Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Graphical Editors for Defining Scaling Policies Analysable Using Simulations

Tim Summerer

Course of Study: Informatik

Examiner: Prof. Dr.-Ing. Steffen Becker

Supervisor: Floriment Klinaku, M.Sc.

Commenced: December 1, 2021

Completed: June 1, 2022

Abstract

Context. This thesis is concerned with improving the engineering of auto-scaling policies for cloud based applications through a model-based approach. Throughout this paper I create a graphical editor for the scaling policies introduced by Klinaku et al. [KHB21].
Problem. Working with scaling policies is currently done with a tree-based editor. These can be a problem either for software architects that are used to graphical modeling languages such as UML, for whom a tree editor might make their work more tedious or e.g., for communicating scaling concerns to stakeholders such as financial managers or clients because understanding technical terms via a tree editor can be especially difficult for them.
Objective. The objective is to design and implement a graphical editor for scaling policies that makes the creation of policies easier and to improve the understanding of scaling policies as part of the research question of this thesis.
Method. To implement the graphical editor, I design a model for it based on state-of-the-art research on visual notations. To refine the model, I gather feedback from three experts of software quality and architectures. I have implemented the graphical editor in Eclipse Sirius.
Result. For validation, I perform an evaluation session where participants from the industry and academia have been asked to give feedback via a questionnaire on their experience using the graphical editor. Almost all participants have found the design to be appropriate and two thirds of participants have reported a high value for helpfulness of the graphical editor.
Conclusion. Lastly, I summarize key aspects of the thesis, discuss benefits and limitations to the graphical editor and my findings. Additionally, I present the lessons I learned and point out potential future work.

Kurzfassung

Context. Diese Thesis befasst sich mit der Verbesserung der Entwicklung von Auto-Scaling Richtlinien für Cloudanwendungen mittels eines modellgetriebenen Ansatzes. In dieser Arbeit erschaffe ich einen grafischen Editor für die "Scaling Policies", die von Klinaku et al. [KHB21] eingeführt wurden.
Problem. Das Arbeiten mit "Scaling Policies" erfolgt derzeit mit einem Baum-Editor. Dieser kann ein Problem für Softwarearchitekten darstellen, die es gewohnt sind mit grafischen Modellierungssprachen wie UML zu arbeiten, für die ein Baum-Editor ihre Arbeit erschwert oder z.B., für die Kommunikation von Skalierungsanliegen mit Stakeholdern wie Finanzmanagern oder Klienten für die das Verstehen von Technikbegriffen mittels einem Baum-Editor besonders schwierig sein kann.
Objective. Das Ziel ist es einen grafischen Editor für "Scaling Policies" zu entwerfen und zu entwickeln, welcher dabei hilft, die Erstellung von "Scaling Policies" zu erleichtern und das Verständnis von "Scaling Policies", als Teil der Forschungsfrage, zu verbessern.
Method. Um den grafischen Editor zu implementieren, entwerfe ich ein grafisches Modell basierend auf dem Stand der Wissenschaft für grafische Darstellung. Um das Modell zu verbessern, sammle ich Feedback von drei Experten für Softwarequalität und -architekturen. Ich habe den grafischen Editor in Eclipse Sirius implementiert.
Result. Zur Validierung habe ich eine Evaluationssitzung durchgeführt, bei der Teilnehmer aus Industrie und Forschung, die Gelegenheit hatten Rückmeldung mittels einer Umfrage zu ihrer Erfahrung mit dem grafischen Editor zu geben. Fast alle Teilnehmer befanden das Design als passend und zwei Drittel der Teilnehmer gaben einen hohen Wert an Nützlichkeit des grafischen Editors an.
Conclusion. Abschließend fasse ich die Schlüsselaspekte der Thesis zusammen, diskutiere Vorteile und Limitierungen des grafischen Editors und meiner Erkenntnisse. Zusätzlich präsentiere ich gelernte Erkenntnisse und zeige potentielle zukünftige Arbeitsschritte auf.

Contents

1 Introduction **1**

1.1 Thesis Overview 1
1.2 Contributions 2
1.3 Thesis Structure 2

2 Foundations and Related Work **3**

2.1 Foundations 3
2.2 Related Work 7

3 Design **13**

3.1 Requirements 13
3.2 First Draft 13
3.3 Feedback on the First Draft 15
3.4 Second Draft 16

4 Implementation **19**

4.1 Foundations 19
4.2 Creating the Graphical Editor 21
4.3 Discussion 29
4.4 Overview of the Graphical Editor 32

5 Evaluation **39**

5.1 Study Design 39
5.2 Results 42
5.3 Discussion 44
5.4 Threats to Validity 45
5.5 Resulting Implementation Adjustments 46

6 Conclusion **47**

6.1 Summary 47
6.2 Benefits 48
6.3 Limitations 48
6.4 Lessons Learned 49
6.5 Future Work 49

Bibliography **51**

A Appendix **53**

A.1 Summary of Study Results 53
A.2 Screenshots 63

List of Figures

2.1 An Infrastructure model in Argon 8
2.2 An Overview diagram in AjiL 9
2.3 A Snapshot of the Design Window of Stratus ML 11

3.1 First draft of the graphical editor 14
3.2 Second draft of the graphical editor 17

4.1 Empty Scaling Policy 21
4.2 Scaling Policy with Trigger added 22
4.3 Scaling Policy with Adjustment Type 22
4.4 Scaling Policy with Constraints added 23
4.5 Scaling Policy with a Target Group added 24
4.6 Palette of the graphical editor 26
4.7 Excerpt of the Ecore meta model 28
4.8 Architecture diagram 30
4.9 Example SPD 33
4.10 Example with constraint 1 fulfilled 34
4.11 Example with constraints 1-2 fulfilled 34
4.12 Example with constraints 1-3 fulfilled 35
4.13 Example with constraints 1-4 fulfilled 36
4.14 Example with constraints 1-5 fulfilled 37
4.15 Example with all six constraints fulfilled 38

5.1 Given example for the evaluation study (latest graphical model) 40

A.1 Given example for the evaluation study (as provided to participants) 63

1 Introduction

1.1 Thesis Overview

The thesis is concerned with improving the engineering of auto-scaling policies for cloud applications through a model-based approach and therefore, to make these scaling policies more robust and to potentially reduce costs. Throughout this paper I create a graphical editor for the scaling policies introduced by Klinaku et al. [KHB21].

For doing that, I introduce the most relevant topics in this context, namely, service oriented architecture, microservices, cloud native applications and, of course, scaling policies. In addition, best practices for visual notations are also listed. Furthermore, with regard to related work, I present Argon, a graphical modeling tool for cloud provisioning. Next, AjiL, a graphical modeling toolkit for microservice development is presented. Lastly, Stratus ML, a modeling framework for cloud applications is discussed.

The problem I address is that, for now, scaling policies are edited using a standard, tree-based editor. These tree editors can, for instance, be tedious to use for software architects that are used to using graphical modeling languages, like Unified Modeling Language (UML), to express complex software architectures. Furthermore, a tree-style representation can also be hard to understand for less technically inclined business stakeholders such as financial managers and clients. Understanding cloud scaling concerns simply by looking at a tree of technical terms can be a problem e.g., when convincing other stakeholders of the relevance to address scaling needs in a software system.

In order to make working with scaling policies easier, I design and implement a graphical editor for scaling policies. More concretely, the graphical editor helps with the creation and understanding of scaling policies.

To achieve this, I first design a model for the graphical editor based on state-of-the-art research on visual notations to then implement the editor based on that model. During the design phase, I gather feedback from three experts of software quality and architectures to then refine the model with feedback taken into consideration. Furthermore, the graphical editor is implemented in Eclipse Sirius.

To validate the prototype, I have performed an evaluation session with participants from academia and the industry. There, I have gathered feedback on the helpfulness and design of the graphical editor and the participants' experience. Moreover, participants have been asked to give their input via a questionnaire on potential applications of such a graphical editor for scaling policies. Out of nine participants, at least eight have found the visual representations appropriate. With regard to the helpfulness of the editor, six out of nine have tended towards a strong helpfulness of the graphical editor.

In addition, I discuss who can benefit from the graphical editor and what the graphical editor and empirical results are limited to. Moreover, I point out potential future work and go over the lessons I learned throughout the course of this bachelor thesis.

1.2 Contributions

The contribution of this thesis is the development of a graphical editor for scaling policies based on state-of-the-art best practices for visual notations. This contribution is reinforced by the performed evaluation study on the design and helpfulness of the graphical editor. Throughout the thesis I address the following research question: *How graphical editors can aid the software architect to create and understand scaling policies better?*

1.3 Thesis Structure

Chapter 2 – Foundations and Related Work: First, I provide an overview of the research areas the thesis is focused on. Also, I introduce scaling policies and give insight into best practices for visual notations.

Chapter 3 – Design: Here, I go over the design and evolution of the model for the graphical editor.

Chapter 4 – Implementation: The creation of the graphical editor is discussed step by step in this chapter. Afterwards, I provide an overview of the graphical editor and an outlook beyond the current implementation.

Chapter 5 – Evaluation: In this chapter, I discuss the evaluation of the graphical editor. For that, I have conducted an evaluation session where participants from academia and the industry were able to try the graphical editor and provide their feedback.

Chapter 6 – Conclusion I conclude by summarizing the thesis and also pass in review benefits, limitations, lessons learned and potential future work.

2 Foundations and Related Work

2.1 Foundations

In the following, I introduce the relevant areas of research around scaling policies i.e., the service oriented architecture (SOA) style, microservices and cloud native applications (CNAs). In addition, I introduce scaling policies and the problem that I am addressing with this thesis. Finally, I go over best practices for visual notations.

2.1.1 Problem

The problem that I address throughout this thesis is the lack of a graphical editor for scaling policies. In order to address the issue, I am concerned with answering the research question as to *how graphical editors can help the software architect to create and understand scaling policies better*. To begin, I now discuss the relevant realms of software architecture.

2.1.2 Literature Research Methodology

To find literature, I primarily utilized the Google Scholar[1] search engine. The searches were restricted to the field of service oriented architecture, microservices and cloud native applications. Therefore one of these terms was included in the search terms. In addition, the keywords graphical editor or graphical model were added when applicable. To tune the search towards software architecture and modeling, I added these terms.

