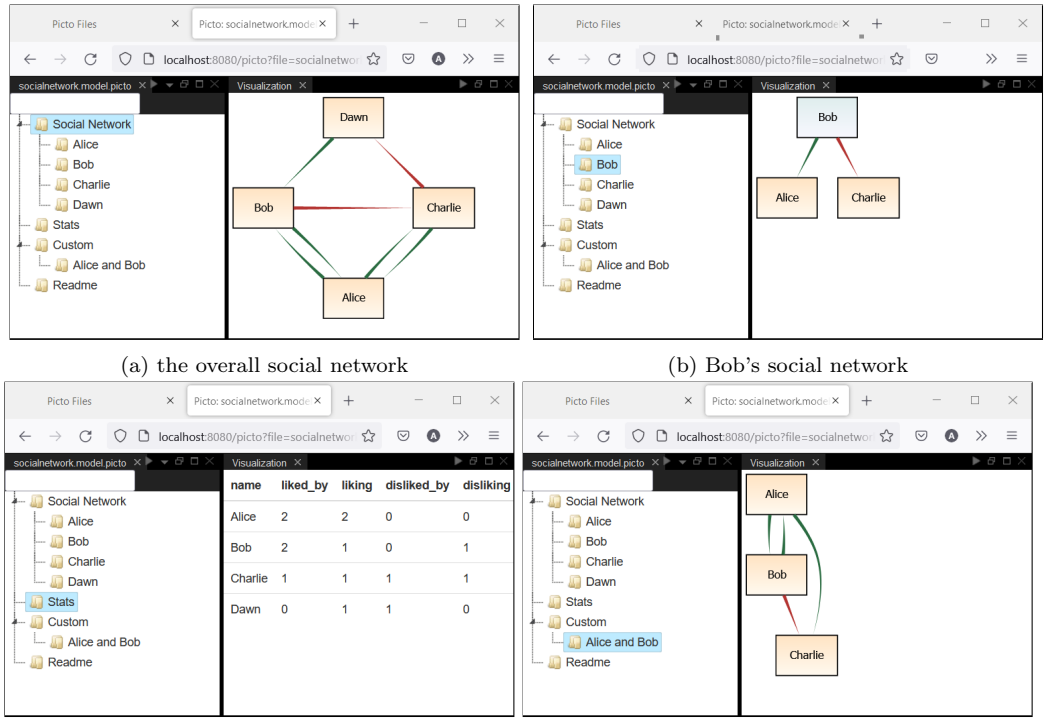
3.1. User Interface

Figure 3 shows the user interface (UI) of Picto Web. The UI has two main areas. The left side is a tree view which lists the views produced from the underpinning model(s) in a hierarchical way, and the right side is a panel that displays the content of the active, selected view. In our example, when a user selects the *Social Network* view on the left hand side, Picto Web displays the overall social network of Alice, Bob, Charlie, and Dawn (Figure 3a).

Viewers can also drill down to display the local social network of each person through the tree view on the left. For example, Figure 3b displays Bob's social network. Besides being able to display view contents in SVG/HTML as in Figures 3a and 3b, Picto Web is also able to render model views in other formats. For example, Figure 3c displays a summary of the social network in the form of a HTML table.

In addition to procedurally-generated views (e.g. one view for each person in the network), developers can also define custom views, as shown in Figure 3d. The figure is customised only to display the social network of Alice and Bob.

[7] `https://eclipse.org/epsilon/doc/picto`



(a) the overall social network

(b) Bob's social network

(c) an HTML table summarising the social network

(d) custom Alice and Bob's social network

Figure 3: Picto Web client displays different views.

3.2. Control Flow

This section demonstrates Picto Web's internal control flow, visualised in the sequence diagram of Figure 4. Suppose that the `ViewContentCache` has been populated with view contents through an initial transformation executed when a developer (Lisa) created the social network model – the process is not displayed in the diagram. Stakeholders Mike and Nick wish to explore Lisa's model and start running Picto Web clients. The clients subscribe to the broker provided by the Picto Web server application so that they can receive further updates on the model.

