ELVIZ: A Query-Based Approach to Model Visualization

Marie-Christin Ostendorp, Jan Jelschen, Andreas Winter

University of Oldenburg
Germany
{ostendorp, jelschen, winter}@se.uni-oldenburg.de

Abstract: Visualization is an important technique for understanding and exploring complex models. To be useful, visualizations have to be specifically tailored towards the visualization task and the analyzed model. Many standard charts or graph-based visualizations exist, but need to be mapped to the concepts of the model under study. Existing model visualization tools often have predetermined visualization kinds and content, impeding reuse of standard visualizations for various purposes, or lacking the ability to flexibly map source model concepts to different visualization elements.

This paper presents ELVIZ („Every Language Visualization"), an approach to visualization, generic regarding both the source model, and the kind and content of the visualization. ELVIZ applies model-driven engineering techniques both to transform arbitrary source models into the desired visualization models, and to generate said model transformations from a query-based mapping of source model concepts to visualization concepts. This cleanly decouples source and visualization meta-models, allowing to reuse and combine standard visualizations for various source models.

The ELVIZ approach is applied to scenarios from software visualization in software evolution and measuring energy consumption of mobile applications, using different kinds of visualizations.

1 Motivation

Models are everywhere in computer science and software engineering, e.g. process models, test models, or design models [Béz13]. To ease the understanding of these models, or serve as a basis for communication, visualizations are very helpful. For example, to investigate code smells like too large classes in an existing software system a bar chart might be very helpful: Instead of investigating the underlying code manually by regarding every single class, a bar chart - in which each bar represents one class of the system and where its height is set by the count of methods - is able to present the required information at one glance. This bar-chart example will be used in Section 3.1 again to clarify the approach presented in this paper using the model of a small software system as depicted in Figure 2 to generate the bar chart as presented in Figure 5c.

Many tools exist to generate such visualizations. However, the approaches often support only a single kind of visualization, e.g. from using standard graphics like simple bar charts, to elaborate visualizations like using a three-dimensional "city" to represent sets of characteristics of a model as properties of its buildings [WL07]. Which visualization is appropriate depends on both the model under study, and the goals of the analysis task the visualization is set to support. For instance, instead of visualizing the count of methods per class to identify code smells, the aim of the visualization can be to show the percentual distribution of the methods onto the different classes. In this case a pie diagram might be more appropriate than a bar chart: Another visualization is needed but the visualization content stays the same - the pieces should represent the classes and its size is set by the count of methods of the particular class. Instead of generating a totally new visualization or even taking another visualization tool into account, it would be more pleasant and time-saving to reuse the specification of the visualization content for another visualization kind. A reverse scenario might be that the content of the desired visualization changes e.g. visualizing the percentual distribution of the attributes onto the classes, but the visualization kind - a pie diagram - stays the same. In this case the reusability of the visualization kind is required.

So what is needed is an approach where visualization content and kind can be specified separately, enabling independent reuse for different source models. This should be provided by ELVIZ („Every Language Visualization"): ELVIZ is a generic, query-based model visualization approach, using model-driven techniques to decouple source models, visualization models, and the tools used for rendering. In this context, rendering means layouting the models content according to predefined visualization needs. Within ELVIZ, a model can be of arbitrary structure, as long as this structure is rigidly defined by an appropriate meta-model. Queries over the source meta-model (expressed in an appropriate query language) are used to define the concepts to be visualized - the visualization content. Visualization kinds i. e. the paradigms followed by concrete visualizations, are represented by their meta-model. Queries can be associated with concepts of the visualization meta-model, to form a mapping, which is automatically turned into a model transformation. Once a visualization kind is specified by a meta-model and the appropriate renderer is provided content, this specification can be reused for different source models in various combinations with different specifications of the content via queries. So ELVIZ can be seen as a framework to generate an appropriate visualization tool rather than a visualization tool itself.

In the following, ELVIZ is described in detail, starting with the presentation of related work in Section 2. Section 3 clarifies ELVIZ with an illustrative example. In Section 4, ELVIZ is applied to two different areas of application – to the field of software metrics visualization, and to the field of power consumption of applications on mobile devices. Section 5 summarizes the presented ELVIZ-approach and refers to possibilities to further optimize and build upon it in the future.

2 Related Work

ELVIZ's main use case is data visualization for software analysis in software evolution: In the past various tools and approaches were published in this context: These include graph-based visualization as for example „Tulip"[Aub01], „da Vinci" [FW94], „ASK-Graphview" [AvHK06], and „GraphViz" as an Open source graph (network) visualization project [EGK$^+$01]. Apart from the approach to visualize models using graphs, there exist visualization approaches especially developed for the field of software visualization: the Unified Modeling Language is a well-known OMG-Standard to define models graphically [OMG06]. Lanza et. al presented visualizing software systems as „CodeCities” [WL07].

In these approaches, the kind and content of the visualization is predetermined. To get a different visualization, the person creating the graphical representation has to get familiar with these tools and approaches. In contrast to this, ELVIZ aims at leaving the choice for a suitable graphical representation - including kind and content - to the person creating the visualization. Thereby, ELVIZ is a framework to generate an appropriate visualization tool rather than a visualization tool itself.

The BIRT framework presented by the Eclipse Foundation [Ecl14] is an eclipse plugin able to visualize different contents with different kinds of visualization. BIRT is not easily expendable by new totally individual kinds of visualization as it has a limited range of available report items. Furthermore BIRT can not easily be integrated into a larger toolchain - e.g. extending for instance a metric calculation tool to visualize its results directly. In contrast ELVIZ aims at being easily extendable and should provide the possibility to be integrated into a larger toolchain.