Of course, doing follow-up searches by author on high quality papers e.g. to find a later performed study was done. To select the most relevant scientific papers, I established the following four requirements.

Requirements

- P1 The graphical editor/model *must* be used for modeling or software architecture purposes
- P2 A clear connection to SOA/microservices/cloud native applications *must* exist
- P3 Criteria or goals for the editor/model *should* be established
- P4 An evaluation of the editor/model *should* be available

[1] https://scholar.google.com/ last access 18.2.2022

In addition, to corroborate claims I make throughout this thesis, I utilized Google Scholar to search for scientific papers underpinning the respective statements independent of the above-mentioned criteria and search terms.

2.1.3 Service Oriented Architecture (SOA)

The term service oriented architecture can be split into two parts, the service orientation and the architecture. On the one hand, there is the service orientation which means that in this approach to software engineering, services are used to provide the desired functionality. From a technical standpoint, services themselves can be seen as providers of an interface for consumers [PL03]. From a business perspective, services can be seen as units of transaction that are described in a contract and fulfilled by the business infrastructure [PL03].

On the other hand, the architecture can be interpreted as the way technical modules are arranged [PL03]. Architecture can also be seen as the approach to develop and deploy functional units systematically [PL03].

To combine the two terms again, SOA is structuring the technical infrastructure so that it provides the business services in a way that can be restructured in the future to provide new or upgraded business services [PL03].

2.1.4 Microservices

In the paper "Towards an Understanding of Microservices" [SRH17], Shadija et al. discuss key characteristics of microservices.

First, it has to be mentioned that microservice architectures are a subset of SOA that is influenced by Domain Driven Design (DDD) [SRH17]. There are several definitions for microservices. In the following, I summarize the key aspects that can be drawn from the definitions.

The definition of Dragoni et al. states that microservices "should be highly cohesive units" [SRH17], and "they should do one thing well" [SRH17]. In addition, the ability to be deployed independently is also needed [SRH17]. Furthermore, a microservice architecture should address a single business functionality with a clearly defined interface [SRH17]. According to the definition of Cockroft, the focus of microservice architecture is on the business capabilities [SRH17].

In addition, Newman suggests "the importance of service autonomy" [SRH17]. Moreover, Thones definition implies that microservices "should be self-reliant, flexible and fault-tolerant [...] [and] should have a single responsibility" [SRH17].

Overall, microservices are services that fulfill one singular business functionality such as handling financial transactions and are supposed to perform this one function extremely well without the reliance on other potential services.

2.1.5 Cloud native applications (CNA)

In 2017 Kratzke and Quint performed an extensive literature study to better understand cloud native applications (CNAs). Based on literature findings, they propose the following definition of a CNA. "A cloud-native application (CNA) is a distributed, elastic and horizontal scalable system composed of (micro)services which isolates state in a minimum of stateful components. The application and each self-contained deployment unit of that application is designed according to cloud-focused design patterns and operated on a self-service elastic platform." [KQ17]

By this definition, a CNA can adapt to varying workloads by automatically changing the available resources to the needed demand (cf. Herbst et al. [KQ17]). The CNA scales horizontally by e.g. spawning more containers as needed when the load increases. Size changes in the CNA have been considered at design time and therefore do not need architectural changes to the CNA (cf. Bondi [KQ17]).

In addition, the use of standard containers for "self-contained deployment" is common. The containers allow the encapsulation of a software component and its dependencies in a self-describing and portable way in order to be used in different environments (i.e., a variety of machines) [KQ17]. The elastic platform that CNAs operate on is simply the means of execution (e.g., externally hosted virtual machines with container capabilities) while also providing communication and data storage capabilities (cf. Fehling et al. [KQ17]).

2.1.6 Scaling Policies

To manage the varying demand of these applications, so called, scaling policies can be used. Going one step further, it is possible to evaluate if the created scaling policies fulfill their purposes as expected [KHB21]. This is achieved by using terminating simulations.

Since the graphical editor will be an editor for these scaling policies, it is important to understand the concepts of the metamodel for scaling policies. The core concepts of the metamodel are as follows.

Scaling Policies are encapsulated in Scaling Policy Definition (SPD) files. The SPD files then constitute the artifact for evaluation purposes. Furthermore, a scaling policy can be set active or inactive.

Every policy has a ScalingTrigger that specifies the property to monitor, how to monitor it and a threshold for its value. ScalingTriggers can be of the following types: CPU utilization, RAM utilization, HDD utilization, network utilization, a point in time, an amount of time elapsed, elapsed idle time, the amount of running tasks, response time or even resource utilization based triggers.

When this trigger fires, the TargetGroup will be adjusted. The TargetGroup represents the physical or virtual environment that is being scaled. Furthermore, the AdjustmentType specifies how the TargetGroup is adjusted. An AdjustmentType can be relative i.e., a percentage based adjustment based on the previous value, it can be absolute i.e., if the trigger fires the value will be set to a predefined setting, or it can be a step adjustment i.e., an incremental adjustment that increases the value by a set amount.

Finally, policies can be configured with PolicyConstraints that can specify constraints for analysis purposes. The available constraints are a group size constraint for the target group, an interval constraint to limit the policy to a specific time interval, a cool-down constraint to enforce a minimum time between scaling adjustments of the policy and a thrashing constraint to specify a minimum time between scaling in and out or vice versa.

2.1.7 Best Practices for Visual Notations

The way graphical conventions are decided is “generally shrouded in mystery” [Hit02; Moo09]. One possible explanation might be that mathematically-thinking people see graphical notations as informal. However, the form of presentation is at least as important as the content [LS87; Moo09].

According to Moody [Moo09], a good graphical editor needs to have a *design goal*. In addition, the *design rationale* is important, therefore design decisions and their reason must be documented.

Regarding the visual representation of a model, there should be a one-to-one mapping between the model and its (graphical) notation to maximize precision and expressiveness of the language.

Another concern to address for a good graphical editor is *perceptual discriminability* which means how easy the graphical symbols are to discern. This can be quantified by the difference in visual variables between the individual elements of the graphical notation.

Visual variables are e.g. the horizontal or vertical position, shape, size, color, brightness, orientation and texture [Moo09]. Using these variables one defines a primary notation of the graphical language that specifies it formally. In addition, the secondary notation which is not formally specified is used to clarify the meaning of the primary notation.

To improve perceptual discriminability one should increase the *visual distance* i.e., the amount of visual variable differences of the elements in the notation. The larger the visual difference, the faster and more accurate it is for the human mind to recognize the elements [Moo09].

The shape of the elements is the primary basis for the human brain to discern objects, which makes it an especially important visual variable. Furthermore, redundant coding i.e., differentiating elements by having multiple discerning visual variables can aid perceptual discriminability. Elements can “pop out” if one of their visual variables is unique among all elements. If visual elements only differ in text, they have a visual distance of zero. Nonetheless, text can be effective to differentiate between instances of the same visual elements [Moo09].

Elements are *semantically transparent* when their meaning can be inferred by their appearance. *Icons* are symbols that perceptually resemble the concepts they represent e.g. a lock icon represents the act of locking something.

Moody [Moo09] states that complex diagrams need modularization or a form of hierarchical structuring. If a diagram consists of 7±2 “bubbles”, that is acceptable and in line with the limitations of the human brain.

Color is the most cognitively effective visual variable as it is detected three times faster than shape [Loh93]. Despite this, one should never differentiate visual elements only by color because color can be perceived differently by different people e.g., those with color blindness [Moo09].

In addition, including text with other visual variables is more effective than to have text included separately [MM03]. The "*span of absolute judgment*" i.e., the amount of visual elements the human mind can discern at the same time is roughly seven [Mil56]. The decision to not show some constructs graphically is the "general purpose tool of last resort" according to Moody who cites Oberlander [Moo09; Obe95].

Finally, different audiences might require different visual dialects [Pet95]. It is proven that experts and novices process diagrams very differently [LMO95; Moo09].

2.2 Related Work

In the following, I discuss the most relevant graphical editors for modeling and software architecture in the realms of service oriented architecture (SOA), microservices and cloud native applications (CNAs) to get an impression as to how graphical editors currently aid software architects. To select the most relevant scientific papers, I established four requirements which I discussed in Section 2.1.2.

2.2.1 Argon

Argon is a graphical modeling tool to specify the provisioning of cloud resources [SIA17]. It has a fixed metamodel for the infrastructure which is specified in a domain specific language (DSL). The metamodel is specific to Amazon AWS and Microsoft Azure, but it is not limited to these cloud service providers.

Argon aims to remedy the need to learn different scripting languages and instead focuses on the modeling of the cloud resources. The metamodel uses Eclipse Ecore and Eclipse Modeling Framework. For the graphical model EuGENia [2] was used. EuGENia is a tool which aids in creating Graphical Modeling Framework[3] (GMF) based editors.

Argon uses a layered model to bridge the gap between the graphical notation and the usable code for the cloud service providers. The graphical model fulfills the role of the Platform Independent Model (PIM) which describes the structure and behavior independent of the target platform. The Platform Specific Model (PSM) contains the information about the behavior and structure of the target platform. Transformations that can be model-to-model and model-to-text are used to then generate a usable model for the cloud service provider. In Figure 2.1 a graphical infrastructure model in Argon can be seen.

While Argon is a graphical editor to model cloud resources, the graphical editor for scaling policies is solely focused on the scaling of resources by using scaling policies. The scaling of cloud resources in Argon is part of their model but only in a limited fashion i.e., one singular part of the graphical model. Here, scaling policies and therefore my editor go a step further by diving deeper into the scaling of resources.