Mike is interested in knowing more about Alice's social network. Therefore, he sends a view request to Picto Web with a parameter containing Alice's view path, "/0/1"[8]. Since the view content of the path is already available in the `ViewContentCache` from the initial transformation, `PictoController` (the component that handles the request) retrieves Alice's view content from the cache and sends it back to Mike's client to be displayed.

At some point, Lisa decides to rename "Alice" to "Alex" and save the

[8]Simplified format

change to the model file. `FileWatcher` detects that the file has been modified and notifies the `Transformer` to generate new view contents from the modified file and update the `ViewContentCache`. The `Transformer` also publishes the new content views to the `Broker` so that the subscribers, Mike and Nick, also receive the new content views and perform live updates on the view contents they currently display. s

4. Impact

This is the main section of the article and reviewers will weigh it appropriately. Please indicate:

- Any new research questions that can be pursued as a result of your software.
- In what way, and to what extent, your software improves the pursuit of existing research questions.
- Any ways in which your software has changed the daily practice of its users.
- If applicable, how widespread the use of the software is within and outside the intended user group (downloads, number of users if your software is a service, citable publications, and so on).
- If applicable, how the software is being used in commercial settings or how it has led to the creation of spin-off companies. Please note that points 1 and 2 are best demonstrated by references to citable publications.

5. Conclusions

In this paper, we presented Picto Web, a web-based version of the Picto model visualisation tool which was originally implemented as a plug-in of the Eclipse IDE. Unlike Picto, Picto Web does not require local software installation, which makes it more suitable for a broader audience of developers and stakeholders who would benefit from access to dynamically-generated graphical views of complex software and system models.

6. Future Plans (optional)

Going forward, we plan to further improve the performance of visualisation transformations in Picto Web by establishing an element/property access trace of the visualisation transformations, to enable more fine-grained regeneration and invalidation of generated views.

Acknowledgements (optional)

The work in this paper has been funded through the HICLASS InnovateUK project (contract no. 113213).

References

If the software repository you used supplied a DOI or another Persistent IDentifier (PID), please add a reference for your software here. For more guidance on software citation, please see our guide for authors or this article on the essentials of software citation by FORCE 11, of which Elsevier is a member.

References

[1] N. Boccara, Modeling Complex Systems, Springer New York, New York, NY, 2010, pp. 1–23. doi:10.1007/978-1-4419-6562-2_1.
URL https://doi.org/10.1007/978-1-4419-6562-2_1

[2] R. E. Klosterman, Simple and complex models, Environment and Planning B: Planning and Design 39 (1) (2012) 1–6. arXiv:https://doi.org/10.1068/b38155, doi:10.1068/b38155.
URL https://doi.org/10.1068/b38155

[3] D. Kolovos, A. de la Vega, J. Cooper, Efficient generation of graphical model views via lazy model-to-text transformation, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 12–23. doi:10.1145/3365438.3410943.
URL https://doi.org/10.1145/3365438.3410943

[4] Sprotty, eclipse/sprotty: A diagramming framework for the web, accessed: 2022-06-22 (2022).
URL https://github.com/eclipse/sprotty

[5] Eclipse Foundation, Documentation: Eclipse graphical language server platform, accessed: 2022-06-23 (2022).
URL https://www.eclipse.org/glsp/documentation/

[6] C. Schneider, M. Spönemann, R. von Hanxleden, Just model!—putting automatic synthesis of node-link-diagrams into practice, in: 2013 IEEE Symposium on Visual Languages and Human Centric Computing, IEEE, 2013, pp. 75–82.

[7] B. Ogunyomi, L. M. Rose, D. S. Kolovos, Incremental execution of model-to-text transformations using property access traces, Software & Systems Modeling 18 (1) (2019) 367–383. doi:10.1007/s10270-018-0666-5. URL https://doi.org/10.1007/s10270-018-0666-5

[8] L. M. Rose, R. F. Paige, D. S. Kolovos, F. A. C. Polack, The epsilon generation language, in: I. Schieferdecker, A. Hartman (Eds.), Model Driven Architecture – Foundations and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 1–16.