Therefore, ELVIZ is based on model-driven techniques [Ken02], to allow flexible and automatic specification and generation of desired model visualizations.

Similar works on this kind of visualization generation by applying model-driven engineering to the field of model visualization were introduced [WW05] [BSFL06]. Wolff and Winter [WW05] presented the transformation of Bauhaus Graphs into UML diagrams. Both approaches applied MDE to automatically generate the visualizations. In contrast to this, the ELVIZ approach aims at extending this procedure in that way that not only the visualization is generated automatically but also the required transformation itself.

The ELVIZ-approach combines ideas from the field of graph technologies – especially the idea of querying graphs [ERW08] with applying model driven engineering to the field of information visualization [BSFL06]. to allow independent reuse of visualization content and kind.

3 The ELVIZ-Approach

An overview of ELVIZ is depicted in Figure 1. Two major processes can be distinguished: **1)** *Generating a Visualization Tool* to create a customized visualization tool, and **2)** *Executing a Visualization Tool* using this generated tool to produce visualizations of provided models. The objective of the tool generation process lies in the construction of a reusable visualization tool for a new kind of visualization. This tool can be used to produce a concrete graphical output for different source models. Both processes are briefly outlined below, followed by more detailed explanations using an example in Section 3.1.

For the tool generation process, it is necessary to specify the input and output of the tool to be generated: Therefore, the following steps have to be performed: First, the input has

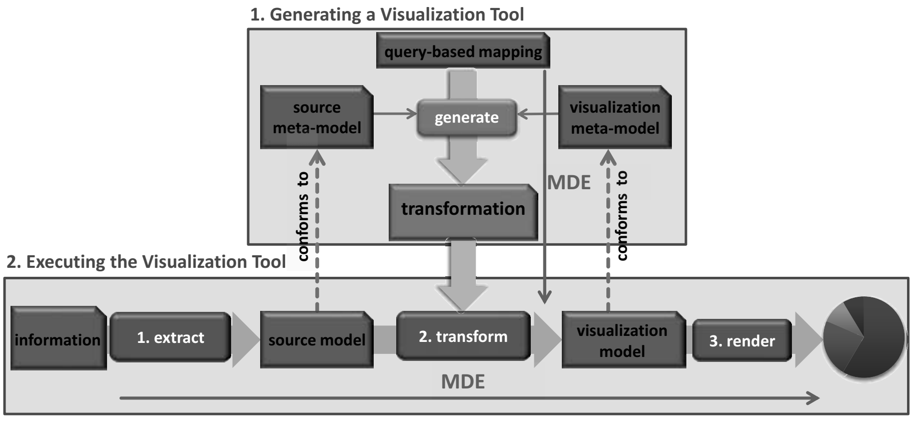


Figure 1: Overview of the ELVIZ-approach.

to be specified via the *source meta-model*. The graphical output has two parts to be specified: the kind of visualization, and its content. These two aspects are specified separately from each other to allow separate reusability of these specifications for other tool generations, thereby reducing workload for generating a new tool. The kind of visualization can be specified via the *visualization meta-model*, while the content is specified as *query-based mapping* between the elements of the source meta-model and the elements of the visualization meta-model. By associating a query over the source meta-model with each concept of the visualization meta-model, a mapping is defined: This mapping describes which elements of source models should be represented by which elements in graphical representations.Defining queries is not necessarily easier than writing transformations. However, ELVIZ allows to manage queries and specfication of vizualization kinds separately, enabling independent reuse, which is the real benefit. Furthermore queries do not limit the possibilities for the contents of graphical representations [HE11].

ELVIZ assumes that input data is either readily available as models conforming to well-defined meta-models, or that an appropriate fact extractor (e.g. a *parser* in case of source code analysis) is provided to create such a representation from “plain” input data. Also, besides the specification of the visualization via a visualization meta-model, a *renderer* has to be available which creates the real graphical output conforming to the kind of visualization. Such a renderer has to be implemented only once for a given visualization meta-model, though alternative realizations are possible. As last step, the source meta-model, the mapping, and the visualization meta-model are used to automatically *generate the transformation* making up the customized visualization tool.

Visualization tools generated in this way are used in the execution process. Here, the following steps have to be performed: *extracting*, *transforming*, and *rendering* the data. As a first step, a *source model* is extracted from the *input data* to be visualized. This is a pre-processing step; for example, if the input data consisted of a Java application to be visualized in a software evolution scenario, this step would correspond to parsing the source code to create a representation on abstract syntax tree level. It yields a model representation conforming to the source meta-model, which is a prerequisite for the upcoming transformation step. The next step takes a prior generated transformation, executes it, and generates a *visualization model* fulfilling the mapping and conforming to the visualization meta-model. In the last step, this model is finally transferred to an appropriate renderer to create the actual graphical representation, e.g. as an image file.

An implementation of the ELVIZ approach has been created using the technological space of *TGraphs* [ERW08] to represent models and meta-models. TGraph-based models can be queried and transformed using the *Graph Repository Query Language (GReQL)* [ERW08] and the *Graph Repository Transformation Language (GReTL)* [HE11], respectively. ELVIZ is not tied to these techniques, though. An alternative realization could, for example, be implemented using ATL in OMG's *Model-Driven Architecture (MDA)* [JK06, Sol03]. In this case ATL is used instead of GReTL and OCL replaces GReQL.

3.1 Example – Visualizing Software Metrics with Bar Charts

This example shows how a visualization tool, able to visualize properties and metrics of java systems, is generated using ELVIZ (Section 3.1.1), and how to execute the generated visualization tool to produce the graphical representation(Section 3.1.2).

```
package com.vizsave;
public class Database {

        public void initialize() {…}
        public void store(Picture p) {…}
        public void delete(Picture p) {…}
        public void get(Object Picturep) {…}
}
```

```
package com.vizsave;
public class Picture {
        private Database database;

        public Picture() {…}
        public void edit() {…}
        public void save() {…}
}
```

Figure 2: Source Code of the example system.