[2] https://www.eclipse.org/epsilon/doc/eugenia/ last access 20.12.2021
[3] https://www.eclipse.org/modeling/gmp/ last access 18.02.2022

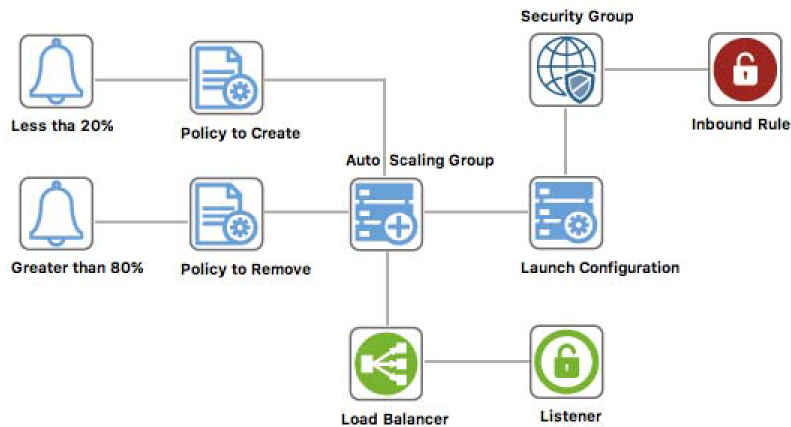


Figure 2.1: An Infrastructure model in Argon

Source: J. Sandobalin, E. Insfran, S. Abrahao. "An infrastructure modelling tool for cloud provisioning". In: 2017 IEEE International Conference on Services Computing (SCC). IEEE. 2017, p. 359, Fig. 9

Later Sandobalin et al. performed a study where they compared the code based infrastructure as code (IaC) tool Ansible to the graphical IaC tool Argon [SIA20]. In the study they assessed the quantitative properties effectiveness and efficiency. On the one hand, for effectiveness eight requirements were given and the number of fulfilled requirements was counted. On the other hand, for efficiency the number of fulfilled requirements per time was measured [SIA20].

Additionally, three perception based properties have been assessed using questionnaires with a 5-point scale. Perceived ease of use, perceived usefulness and intention to use (again) are the measured qualitative properties [SIA20].

To measure the properties, three identical experiments were performed in which roughly 20 students, per experiment, of undergraduate (i.e., bachelor's) or master's computer science students participated.

The results of the study showed that Argon is more effective than Ansible. Furthermore, hypothesis testing confirmed that Argon yielded more correctness [SIA20] with regard to the participants created models. Better efficiency of Argon versus Ansible was also confirmed by hypothesis tests [SIA20]. In addition, questionnaires and hypothesis tests signified that Argon is perceived as easier to use than Ansible [SIA20]. Participants also perceived Argon as more useful than Ansible [SIA20]. Finally, with regard to intention to use again, the participants stated a greater intention to use (again) for Argon than Ansible. This has been backed by a hypothesis test and the questionnaires [SIA20].

According to Sandobalin et al., the results are representative for undergraduate and master students of computer science and related study programs [SIA20]. However, it is mentioned that the study should be repeated with more experienced industry experts [SIA20]. Going a step further, performing a study with more complicated tasks to verify the measured properties also in regard to more complex infrastructures is still open for verification [SIA20].

2.2.2 AjiL

AjiL is a graphical toolkit for model-driven microservice development [RSW+20]. To facilitate the graphical development, it uses an Eclipse based editor. More precisely, it is built upon Eclipse Modeling Framework (EMF)[4] for the model, Sirius[5] for the graphical model and Acceleo[6] for code generation [SWR+18]. Also, constraint validation is supported [SWR+18]. Furthermore, AjiL targets Java and Spring cloud based microservices [SWR+18].

Two types of box-and-line diagrams are used, namely an overview perspective and a detailed view of single microservices. On the one hand, the overview perspective uses colored cubes to represent a microservice, interfaces are shown as circles and edges with textual annotations are used for interactions. An example of the overview perspective can be seen in Figure 2.2. On the other hand, the detailed view gives an overview for each microservice and its integral components [RSW+20].

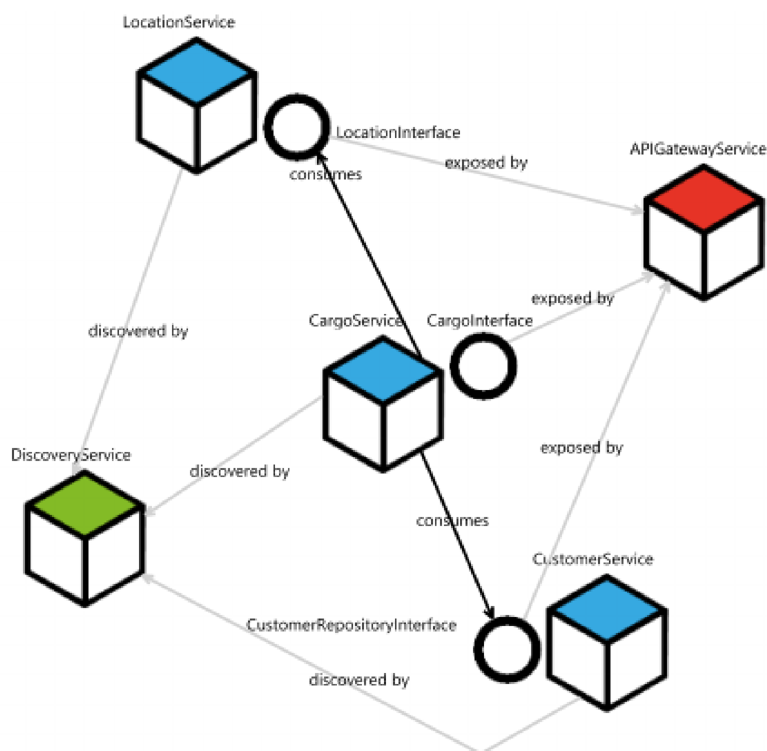


Figure 2.2: An Overview diagram in AjiL

Source: J. Sorgalla, P. Wizenty, F. Rademacher, S. Sachweh, A. Zündorf. "AjiL: enabling model-driven microservice development". In: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings. 2018, p. 3, Fig. 2

[4] https://www.eclipse.org/modeling/emf/ last access 18.01.2022
[5] https://www.eclipse.org/sirius/ last access 18.01.2022
[6] https://www.eclipse.org/acceleo/ last access 18.01.2022

According to Sorgalla et al. AjiL "aims to ease the effort of redundantly and cumbersome handcrafting of implementation in the development process of MSAs leveraging Model-driven Engineering (MDE)" [SWR+18]. Furthermore, it "enables developers to create diagrams of their intended microservice landscape and is able to generate preconfigured system foundations based on the diagram's information" [SWR+18].

One goal for AjiL is to teach microservice architectures to students by providing a graphical notation for relevant concepts. In addition, the code generation allows to understand the relationship between the model and the generated artifact [RSW+20].

However, using graphical notations needs a lot of screen space [RSW+20]. As a result, this makes working with complex architectures in AjiL difficult and slow [GBU08; RSW+20]. Another disadvantage is that the high degree of abstraction limits the expressiveness of AjiL [RSW+20]. The authors also mention that the implementation "lacks heterogeneity in the realization" [RSW+20] i.e., it is not as technologically agnostic as desired. It also lacks stakeholder specific views [RSW+20].

Overall, AjiL is concerned with the modeling of microservice based systems and it does so by being a graphical editor with boxes and lines. This is similar to my editor in that I also provide graphical modeling capability with box-and-line diagrams. Contrary to AjiL, my editor is concerned with scaling policies. These can be used with a microservice architecture but in itself these are two different perspectives of software architecture.

The goals of AjiL are to ease implementing microservice architectures. Again, this is similar to my goal of easing the development of scaling policies. However, scaling policies are concerned with *scaling* while designing a microservice architecture is a task on a different level of abstraction.

2.2.3 Stratus ML

Stratus ML is a "technology agnostic [...] modeling framework for cloud applications" [HT15]. It offers a multi-layer view for different stakeholder perspectives. Namely, the developers, providers, administrators and financial managers [HT15]. The implementation of the layer views is custom while Stratus ML is built on Microsoft Visual Studio 2012 and the Microsoft DSL (Domain Specific Language) toolkit.

Similar to scaling policies, Stratus ML allows visual modeling of, so called, adaptation rules and actions which scale the respective cloud application. It also supports grouping components for adaptation, very similar to the target groups in scaling policies. Furthermore, Stratus ML adheres to the Model-View-Controller (MVC) pattern [HT15]. This means that when a graphical model in Stratus ML is changed, Stratus ML performs the necessary steps for validation, transformation and analysis so that the view is consistent with the changed model [HT15].

The metamodel of Stratus ML supports five concerns of cloud applications: the service composition of the app, the availability, the adaptation, the provider and the performance [HT15]. Overall, Stratus ML is a framework that aids with modeling cloud applications on different levels of abstraction e.g., the service composition and also considers stakeholder perspectives such as the perspective of an administrators view on the model. In Figure 2.3 a snapshot of the design windows of Stratus ML can be seen.