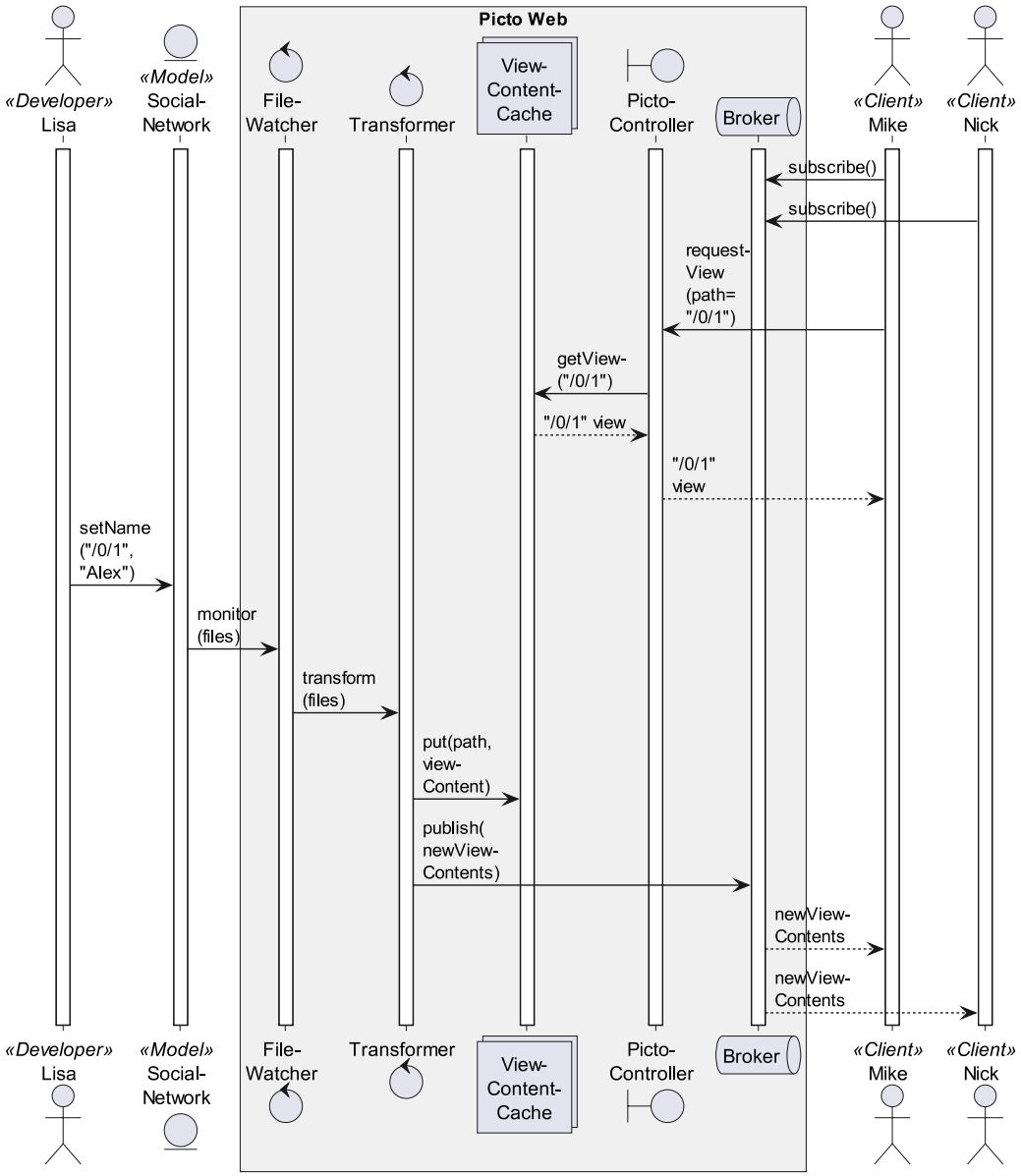


Figure 4: Requests and Updates of Picto Web's view contents.