In this example scenario, a simple metric: the number of methods per class is to be visualized. This can be represented appropriately by a bar chart, where bars represent classes, and the height of each bar is determined by the number of methods. A small, fictitious Java application to save images in a database is used in this example. It consists of two classes the class „Database" to realize a database connection and the class „Picture". Each contains a few methods, as indicated by the code snippets in Figure 2.

3.1.1 Generating a Visualization Tool

To generate the visualization tool, the input of the tool and the kind and content of the desired visualization need to be specified.

Specifying the input. Since the input are Java software systems, an appropriate Java meta-model is needed. For this simple example, a minimal meta-model is used, representing only those concepts carrying the information needed for the intended visualization (Figure 3 left), and consists of three classes to represent *packages*, *classes*, and *methods*. Their containment relationships are modeled with aggregation associations. Each class also has a *name* attribute.

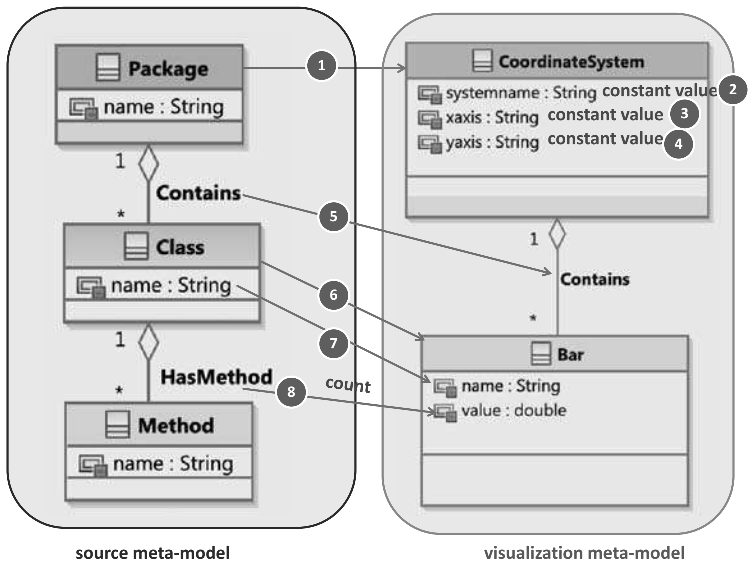
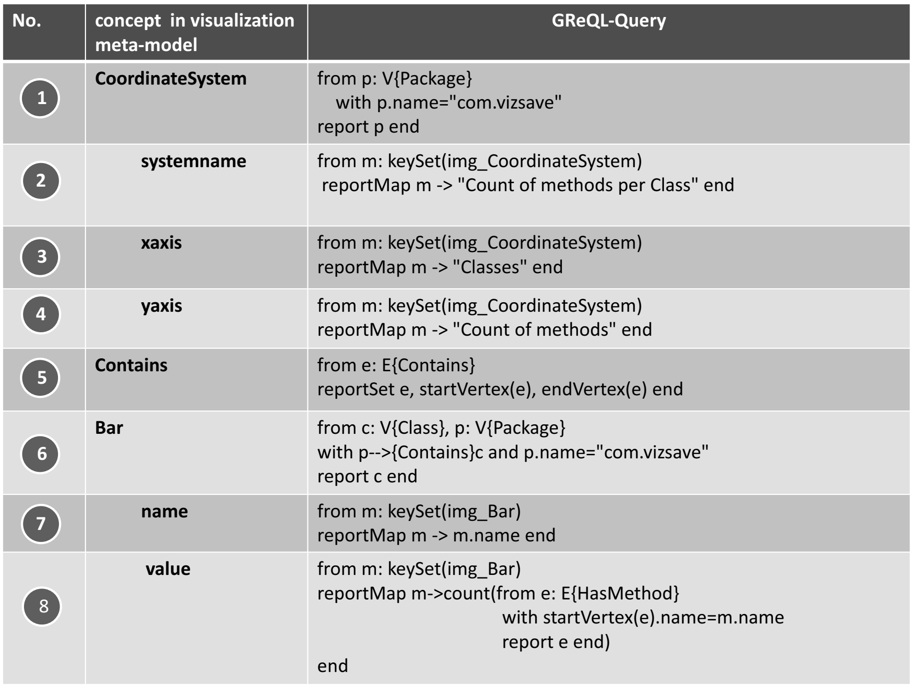


Figure 3: Source meta-model, visualization meta-model and mapping between them.

Specifying the kind of visualization. In the example considered here, the number of methods per class should be visualized as bar chart. The corresponding visualization meta-model is shown on the right-hand side of Figure 3: A bar chart is represented by a *CoordinateSystem*, with a *name* and labels for *x-axis* and *y-axis*. It consists of arbitrarily many *bars*, each having a *name* and *value* determining the height of the bar. In the TGraph-based ELVIZ implementation, visualization meta-models are created using GReTL, which is designed to create target meta-models and models simultanously. It is also possible to use pre-existing target meta-models, and the implementation could easily be extended to also allow visualization meta-models to be specified using UML class diagrams, which can be imported in standard XMI exchange file format.