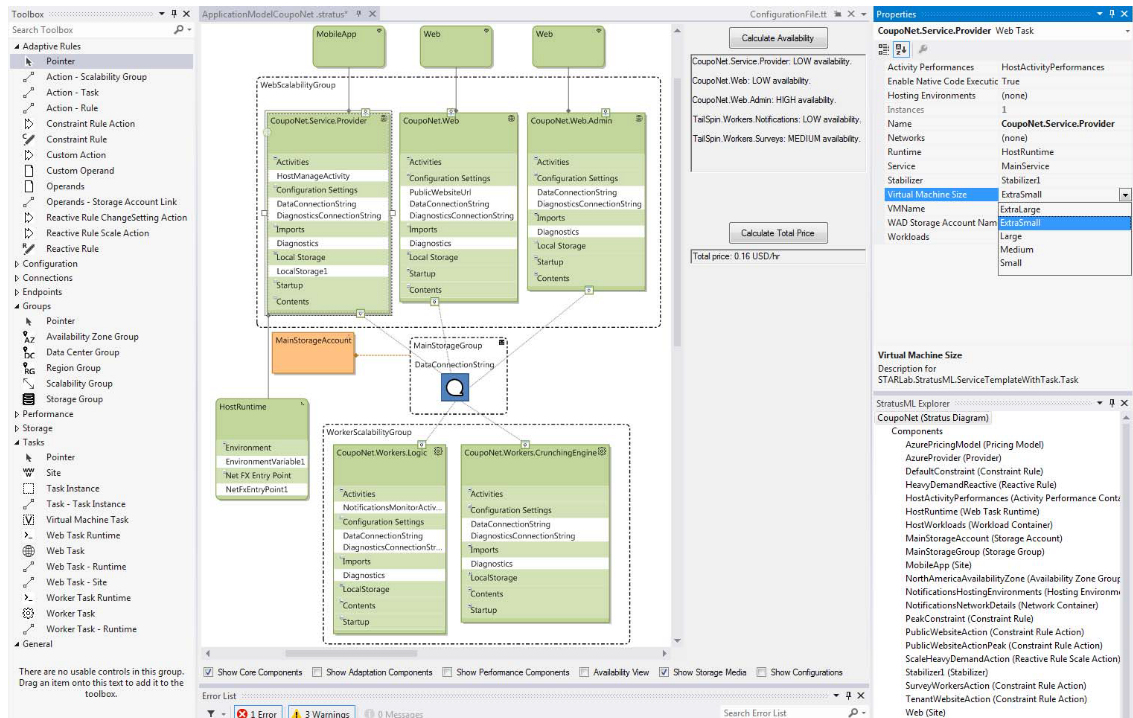


Figure 2.3: A Snapshot of the Design Window of Stratus ML

Source: M. Hamdaqa, L. Tahvildari. "Stratus ML: A layered cloud modeling framework". In: 2015 IEEE International Conference on Cloud Engineering. IEEE. 2015, p. 100, Fig. 3

Fundamentally, Stratus ML's adaptation rules are very similar to scaling policies as they work with rules, actions, constraints and groups. However, Stratus ML is a general purpose modeling framework for cloud applications whereas my editor for scaling policies is a tool designed to do one thing well: *to help in the creation and understanding of scaling policies*. As a result, the graphical editor for scaling policies separates the concern and complexity of *scaling* whereas Stratus ML does not. This is a possible advantage, that Stratus ML cannot provide.

3 Design

3.1 Requirements

Below, the requirements for the design of the graphical editor are listed, based on the previously performed literature survey.

REQ1 The graphical editor must have a design goal [Moo09].

REQ2 There has to be a one-to-one mapping between the metamodel and the graphical model [Moo09]. However, this does not mean that the entire metamodel has to be represented graphically.

REQ3 The elements of the graphical model should have a high visual distance between them, to make them easily discernible [Moo09].

REQ4 Instances of the same element of the graphical model should be differentiated by text [Moo09].

REQ5 Icons or symbols that resemble the meaning of the graphical element should be used [Moo09].

REQ6 A diagram should not contain more than five to nine elements [Moo09] because the "span of judgment" of the human brain is roughly seven [Mil56].

3.2 First Draft

The design goal (REQ1) for the editor is to make scaling policies easier to work with and to help the user understand scaling policies better. This goal is directly based on the addressed research question i.e., how to graphical editors can help the software architect to create and understand scaling policies better.

In Figure 3.1 the first draft of the graphical scaling policy editor is shown. In addition, you can see two policies that scale a single target group. The policy on the left scales out by increasing the number of elements in the target group by 10 if there are more than 10 tasks currently. The policy on the right scales in. If the CPU usage is below a threshold of 50%, then the number of elements of the target group get reduced by 50%. For both policies there is a constraint that the number of elements of the target group must always stay between 10 and 50.

Scaling policies are displayed as large, red rectangles to highlight them as core elements. They are shown with the name of the scaling policy above the rectangle and the diagonal arrows symbolize the scaling aspect. Triggers are represented by smaller, yellow squares as they are dependent on the policy. In addition, next to the square the threshold and direction are shown. A representative icon based on the type of trigger is displayed in the middle of the square. Adjustment types are

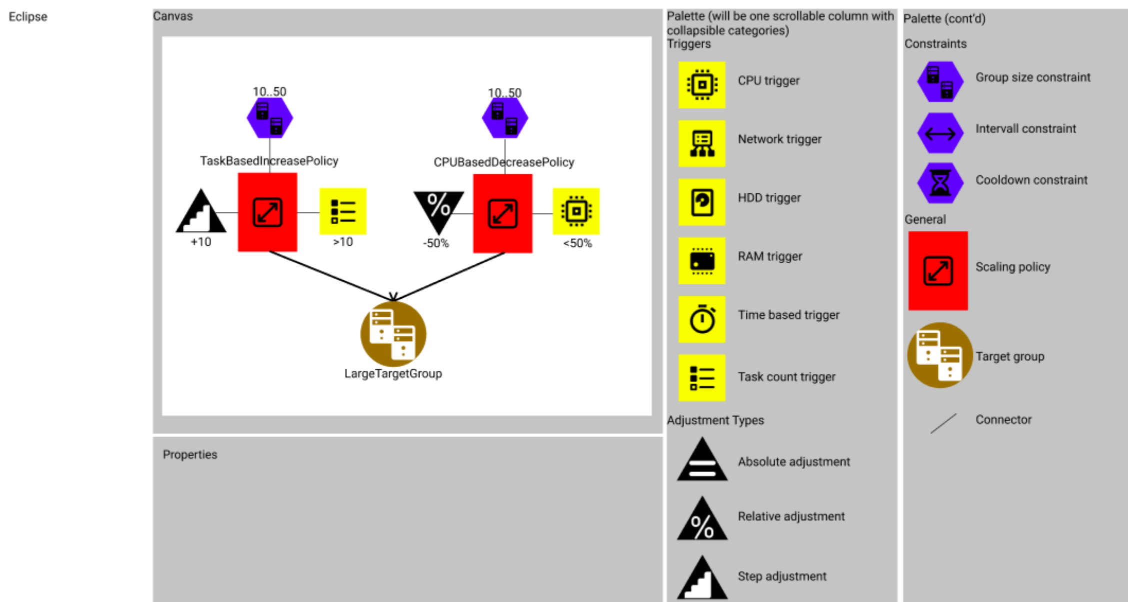


Figure 3.1: First draft of the graphical editor showing two policies to scale in and out

represented by black triangles and an icon to symbolize the type of adjustment. The triangle is pointing upwards if the adjustment scales out, and it points downwards if the adjustment scales in. The adjustment value and direction is also shown as text next to the triangle. Target groups are shown as large (because it is a more independent component), brown circles with their name next to it. Arrows point from a scaling policy to a target group if the scaling policy monitors and scales the target group. Policy constraints are shown as small, blue hexagons with an icon inside to resemble the type of constraint. Text next to the hexagon shows the basic setting for the respective type of constraint e.g., for a cool-down constraint the duration is shown.

Other properties of the elements are put into the respective properties panel below the canvas. This results in a one-to-one mapping between Ecore metamodel and graphical metamodel to fulfill REQ2. The colors have been chosen with color vision deficiencies in mind. Green is not used because it can be difficult to tell apart from several colors such as red, brown, blue, yellow et cetera [Ayt+17]. Combinations of blue, purple and gray have also not been used for the same reason [Ayt+17].

3.2.1 Design Decisions of the First Draft

Different shapes, sizes, colors and icons[1] are used for different metamodel elements to make them easily discernible by increasing their visual distance (REQ3) [Moo09]. Colors are used only redundantly to ensure that even if users perceive colors differently, they are still able to tell two different elements apart.

All elements of the graphical model are accompanied by text to differentiate instances of the same model element (REQ4). Every graphical element is displayed with an icon to resemble its purpose (REQ5).

[1]The icons used are provided by Icons8. last access 4.5.2022

For a single policy REQ6 is considered fulfilled as a policy with a trigger, an adjustment type, a constraint and a target group are represented by five elements. This is within the span of judgment, even with four more constraints the number of elements does not exceed nine. However, it has to be mentioned that more complex scenarios with multiple policies and target groups can exceed the span of judgment.

3.3 Feedback on the First Draft

To gather feedback, I created a seven-page handout that detailed the creation of a complete scaling policy, illustrated by images of the first draft. The handout also included a page containing the design considerations of the first draft and two more example policies to illustrate the graphical editor further. Furthermore, the participants were asked to provide three kinds of feedback: what they *liked*, what they *would like to see improved* and what they *did not like*.

In the following, I list the feedback given by the participants.

Participant 1 (P1) stated that the goal of editing and creating policies was fulfilled. However, P1 reported that understanding scaling policies was difficult. The participant suggested for the palette of graphical elements to be more uniform without different shapes, for example. They also found the connecting of elements “unintuitive” and suggested a different representation of relations between elements.

Participant 2 (P2) liked the draft overall, especially the different shapes to find groups of the same type of element. Nonetheless, P2 was unsure whether the “user experience” works. They suggested removing the connector element and e.g., replace it with “something to click on the border”. Furthermore, the participant found the inconsistent arrow design confusing. Participant 2 also suggested using an alternative type of diagram, changing the properties panel, to get more feedback on the draft and to not limit the graphical editor to Eclipse.

The third participant (P3) reported that the idea of the draft is going in the right direction. They also liked the different shapes and the direction of the triangle for adjustment types depending on whether to scale in or out. Participant 3 was “unsure” about the free-floating graph of nodes and edges and suggested a more restricted model. Furthermore, the participant mentioned the idea of “block-style” programming languages like Blockly[2] or NodeRed[3]. Another suggestion of P3 was to use shapes with holes e.g., for the scaling policy and its related elements. In addition, participant 3 “was not a fan” of the color scheme because it was too heavy, recommending the use of pastel colors. Finally, the participant mentioned that they had to search for what they wanted [in the palette].

In conclusion, the impression that I got is that the feedback was mostly positive. The participants provided novel ideas how the graphical editor could be improved. Nonetheless, it also became very clear that even when only asking three people about the design of a graphical model, you get three different opinions. While on the one hand, these opinions can be similar sometimes e.g., the impression that the free-floating model is a bit too open to help with understanding scaling policies optimally. On the other hand, the opinions can also be complete opposites e.g., participant

[2] `https://developers.google.com/blockly` last access 21.02.2022
[3] `https://nodered.org/` last access 21.02.2022

1 disliking the different shapes while participant 2 considers them especially useful. Finally, to draw the line between what to improve and what to keep, the stakeholders i.e., my supervisor and I, have decided what ideas to incorporate and what to disregard.