Specifying the content of the visualization. The desired mapping is indicated by the numbered arrows depicted in Figure 3. Each element of the visualization meta-model must have its counterpart in the source meta-model. For instance, a separate bar chart should be created for each package, which is ensured by mapping these two classes onto each other. Each bar in a coordinate system stands for a Java class in the corresponding package, and the value of the bar is set by the number of methods defined in that class. Therefore, the calculation of the number of *HasMethod* incidences per class has to be specified as part of the mapping to *value*-attributes of bars. To set a specific string as value for labels in the desired graphical output, constant values can be used: For instance, to set the label on the x-axis to “Classes”, the value of the x-axis attribute in class *CoordinateSystem* can be set to the constant string „Classes”.

To realize this mapping, ELVIZ uses queries: queries are based on the source meta-model, and are used to get specific elements out of the source model, or perform calculations over it. By specifying a query for each concept of the visualization meta-model – as classes, attributes and associations–, each query's results can be used in a model transformation to create instances or set attribute values in the target visualization model. In the TGraph-based ELVIZ implementation, queries are expressed using *GReQL*. The GReQL queries for the mapping of this example are shown in Figure 4, with numbers corresponding to those depicted in Figure 3.



No.	concept in visualization meta-model	GReQL-Query
1	**CoordinateSystem**	from p: V{Package} with p.name="com.vizsave" report p end
2	**systemname**	from m: keySet(img_CoordinateSystem) reportMap m -> "Count of methods per Class" end
3	**xaxis**	from m: keySet(img_CoordinateSystem) reportMap m -> "Classes" end
4	**yaxis**	from m: keySet(img_CoordinateSystem) reportMap m -> "Count of methods" end
5	**Contains**	from e: E{Contains} reportSet e, startVertex(e), endVertex(e) end
6	**Bar**	from c: V{Class}, p: V{Package} with p-->{Contains}c and p.name="com.vizsave" report c end
7	**name**	from m: keySet(img_Bar) reportMap m -> m.name end

8	**value**	from m: keySet(img_Bar)
		reportMap m->count(from e: E{HasMethod}
		with startVertex(e).name=m.name
		report e end)
		end

Figure 4: GReQL-Queries for the mapping shown in Figure 3.

GReQL queries usually start with a *from*-clause, specifying the domain of discourse: For example, the GReQL query for the coordinate system (1) defines a variable *p* to range over all nodes (vertices) of type *Package*. Constraints are specified in the with-clause. Here, the with-clause specifies that only nodes with their name set to "com.vizsave" should be considered. In the return-clause of the GReQL-query, the structure of the result returned for each considered element of the domain is defined. In this case, this is just the node itself, yielding a list of *Package*-nodes, which have the specified name. Note that this constraint is for the example's sake, and neither necessary, nor advisable, as binding to a specific package name would needlessly impede reusability.

Another important construct in the GReQL-queries in Figure 4 are the *keySet*-function and *img_*-maps, used for instance in the GReQL query for the *systemname* attribute of the coordinate system (2). The *img_*-maps are provided by the transformation language GReTL to refer to already established mappings. Here, the query's domain of discourse ranges over all elements already mapped to the coordinate system by the previous query, i.e. packages named “com.vizsave”.

Generating the tool. The source meta-model, the visualization meta-model, and the mapping are used to generate a model transformation embodying the desired visualization. The target transformation language of the TGraph-based ELVIZ implementation is GReTL [HE11]. GReTL expressions rely on GReQL queries, allowing to directly use the queries specified in Figure 4 to transform models conforming to the source meta-model into models conforming to the visualization meta-model. The following example shows a GReTL rule to create a bar in the bar chart for each class contained in a specific package.

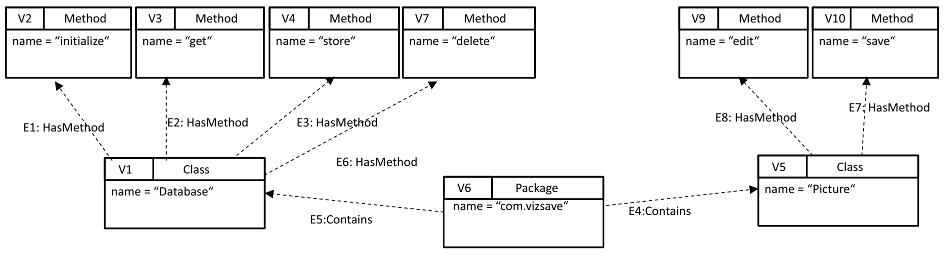
```
CreateVertexClass Bar <== from c: V{Class}, p: V{Package}
                                   with p-->{Contains}c and
                                      p.name="com.vizsave"
                                   report c end;
```

This directly corresponds to Query (6) in Figure 4, associating it with a concept of the visualization meta-model. On the left-hand side, the *CreateVertexClass* command of GReTL is used, to both create a new class in the visualization meta-model called *Bar*, and to create instances of this class for each element returned by the query on the right-hand side. This query ranges over *Classes* and *Packages*, selecting those pairs, where the class is contained in the package, and the package’s name is “com.vizsave”, and reporting the classes. Each of these classes thereby becomes an *archetype* for an instance of class *Bar*. Thus, the whole transformation can be generated automatically using information from the mapping, source meta-model, and visualization meta-model: Internally, the transformation uses the GReTL API, to dynamically construct the required transformation code as Java classes. GReTL can use existing target meta-models, but is also able to generate target meta-models (schemas) and conforming target models (graphs) at the same time. This means a visualization meta-model can be specified directly using GReTL. This will then also provide the framework for the generator which will be able to create visualization tools, parameterized by a set of queries providing the mapping to a source meta-model. To generate actual TGraphs for a specific source model, each visualization meta-model concept needs such a GReQL-query, which will be included dynamically.

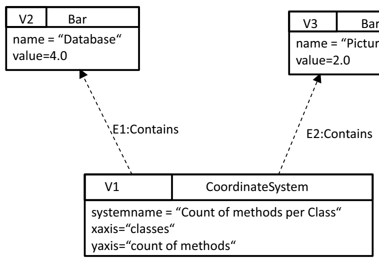
3.1.2 Executing a Visualization Tool

Transformations generated in the process described above essentially represent a visualization tool tailored to specific needs dictated by the source data and the task to be supported. This assumes that these transformations can integrate into an environment where data extraction and visualization renderer tools are already present. The advantage of using ELVIZ is that these tools have to be integrated only once, and then can be reused and recombined for further visualizations. The three steps described in the following (and depicted in Figure 1) can therefore be performed automatically by the generated visualization toolchain.

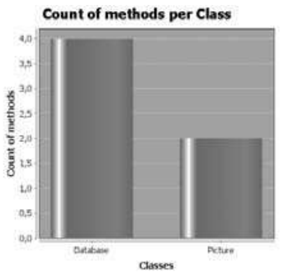
Extract. As a preprocessing step, a source model suitable for further processing by model transformations has to be extracted from the input data. In the example, the input is Java



(a) The source model as TGraph.



(b) The visualization model as TGraph.



(c) Generated bar chart.

Figure 5: Source model, visualization model, and graphical representation.

code (Figure 2). A parser is used to extract the required information and creates a model conforming to the source meta-model. ELVIZ assumes that such extractors are provided, or that source data is already available as models, together with appropriate meta-models they conform to. The model representation of the example is presented as TGraph in Figure 5a. In this figure, certain characteristics of TGraphs become obvious. TGraphs consist of nodes and directed edges. For example, the node *V6* represents the package "com.vizsave" of the example application. It has a type (*Package*) and an attribute (*name*). The edges represent links between different nodes. For instance, *V6* is connected with *V1* by an edge of type *Contains*. This expresses that the class *Database* is contained in the package "com.vizsave".

Transform. The generated transformation is executed automatically. Based on the provided queries, mapped to visualization meta-model concepts, it produces the model shown in Figure 5b. There is a single instance of *CoordinateSystem*, corresponding to the only Java package of the source model. It contains two bars, one for each class. The values of the bars (their height) is set to the number of methods defined in the corresponding classes, four and two, respectively.

Render. As last step, the visualization model is rendered. ELVIZ requires a renderer for each kind of visualization, i.e. for each visualization meta-model. A simple renderer for bar charts to create the actual graphical representation has been implemented using the JFreeChart library [Lim19]. Its output for the example program can be seen in Figure 5c.

4 Application of the ELVIZ-Approach

The example used in the previous section has been kept intentionally simple, to focus on the principal concepts of ELVIZ. ELVIZ has been employed to different fields of application, using more complex models. In this paper, three visualization scenarios from two different domains are presented. The first domain is the visualization of software metrics on a Java system, as in the previous example, however, an industry-scale Java meta-model was used. A small Android application called *GPSPrint* serves as input data, and metrics are visualized in two different ways: using bar charts (Section 4.1), and using *Code Cities* [WL07] (Section 4.2). The second application domain is that of energy-efficient applications. Here, the energy consumption of mobile applications, and its distribution across different components of a mobile device is visualized using pie charts (Section 4.3).

4.1 Visualizing Software Metrics using Bar Charts

The amount of methods per class has been visualized for the GPSPrint App using the same bar chart visualization meta-model as as presented previously, depicted on the right-hand side of Figure 6. As source meta-model, a Java meta-model, developed in the context of the SOAMIG project [FWE^{+}12] is used. It consists of 86 node types and 67 edge types. Figure 6 shows a small section of this meta-model on the left-hand side, containing those concepts relevant for the metrics to be evaluated and visualized here.

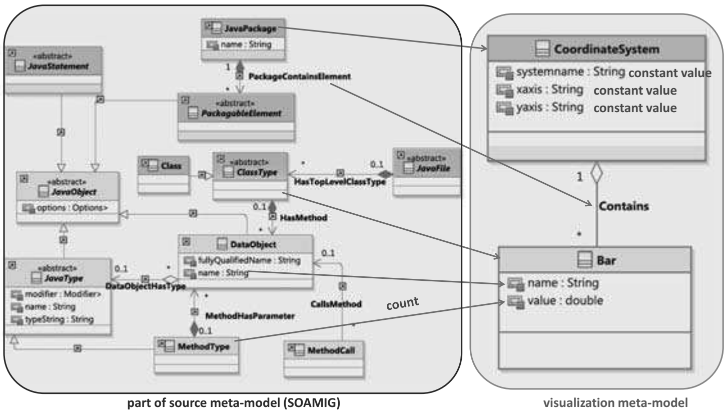


Figure 6: Source meta-model and mapping for the GPS Print Application.

The content of the desired bar chart should show the number of methods per class. Therefore, the mapping has to be specified as shown in Figure 6 – each bar represents one class of the GPSPrint Application, and the height of the bar is set by the number of methods of each class. The result of this visualization is shown in Figure 7: three classes are easily identifiable as containing considerably more methods than the average, *GPSItem*, *GPSPrint*, and *ViewItemList*.