3.4 Second Draft

First, in the time between the creation of the first draft and the second, two additional concepts have been added to the graphical editor. The thrashing constraint has been added. Its purpose is to enforce a grace period on a policy so that a specified amount of time has to elapse before scaling out can occur after scaling in or vice versa. In addition, the concept of active and inactive scaling policies has been added, although not depicted in the second draft figure 3.2. Currently, there are several promising ways to show this e.g., by changing the color of the policy name or e.g., by filtering the policies shown on the canvas by their state.

Secondly, several graphical elements have been removed after the feedback session. The connector element has been removed from the palette as it is not strictly part of the graphical notation. Furthermore, the connecting lines between the scaling policy and its adjustment type, scaling trigger and constraints have been removed.

To help users better understand how to set up a scaling policy, connecting graphical elements with lines has been replaced with placeholders of the three aforementioned parts of a scaling policy. The user can therefore always see what shape has to be placed near the policy to properly set it up. Moreover, the size of the hexagons representing constraints has been reduced by 30% to hint at them being optional for scaling policies whereas the larger scaling trigger and adjustment type are mandatory. Nonetheless, the explicit connecting line between scaling policy and target group remains to signify what policies affect which target groups[4].

Based on the feedback, the use of colors has been dialed back. Color is no longer used as a visual variable to discern different types of model elements and instead a single “light blue” color has been chosen for all elements. Despite the removal of color as a discerning visual variable, this opens up the opportunity of adding colored elements in possible future work e.g., for visualizing analyses. The second draft can be seen in Figure 3.2.

After creating the second and final draft, I proceeded with the implementation of the prototype of the graphical editor for scaling policies.

[4]Target groups can be scaled by as many policies as desired.

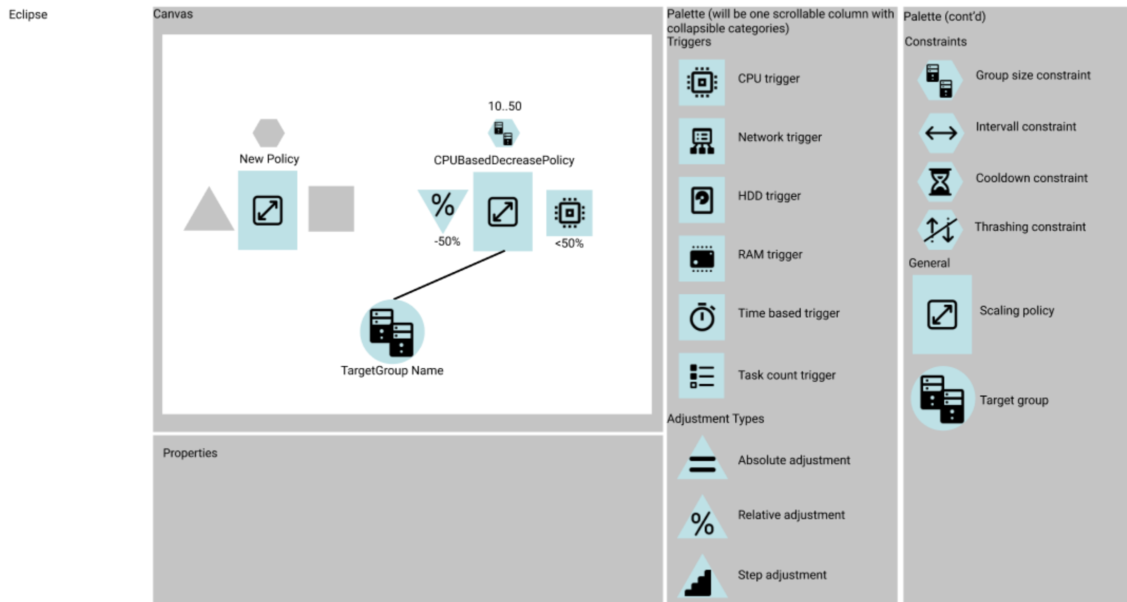


Figure 3.2: Second draft of the graphical editor showing a policy stub and a configured policy

4 Implementation

4.1 Foundations

As discussed in the previous Chapter 3, the goal is to implement a *graphical* editor for *scaling policies* that helps the software architect to create and understand scaling policies better. To find a suitable way of implementing the editor, I considered the following aspects.

Since the goal is to come up with a graphical editor, I informed myself about available tools and frameworks to facilitate the development of a graphical application. Before doing so, I also considered the other aspect i.e., implementing an *editor for scaling policies*. Scaling policies are specified as an Ecore model. Ecore models are a part of Eclipse Modeling Framework (EMF). As a result, I deemed Eclipse to be the most natural candidate to implement the graphical editor with, mainly because it allows to build the graphical editor directly on top of the Ecore metamodel inside Eclipse.

Eclipse has been [Gee05] and, I argue, still is a popular integrated development environment (IDE) for Java and other programming languages. In addition, Eclipse is platform-agnostic as it is offered[1] for macOS, Windows and Linux operating systems and even for both x86_64 and ARM64 architectures (except Windows), ideal for a research project where stakeholders, testers and potential users might be using varying operating systems e.g., depending on their personal preferences.

4.1.1 Framework Candidates

Graphical Modeling Framework

The first candidate was Eclipse Graphical Modeling Framework (GMF)[2]. GMF allows set up a graphical definition on top of the EMF model which is useful to specify nodes, edges, compartments and more[3]. These graphical elements can be configured with custom icons, constraints and validations[4]. In addition, actions and layouts are specifiable[5]. Nonetheless, all of this functionality is provided on a low level i.e., basic functionality such as nodes and edges are specified using provided wizards but advanced features such as actions are added by specifying XML and Java code directly.

[1] `https://www.eclipse.org/downloads/packages/` last access: 16.5.2022
[2] `https://wiki.eclipse.org/Graphical_Modeling_Framework` last access 9.3.2022
[3] `https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1` last access 9.3.2022
[4] `https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_2` last access 9.3.2022
[5] `https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_3` last access 9.3.2022

Graphiti

The second candidate was Eclipse Graphiti[6]. Graphiti builds on top of EMF, Eclipse Graphical Editing Framework (GEF) and Draw2D ("a layout and rendering toolkit [...] [for] Java SWT"[7]). Therefore, it operates on a different level of abstraction. Rather than specifying a GMF or GEF model to create a diagram, Graphiti offers a Java API and provides its own objects[8]. As a result, Graphiti promises to speed up and ease development and to offer a common look and feel. Finally, Graphiti is not strictly limited to Eclipse but its only rendering engine is currently Eclipse.

Sirius

The third candidate was Eclipse Sirius[9]. Sirius allows to create "your own graphical modeling workbench"[10]. It offers a "workbench for model-based architecture engineering" that can be "tailored to fit specific needs". To provide this functionality, Sirius is built on top of EMF and GMF[11].

In Sirius, one can specify one's own "modelers" i.e., graphical editors. This is done using the *specifier* perspective. These can then be used by users on the models specified for the graphical editor. Therefore, Sirius refers to this as the *user* perspective. In addition, it is possible to extend or modify Sirius beyond the functionalities provided to specify or use graphical models. Naturally, this level of Sirius is referred to as the *developer* perspective. A more detailed introduction into Sirius can be found in Section 4.2.

Framework Selection

Based on its description, Sirius appears to be the most promising contender. Since I was not familiar with the development of graphical editors, the high level of abstraction that Sirius provided, in combination with the depth that the developer perspective offered, led me to try it first.

By following Sirius' starter tutorial[12], I quickly understood how to set up a basic graphical editor with nodes, edges, custom icons. Even going a step further, it also showed how to create a palette. Furthermore, the advanced tutorial gave insight into other available elements such as containers, bordered nodes and edge creation tools. These would all become useful after the second draft.

After trying to apply the knowledge gathered from the tutorial, I confidently created working, early prototypes of the graphical editor while creating and adjusting the drafts.

By the time the draft phase ended, I was confident that what was drafted in Chapter 3 could be realized well enough in Sirius, that it became the immediate choice as the framework to use for the implementation.

[6] `https://www.eclipse.org/graphiti/` last access 9.3.2022
[7] `https://www.eclipse.org/gef/draw2d/` last access 10.3.2022
[8] `https://www.eclipse.org/graphiti/` last access 9.3.2022
[9] `https://www.eclipse.org/sirius/` last access 9.3.2022
[10] `https://www.eclipse.org/sirius/doc/` last access 9.3.2022
[11] `https://www.eclipse.org/sirius/doc/` last access 9.3.2022
[12] `https://wiki.eclipse.org/Sirius/Tutorials/StarterTutorial` last access 10.3.2022

4.2 Creating the Graphical Editor

To start, one creates a "Viewpoint Specification Project" (VSP). After configuring the basics of the project, one creates a new diagram where the root domain class has to be specified. In the case of the scaling policy meta model, the `SPD` class has been chosen as the root.

Now, the actual implementation can begin. The first goal is to be able to display all the visual elements specified in the draft.

4.2.1 Displaying the graphical elements

To do so, I started by implementing the scaling policy as a container. The reason to use a container instead of a node is to facilitate the design shown in Figure 3.2 where next to the policy, the adjustment type and scaling trigger are placed. Inside the container, there are three sub-containers. From left to right, there is one container for the scaling trigger, on its right there is the container to show the policy icon and farthest right there is the third container for the adjustment type. The leftmost and rightmost containers serve as both placeholders and surrounding boxes for the scaling trigger and adjustment type respectively.

Please note that all figures are of the latest version of the graphical editor which includes final adjustments made after conducting the evaluation study. The most prominent difference that resulted from the feedback is that the scaling trigger is placed to the left of the scaling policy and the adjustment type is placed on the right.

For the policy constraints, the top-level container of a scaling policy is given bordered nodes i.e., nodes that appear on the border of another node or container. These bordered nodes function as both a placeholder to hint at the ability to add constraints and to then display added constraints on top of a policy. In Figure 4.1, you can see how this looks in practice.