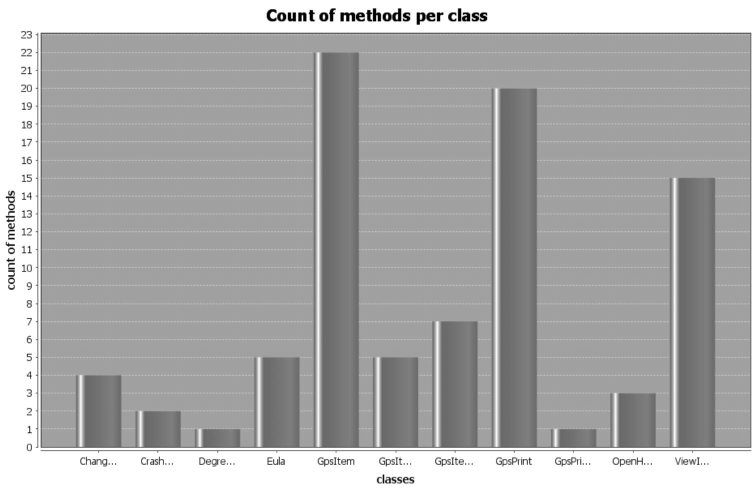


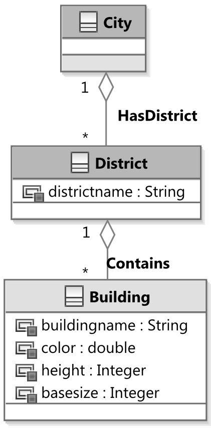
Figure 7: Bar chart of the number of methods per class for the GPSPrint Application.

4.2 Visualizing Software Metrics using Code Cities

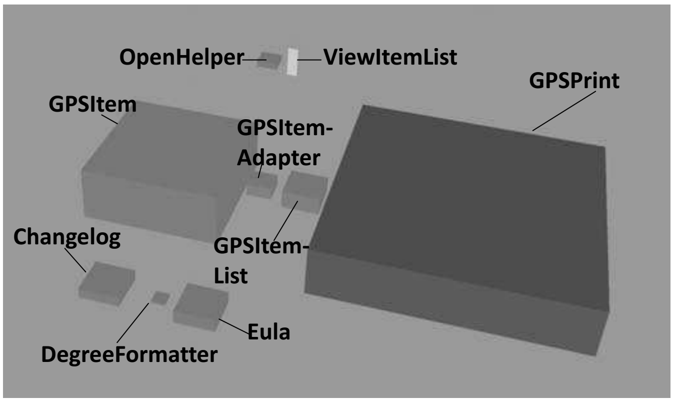
A very different kind of visualization, also realizable using ELVIZ, are *Code Cities*. To do this, an appropriate meta-model for this kind of visualization has been created, and is shown in Figure 8a. A *City* consists of *District* with a *districtname*. Districts contain *Buildings*, where each building is characterized by a name (*buildingname*), color (*colorIntensity*), *height* and the length of its square base area (*basesize*).

The mapping is based on the Code City semantics presented by Lanza et. al [WL07]: a city represents the whole code of an application (here GPSPrint). The districts represent the packages, and the buildings stand for classes in these packages. The name of the building is the name of the represented class. The color hue is set by the number of statements of this class. Here, a linear color gradient is aimed at, represented by a real number between zero and one. Alternatively, a set of possible colors could have been modeled using an enumeration. The height is set by the number of methods, and the base size by the number of attributes. The mapping contains corresponding queries, the most complex of which is the one to set the color hue. It is shown below as GReQL expression:

```
let maxStatements :=
   max(from m: keySet ( img_Building )
       report numStatements(m)
       end ) in
from m: keySet ( img_Building )
reportMap m -> (numStatements(m) / maxStatements)
end
```



(a) Meta-model for code cities.



(b) Code city of the GPSPrint Application with a simplistic renderer based on Java 3D (labels added in manual post-processing).

Figure 8: Code city meta-model and an actual code city rendering.

First, the *let* part of the query sets a variable *maxStatements* to the highest number of statements encountered in a class. Classes are mapped to buildings in this visualization, therefore the set of all classes can be obtained from the archetype of the mapping for buildings. This is done by applying the *keySet*-function to *img_Building*, which is a reference provided by GReTL to the already established buildings mapping. For each class, the number of statements is calculated; GReQL's *max*-function returns only the highest of those values. Then, the actual mapping to color values is specified, associating each building with a value between zero and one, by dividing the number of statements of the corresponding class by the maximum number of statements. A renderer maps this value to a continuous color gradient, e.g. from green, to yellow, to red, for low, medium, and high values.

To ease understanding, and re-use functionality, this query uses the helper function *numStatements* to actually calculate the number of statements. Such functions can be specified as part of the regular mappings, in a similar fashion, by associating a function name with a query. Internally, it will be passed to the GReTL transformation producing the visualization tool, making it available for use in all mapping queries. The query defining the helper function is shown below:

```
using class:
   count(class <--{frontend.java.HasTopLevelClassType}
   -->{^frontend.java.CallsMethod}*
   &{frontend.java.JavaStatement})
```

The *using* keyword is used to declare a parameter called *class*. The query assumes this to be a node of type *ClassType*. A regular path expression yields all statements contained in this class: first, an edge of type *HasTopLevelClassType* is traversed, leading to a *JavaFile* node. From here, all paths of arbitrary length (denoted by an asterisk), ending at a *JavaStatement* node, are considered, excluding those containing edges of type *CallsMethod*, which would lead out of the class.

This application demonstrates that ELVIZ is also able to generate visualizations which are very different from standard charts. The rendered code city image is shown in Figure 8b. A simple renderer using Java3D places boxes in a grid to represent the buildings with given base size and height, and uses the color attribute to paint them. A linear gradient starting at bright green (zero), moving to yellow, and ending at deep red (corresponding to a value of one). From the colors, it is immediately evident that most classes are small in their number of statements, as most buildings are colored in slightly different shades of green. There is one medium-sized class, *ViewItemList*, and the largest class, *GPSPrint*, which will always be assigned a deep red color, as the mappings define the highest statement count occuring to be the maximum number. Also, *GPSPrint* has a disproportionate number of attributes, visible by its large base size, whereas *ViewItemList* has almost no attributes, but above-average number of methods (height of the buildings). Due to the semantics chosen, some classes are not depicted at all, because their base size (attributes) or height (methods) is set to zero.

Using ELVIZ, this visualization model can be used with completely different input data, given appropriate mappings, or only the mappings can be adjusted to change the semantics of generated graphics. Also, other, more sophisticated renderers can be used, without having to change the visualization meta-model or any mappings.

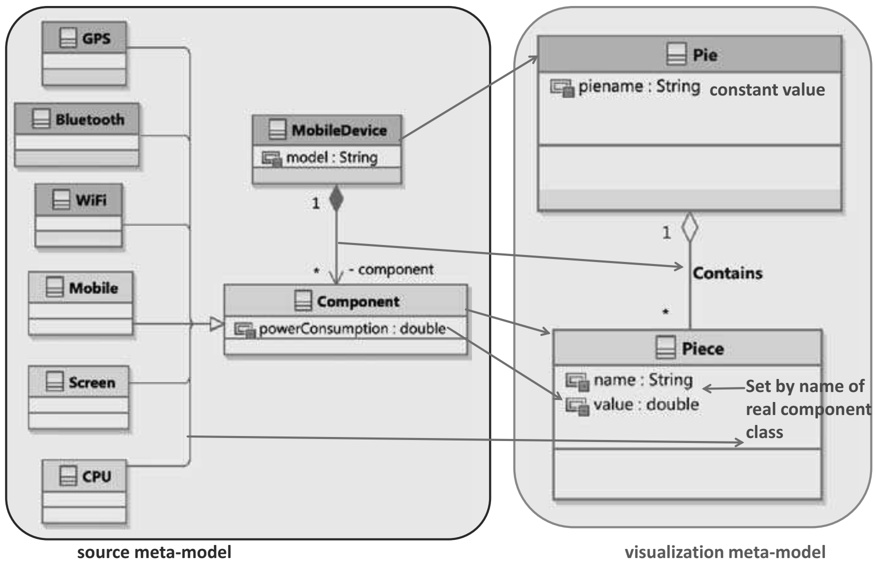


Figure 9: Source meta-model and mapping for GPSPrint concerning energy consumption.

4.3 Visualizing energy consumption using Pie Charts

To show that ELVIZ can not only be applied to the field of software visualization, ELVIZ is applied to another field in this section – to visualize the energy consumption of smartphone applications: A problem of today's smartphones is the limited battery time. This is not only a problem caused by the limited hardware possibilities, but also by the code of the applications [GJJW12]. It is possible to measure the energy consumption of mobile devices, and of applications running on them, using power profiles. A power profile provides mean values of energy consumption for each component of a smartphone. By monitoring the runtime of an application, and the state of the device's components during this time, the consumed energy for each component can be determined. This measurement method has been applied to the GPSPrint Application to get detailed information about its energy consumption. The results can be visualized using the ELVIZ approach, as well.

The source meta-model is shown in Figure 9 (left). A *MobileDevice* consists of different components, like *GPS*, *Bluetooth*, and *WiFi*. Each of these components have a power consumption. The desired kind of visualization is specified by a visualization meta-model: In case of visualizing the energy consumption on a mobile device per component, a pie chart is applicable. Pie charts show the relative distribution of the energy consumption per component. Thus, as visualization meta-model, a meta-model for pie charts is specified, shown on the right-hand side of Figure 9. A pie chart is modeled as a *Pie* with a *piename*, consisting of arbitrarily many *Pieces*, which also have a *name*, and carry a *value* representing its size in relation to the whole pie.

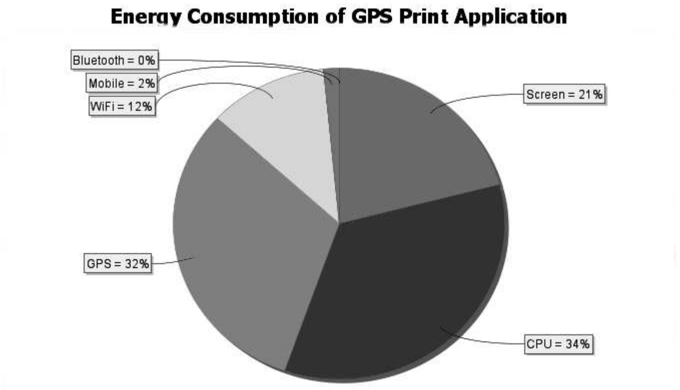


Figure 10: Bar chart for energy consumption of mobile components for GPSPrint.

As content of the visualization, it is desirable to have a pie chart where each piece represents one device component, and the size of the pie's pieces is set by the measured energy consumption of the corresponding component. The arrows between source and visualization meta-model in Figure 9 indicate a mapping according to these requirements.

The rendered result for this visualization, based on measurement data from the GPSPrint application running on an Android mobile phone, is shown in Figure 10. The renderer used to create this image has been implemented using JFreeChart.

5 Conclusion and Future Work

In this paper, the ELVIZ approach has been presented: It is a new approach for generating customized graphical representations of arbitrary source models by applying model-driven engineering and graph querying techniques to the field of model visualization. The ELVIZ approach is source-model-generic, and offers the possibility to create customized kinds of visualizations. A major benefit of the ELVIZ approach lies in managing mappings and visualization meta-models separately, enabling independent reuse: For example the same content e.g. count of methods per class of a Java system can be reused for different visualization kinds e.g. as pie or bar chart without any further effort.