Figure 4.1: Empty Scaling Policy

On the left of the policy, the scaling trigger is displayed. To facilitate this, I created a node for each of the eight types of scaling trigger, visualized by their respective square-shaped representation. In addition, these nodes are sub-nodes of the scaling trigger container, placed on the left of the scaling policy. The resulting design can be seen in Figure 4.2. In reading direction i.e., from left to right, one can read a scaling policy as “If trigger X fires, then adjustment Y is applied”.

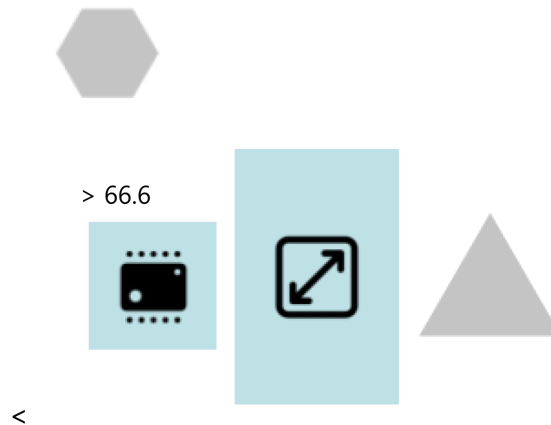


Figure 4.2: Scaling Policy with Trigger added

Now, to add the adjustment types, I created a node for each of the three adjustment types analogously to the scaling triggers. The nodes are visualized by their respective triangle-shaped representation. Moreover, they can only be placed right to the scaling policy so that they appear last in reading direction i.e., from left to right.

When an adjustment type is placed, the gray placeholder is hidden automatically. This also applies to the scaling trigger and policy constraints. Depending on whether the adjustment type is scaling out or in, the triangle points up or down respectively. A policy with an adjustment type added, can be seen in Figure 4.3.

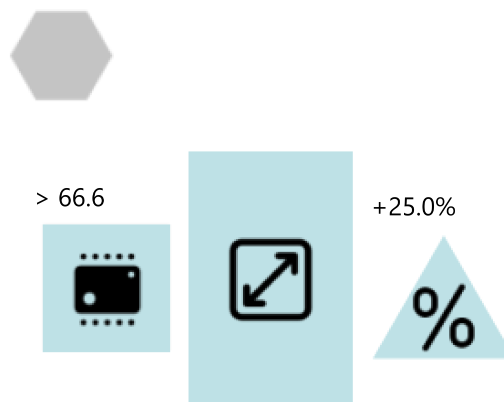


Figure 4.3: Scaling Policy with Adjustment Type

Above a scaling policy, the policy constraints are displayed. The hexagon-shaped placeholder and the four types of constraints are all represented by bordered nodes attached to the root container that represents a scaling policy. The root container is the large box without a border that encompasses the adjustment type, the policy placeholder representation and the scaling trigger.

Again, all constraints are visualized by their respective icon encompassed by a hexagon. The hexagons are intentionally smaller than policy, trigger and adjustment type because they can be omitted if not needed. A policy with two constraints added, can be seen in Figure 4.4.

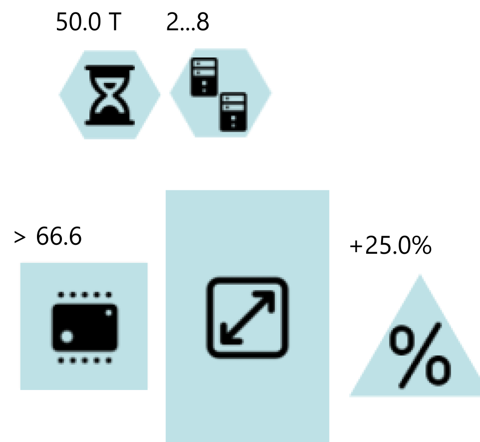


Figure 4.4: Scaling Policy with Constraints added

Finally, the target groups are implemented as nodes that are represented by their large circle-shaped icon. A target group can be connected with a scaling policy via an edge. The edge stands for the fact that the respective scaling policy affects the connected target group. The connection is visualized by a black line. In Figure 4.5 you can see a scaling policy with its affected target group. For the sake of completeness, an example name has been added for the scaling policy and the target group.

4.2.2 Additional in-line information

In order to provide further information about the graphical elements, practically all of them are accompanied by text. This can, for example, be seen in Figure 4.5.

Above the scaling policy placeholder, the name of the scaling policy is displayed. The same goes for the target groups.

For the adjustment types, the adjustment "value" is displayed next to the triangle. Concretely, this is either an absolute value, an percentage or a step value depending on the respective adjustment type. It should be mentioned that these can be both positive values, prefixed with a plus sign, or negative values prefixed by a minus sign.

Scaling triggers are accompanied by text that denotes the threshold and its direction. For all the non-time based triggers, a greater-than or less-than sign is shown for the threshold direction and the threshold value is displayed behind it.

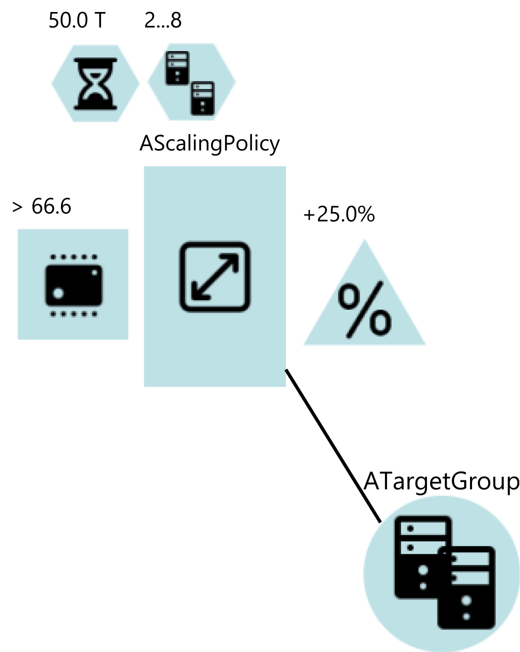


Figure 4.5: Scaling Policy with a Target Group added

Similarly, the idle time trigger is accompanied by this text structure: `idle (>|<) threshold T`. Where the threshold direction is denoted by the greater or less-then sign. The fact that the trigger specifies idle time is denoted via the `idle` prefix and the `T` at the end stands for time units.

Analogously, the response time trigger text is structured as `RT (>|<) threshold T`. The only difference being that `RT` stands for response time in this case.

Lastly, the point in time trigger is simply denoted by `pointInTime T` to specify the time when the trigger is applied.

Three of the policy constraints, namely the interval, cool-down and thrashing constraint are accompanied by text that is structured as `timeDuration T`. The other constraint, that specifies the allowed group size is accompanied by `minSize...maxSize` where the minimum and maximum group size is shown.

Editing in-line properties

Furthermore, to enable the user to change the model by editing the in-line information next to an element, a bit more configuration is required.

For simple cases such as the scaling policy, one only has to create the ability to “direct edit” the label of scaling policy containers and specify that the text entered is stored as the `policyName`. The same goes for all the graphical elements where the text showed next to it is exactly what is stored in the model.

The more interesting cases are where the in-line information requires processing. The language that I have used to do this is Acceleo Query Language (AQL) and it is natively integrated into Sirius. Therefore it requires no further set-up. Documentation on AQL can be found here[13] . In the following, I present two examples.

The first example is how to handle the threshold and its direction for scaling triggers such as the CPU trigger. First, to extract the threshold one configures the direct edit label of the CPU trigger to set the threshold to the output of `[arg0.replace('< ','').replace('> ', '')/]`.

In the following, I explain the code. The `[/]` denotes an AQL expression. AQL then takes the first argument `arg0`, i.e. the input of the label field and removes the greater or the less-than sign. This leaves only the threshold, as it was desired (assuming of course the input to the field complies to the expected format).

Secondly, for the threshold direction the output of `[if arg0.first(1) = '>' then 'EXCEDEED' else 'UNDERCUT' endif/]` is used. What this AQL expression does is that it takes the first digit of the label and compares it to the “>” symbol. If in the first place, a “>” sign is present, the threshold direction is “exceeded”, otherwise i.e. when a “<” is present (as the only other sensible alternative) the threshold direction “undercut” is used.

The second example is how to extract the group size thresholds, namely `minSize` and `maxSize` from its label. To extract the minSize, the following query is used: `[arg0.tokenize('...', false)->first()/]`. This splits the input label at the “...”-point. Then the first token, i.e. the part before the “...” is stored in the `minSize` variable.

Analogously, the maxSize is extracted by: `[arg0.tokenize('...', false)->last()/]`. The only difference being that in this case, the last token is used.

4.2.3 Creating graphical elements

On the right hand side of the graphical editor, the graphical elements that can be placed are shown under Palette (see Figure 4.6).

[13] https://www.eclipse.org/acceleo/documentation/aql.html last access 24.3.2022

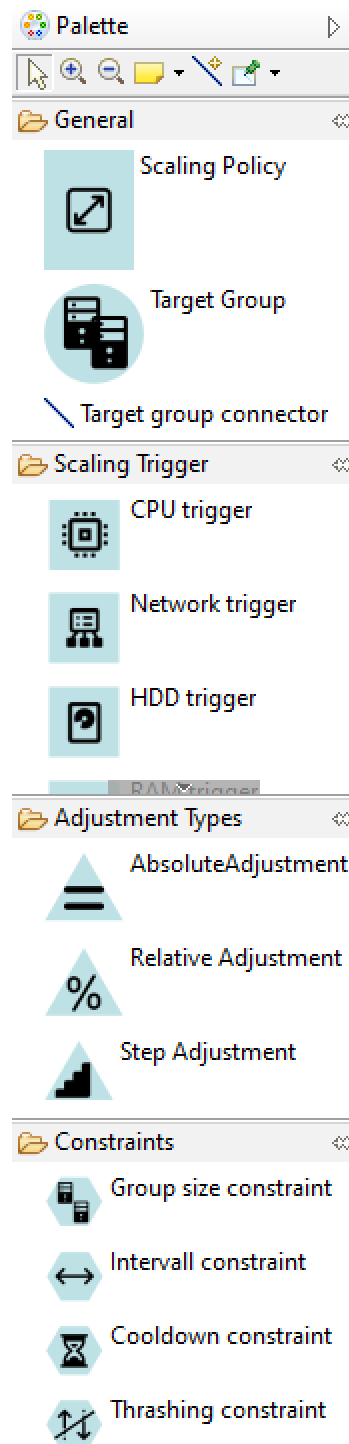


Figure 4.6: Palette of the graphical editor

The available elements are grouped into four categories. Namely, “General”, “Scaling Trigger”, “Adjustment Type” and “Constraints”. What is available in the last three groups is self-describing, e.g., the “Adjustment Type” group contains all available adjustment types. In the “General” group, there are three elements: scaling policies, target groups and a connector tool that allows connecting a scaling policy to a target group and vice versa.

While most of the elements of the palette have been implemented easily and successfully in Sirius, the target group stood out. The issue has been that Sirius requires a root element that is represented by the canvas. In my case, this is the `SPD` node of a `.spd` file. However, to create an element, Sirius needs to know where to store it. For example, when adding a trigger to an already created scaling policy, Sirius stores the trigger through the EReference from the scaling policy to the trigger.

In the original meta-model for `SPD`, the target group is only referenced from a scaling policy. Therefore, when placing a target group on the canvas (that represents the `SPD` node), Sirius will try to create the target group “inside” the `SPD`. This fails, as target groups were originally not referenced by the `SPD` root element.

Since finding a way to work around this directly, e.g., by creating the target group manually and then importing it, proved to be more difficult than anticipated, I adjusted the metamodel in order for `SPD` to contain target groups. In Figure 4.7 you can see that the `SPD` type now has a containment reference to target groups. This allows Sirius to properly create, reference and destroy target groups through the graphical interface.

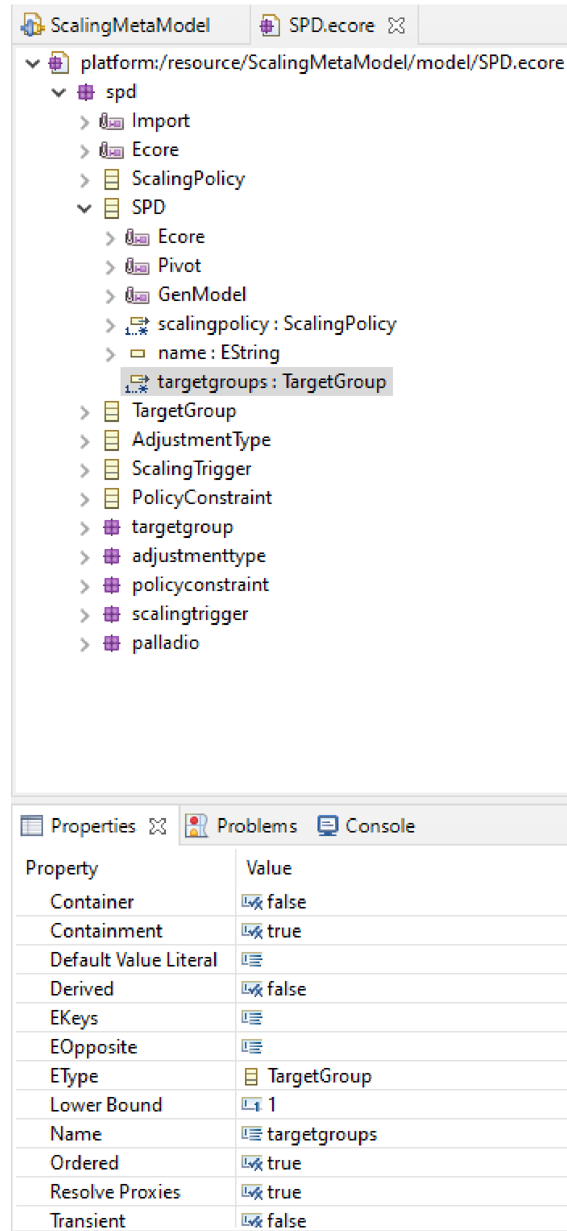


Figure 4.7: Excerpt of the Ecore meta model

The only other element type that stands out slightly are the constraints. While they technically work perfectly fine, because they are realized as bordered nodes in Sirius, they have to be placed on the slim, invisible, northern edge of the scaling policy container. This is a result of time constraints. I anticipate that this will impact the user experience slightly.

4.3 Discussion

In the following, I explain the relation between graphical metamodel and the Ecore metamodel. In addition, I describe conceptually how changes to the Ecore metamodel have to be handled by the graphical metamodel. Lastly, I discuss how constraints could be realized within the graphical editor.

4.3.1 Relation between Ecore metamodel and graphical metamodel

The graphical metamodel is built on top of the Ecore metamodel. This means that the graphical metamodel is specified inside the Eclipse plugin that is providing the SPD Ecore metamodel. In other words, the graphical metamodel and editor are specified alongside the tree-editor i.e., in the inner Eclipse instance. More on the tree editor and the underlying SPD metamodel can be found in this student research project [BFGÖ21].

Inside the inner Eclipse instance where the SPD metamodel is available to instantiate, reside one or more modeling projects. These modeling projects serve two purposes. First, inside them, the SPD models are stored. Secondly, the modeling projects will be configured to be able to create representations, i.e., graphical models based on the specified graphical metamodel.

The graphical metamodel is described in a "Viewpoint Specification Project" (VSP). For the VSP, the SPD metamodel Eclipse plugin is specified as a project dependency. Through the aforementioned dependency, the graphical metamodel will be connected to the SPD metamodel.

Now, for the actual graphical metamodel. It is specified inside the VSP. There, under a viewpoint, a diagram is specified which in turn contains (at least) one layer where then all the containers, nodes etc. are specified for the graphical metamodel. The graphical metamodel/editor is specified in the `ScalingMetaModel.odesign` file of the VSP.

To be able to instantiate the graphical metamodel, the viewpoint of the VSP has to be enabled for the respective modeling project where the SPDs reside. Then, per SPD, a diagram has to be instantiated to create a graphical model of the SPD. Finally, editing the diagram is technically equivalent to "using the graphical editor (on that SPD)". In Figure 4.8, the architectural structure, at development time, described above is visualized. The architecture at runtime provides the dependencies and, of course, the graphical editor for use in any modeling project containing `SPDs`.

To explain the connection between a graphical model element and the SPD model, let me provide you with an example how a scaling policy in the Ecore model is associated with the respective outermost scaling policy container in the graphical model.

The graphical scaling policy container is configured to represent the domain class `spd::ScalingPolicy`. The, so called, "semantic candidates", i.e., where Sirius searches for elements of the specified domain class for the scaling policy container, are of the `feature:scalingpolicy`. The latter is the named reference in the Ecore model from the SPD meta-class to the scaling policy meta-class. Sirius uses the named reference to collect all the scaling policies stored inside the SPD and displays them as scaling policy containers. Again, for the graphical metamodel, the SPD serves as the root element that is represented by the canvas of the graphical editor.

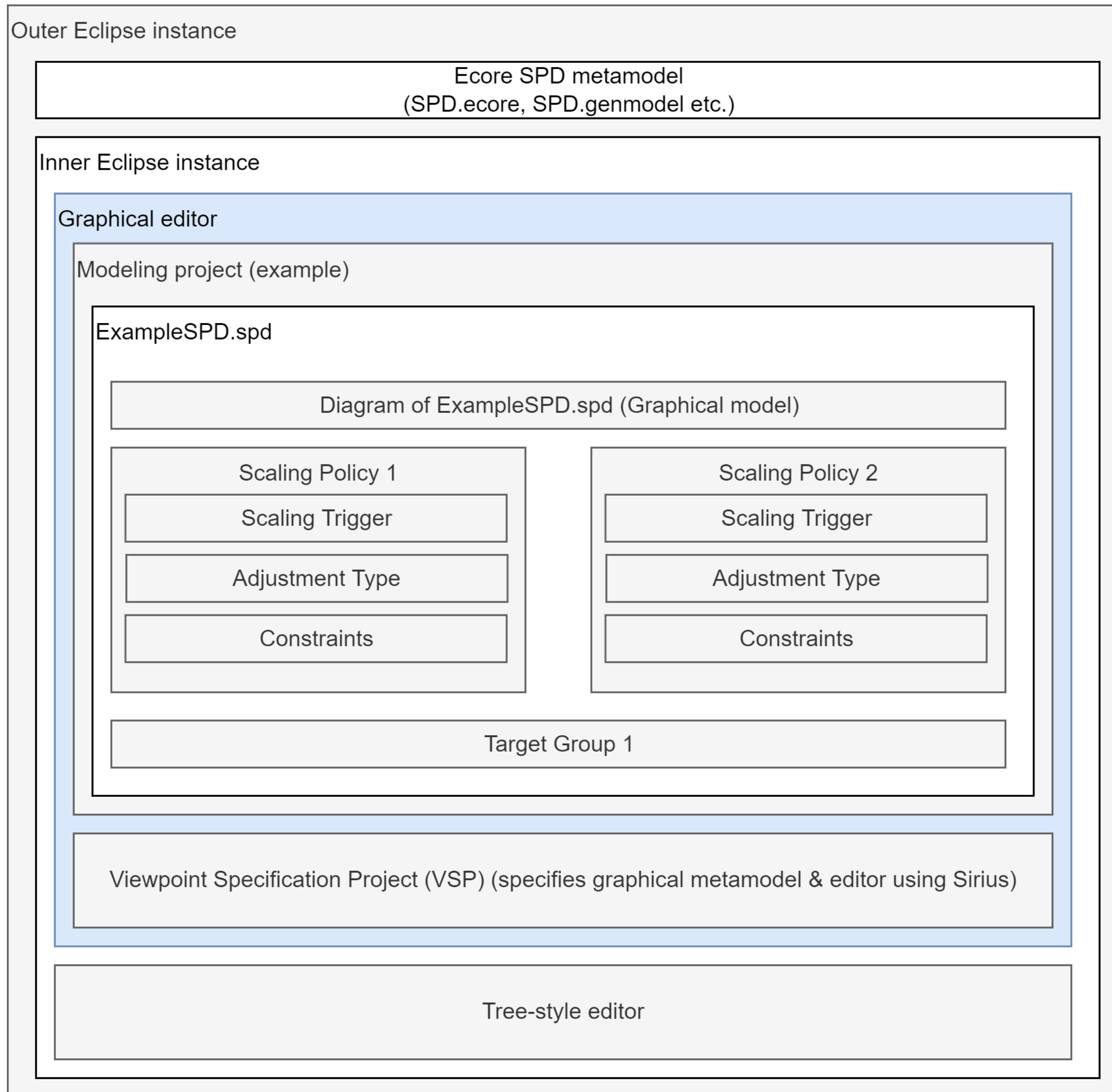


Figure 4.8: Architecture diagram

Analogously, all graphical metamodel elements are discovered, displayed and stored through the Ecore references of the Ecore metamodel by Sirius. The graphical representations, e.g., where the instances of graphical elements of an SPD are placed on the canvas, are stored in the `representations.aird` file of the respective modeling project.