An ELVIZ-based visualization tool is created in four steps: *1*) *Specifying the input* format using meta-models, or re-using an existing meta-model. Input data is assumed to be existent as models conforming to such a meta-model. Otherwise, an appropriate fact extractor needs to be provided. *2*) *Specifying the kind of visualization*, again, using a meta-model defining the abstract syntax, e.g. bars in a bar chart. *3*) *Specifying the content of the visualization* using mappings based on queries. Queries depend only on the source meta-model and can be re-used with different kinds of visualizations. *4*) *Generating the tool*, using as input a source meta-model, visualization meta-model, and a mapping. This is automated by ELVIZ, integrating those parts into a model transformation embodying the desired visualization. For each visualization meta-model, an appropriate renderer is required to provide actual graphical representations of visualization models.

The content can be defined separately from the kind of visualization by creating query-based mappings between elements of source meta-models and elements of visualization meta-models: Each element of a visualization meta-model has its counterpart in the source meta-model. The exact elements to represent are extracted out of the source model by specifying queries over the source model. Thereby, the specification of the *kind* of visualization, and the mapping – embodying the *content* of the visualization – are reusable for different source models and can be combined in different ways for different usage scenarios. Thus, the overall workload for creating a new graphical representation can be reduced. Another advantage of the ELVIZ approach is, that it is not only generic concerning source model, visualization kind, and visualization content, but also concerning to the implementation details of the approach. In this paper, one possible implementation, realizing the ELVIZ approach in the technological space of TGraphs has been presented. The conceptual architecture of ELVIZ can also be implemented using other technologies, such as using ATL and related MDA techniques. Thereby, the ELVIZ approach is able to remain useful for future projects, and to profit from future technologies in the field of model-driven engineering and querying.

The capability of ELVIZ to specify new kinds and contents for model visualizations, and to reuse and combine them, fits well with ongoing work on *software evolution services* [JOMW13], aimed at enhancing interoperability of software evolution tools, and easing their integration. In this area, ELVIZ fills the role of a universal visualization service, required by many software analysis and reverse engineering activities, where it can be integrated with other services to create customized toolchains. ELVIZ will be used as visualization of a service-based metrics tool currently being developed.

Literature

[Aub01] D. Auber. Tulip. In *Graph Drawing*. Springer, 2001.

[AvHK06] J. Abello, F. van Ham and N. Krishnan. ASK-GraphView: A Large Scale Graph Visualization System. IEEE Transactions on Visualization and Computer Graphics. 12(5), 2006.

[BSFL06] R. I. Bull, M. Storey, J.-M. Favre and M. Litoiu. An Architecture to Support Model Driven Software Visualization. In *14th IEEE International Conference on Program Comprehension*, ICPC, Washington, 2006. IEEE.

[Béz13] J. Bézivin. Models Everywhere, `http://modelseverywhere.wordpress.com` (19-09-2013).

[Ecl14] Eclipse. BIRT Project Description and Scope, `http://www.eclipse.org/birt/phoenix/project/description.php` (09-01-2014).

[EGK+01] J. Ellson, E.R. Gansner, E. Koutsofios, S.C. North and G. Woodhull. Graphviz - Open Source Graph Drawing Tools. In *9th International Symposium on Graph Drawing*, LNCS 2265. Springer, 2001.

[ERW08] J. Ebert, V. Riediger and A. Winter. Graph Technology in Reverse Engineering. The TGraph Approach. In *10th Workshop Software Reengineering.*, LNI 126, 2008.

[FWE+12] A. Fuhr, A. Winter, U. Erdmenger, T. Horn, U. Kaiser, V. Riediger and W. Teppe. Model-Driven Software Migration - Process Model, Tool Support and Application. In A. Ionita, M. Litoiu and G. Lewis, eds., *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, Hershey, PA, 2012. IGI Global.

[FW94] M. Fröhlich and M. Werner. Demonstration of the Interactive Graph-Visualization System da Vinci. In R. Tamassia und I. Tollis, eds., *Graph Drawing*, LNCS 894. Springer, 1994.

[GJJW12] M. Gottschalk, M. Josefiok, J. Jelschen and A. Winter. Removing Energy Code Smells with Reengineering Services. In U. Goltz et al., eds., *42. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, LNI 208. Köllen, Bonn 2012.

[HE11] T. Horn and J. Ebert. The GReTL Transformation Language. In J. Cabot and E. Visser, eds., *ICMT* LNCS 6707. Springer, 2011.

[JOMW13] J. Jelschen, M.-C. Ostendorp, J. Meier and A. Winter. A Description Model for Software Evolution Services. *1er Congreso Nacional de Ingeniería Informática / Sistemas de Información*, RIISIC, Cordoba, Argentina, 2013.

[JK06] F. Jouault and I. Kurtev. Transforming models with ATL. MoDELS'05, Berlin, Heidelberg, 2006. Springer.

[Ken02] S. Kent. Model Driven Engineering. In *Third International Conference on Integrated Formal Methods*, IFM, London, 2002. Springer.

[Lim19] Object Refinery Limited. JFreeChart, `http://www.jfree.org/index.html` (2013-09-19).

[OMG06] Object Management Group OMG. UML - Unified Modeling Language, `www.uml.org` (2013-09-06).

[Sol03] R. Soley. Richard Mark Soley. Model Driven Architecture: The Evolution of Object-Oriented Systems? In D. Konstantas et. al, eds., *OOIS* 2817 of *LNCS*. Springer, 2003.

[WL07] R. Wettel and M. Lanza. Visualizing Software Systems as Cities In *Visualizing Software for Understanding and Analysis.*, IEEE, 2007.

[WW05] J. Wolff and A. Winter. Blickwinkelgesteuerte Transformation von Bauhaus-Graphen nach UML. *Softwaretechnik-Trends*, 25(2), 2005.