4.3.2 Handling changes to the Ecore metamodel

In the following, I discuss how changes to the Ecore metamodel can be handled. For this, I will go over three possible cases: additions to the metamodel, changes to the existing parts of the metamodel and removals from the metamodel.

First, if a new metamodel element is added, it is necessary to create the needed visual representations in the graphical metamodel to enable the graphical editor to edit the new element. Existing SPDs and their visual representations are able to be extended with the newly added elements.

Secondly, if an existing Ecore metamodel element is modified, it is necessary to also modify the graphical metamodel. For example, if the name of an EReference is changed, any use in the graphical metamodel of the reference also has to be updated in the graphical metamodel. In addition, existing graphical models and their respective SPDs will require to be updated to reflect the new underlying metamodel for the graphical editor to handle them.

Lastly, if a metamodel element is removed from the Ecore metamodel, it will have to be removed from the graphical metamodel as well. Similarly to the case of modifying the metamodel, existing SPDs and their visual representations will need to be updated to reflect the removed elements for the editor to be able to handle them correctly.

4.3.3 Realizing constraints

In the following, I discuss how constraints of the metamodel could be realized by the graphical editor. The constraints themselves can e.g., be specified using the Object Constraint Language (OCL) [CG12].

To start, I discuss how the constraint which specifies that a name of a scaling policy has to be unique, could be enforced by the graphical editor. One way to realize this is to validate the constraint every time a name of a scaling policy is changed or rather tried to be saved. In case there is no name conflict, the graphical editor can seamlessly save the SPD and continue. However, should the newly specified name conflict with an already existing name, the editor could generate a popup with an error message stating e.g., "The name you are trying to set is already in use. Please specify a different name.". Then the user would be asked to change the name of the policy.

It has to be kept in mind that scaling policies are created with an empty name and therefore the case of multiple policies with an empty name should not cause an error popup, to allow users to place as many policies as needed without being forced to name them before creating the next. To handle this case, a warning e.g., during validation of the model can be displayed to inform the user that there are policies with no name specified.

The second type of constraints I discuss are "nonsensical configurations". For example, a relative adjustment by 0% or a utilization based trigger with an exceeds 0% threshold can be a configuration that is actually not intended or just "nonsensical".

To handle these configurations, I propose using warnings at validation time. This can either be a live validation or a dedicated validation step.

For the live validation e.g., the value displayed next to the model element could be displayed in a different color to highlight a "nonsensical" configuration value. For instance, an "exceeds 0% CPU utilization" scaling trigger can be highlighted by displaying the "> 0%" in yellow or orange.

On the other hand, after a dedicated validation step, a list of warnings and errors can be displayed e.g, in a sub-window below the canvas of the graphical editor. There, the “nonsensical” configuration values would then show up as warnings, assuming they are not prohibited by a metamodel constraint, in which case it must be treated as an error. For example, as described in the first constraint I discussed.

Another variation of a “nonsensical” configuration is a contradictory configuration where the interpretation of multiple configuration values leads to ambiguity that must be resolved. Examples for this can be constructed by combining two constraints of identical type but different values that affect the same target group.

For example, two group size constraints where one states a group size “1..5” and the other specifies a group size between 3 and 7. In this example, it cannot be programmatically resolved what the intended group size constraint is. It could be 1 to 5, 3 to 7 or even combinations like 1 to 7 or 3 to 5.

I propose detecting these in a dedicated validation step. A live validation could be considered as well, however with an increasing amount of policies and constraints, the performance of the editor would be impacted negatively.

To detect these ambiguities or contradictions, all relevant constraints that apply to a target group have to be collected and then checked on a per-constraint type basis whether any of the configuration values cause ambiguity or contradiction. If any are detected, the user will then see them listed in the validation report.

An alternative approach to remedying contradictory constraints that directly affect a target group is to move these constraints from being sub-elements of scaling policies to being sub-elements of the target group in the metamodel.

In this case, by enforcing that a target group can only have e.g., one group size constraint, the issue can no longer be constructed. Enforcing only one instance of a type of constraint per target group can be done efficiently, live during model specification or editing which poses an additional advantage to this approach.

4.4 Overview of the Graphical Editor

As the previous chapters have provided insight into the creation of the graphical editor, in this chapter I will be concerned with giving an overview of the graphical editor. In the following, I depict how to create an example SPD model with the graphical editor to highlight the key aspects of scaling policies.

The example business service that will be managed is a web service that allows users to sign up for an institution e.g., a university. The example consists of a sign-up service and an account management service.

The example fulfills the following six requirements.

1. Both the resources for the sign-up service and the account management service are to be scaled.

2. At least two sign-up service instances are to be always running. In addition, no more than 10 instances can be utilized for the sign-up service. The account management service will usually run on a single instance. However, an additional instance of the account management service can be enabled on demand.
3. If the CPU usage of the sign-up service exceeds 70%, the number of instances shall be doubled.
4. If the CPU usage of the sign-up service drops below 20%, the number of instances shall be reduced by one.
5. If the account management service RAM usage exceeds 80%, an additional instance shall be spun up for faster processing of incoming requests.
6. If the account management service RAM usage undercuts 35%, the additional instance shall be shut down.

To create the example with the graphical editor, one first sets up a new SPD model and specifies the name for the model. In this example, I name the SPD “Example”. This can be seen in Figure 4.9.

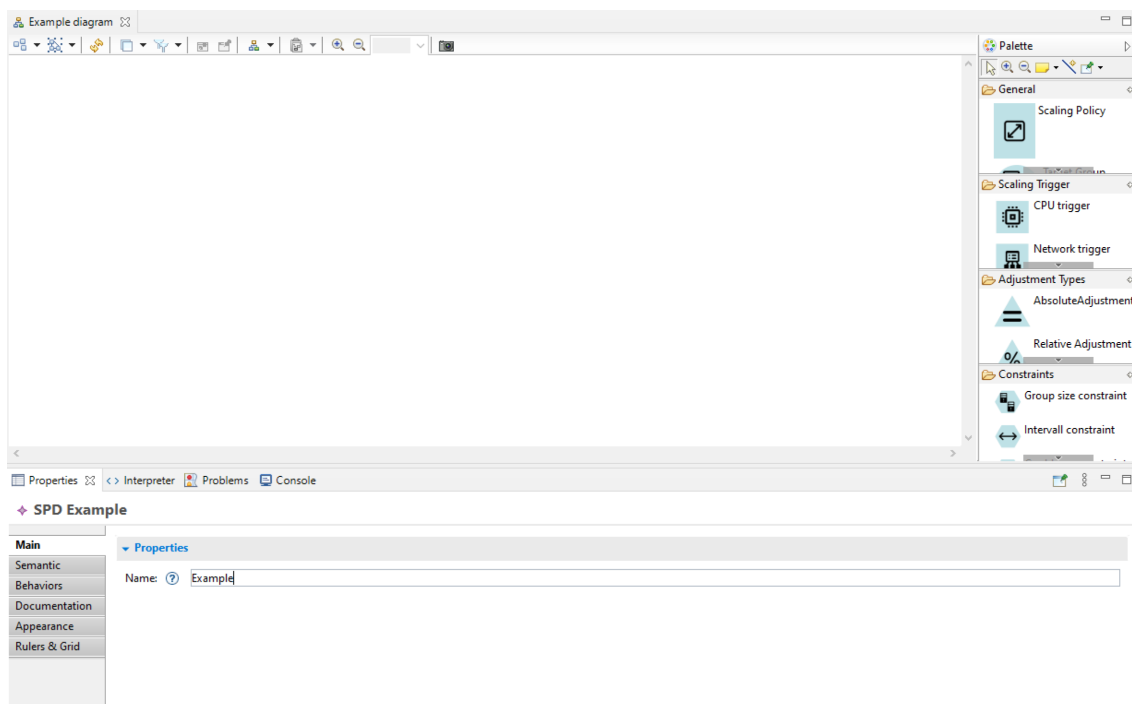


Figure 4.9: Example SPD

For the first requirement, I model the resources for the sign-up service and the account management service with two target groups named “Sign-up service” and “Account Management service” respectively. This can be seen in Figure 4.10.

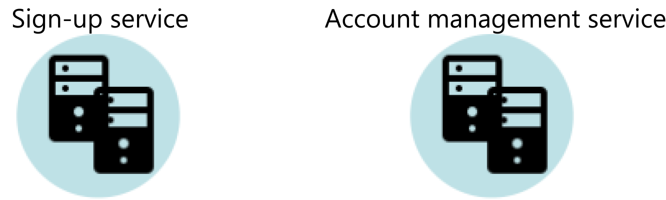


Figure 4.10: Example with constraint 1 fulfilled

In order to model the second constraint, I use policy constraints. To ensure that the sign-up service always has between two and ten instances running, I create a scaling policy with a group size constraint. The group size constraint has a `minSize` of two and a `maxSize` of ten. Analogously, I create a scaling policy with a group size constraint for the account management service with a `minSize` of one and a `maxSize` of two. Then, both the policies are connected with the appropriate target group using the target group connector tool from the palette. The result can be seen in Figure 4.11.

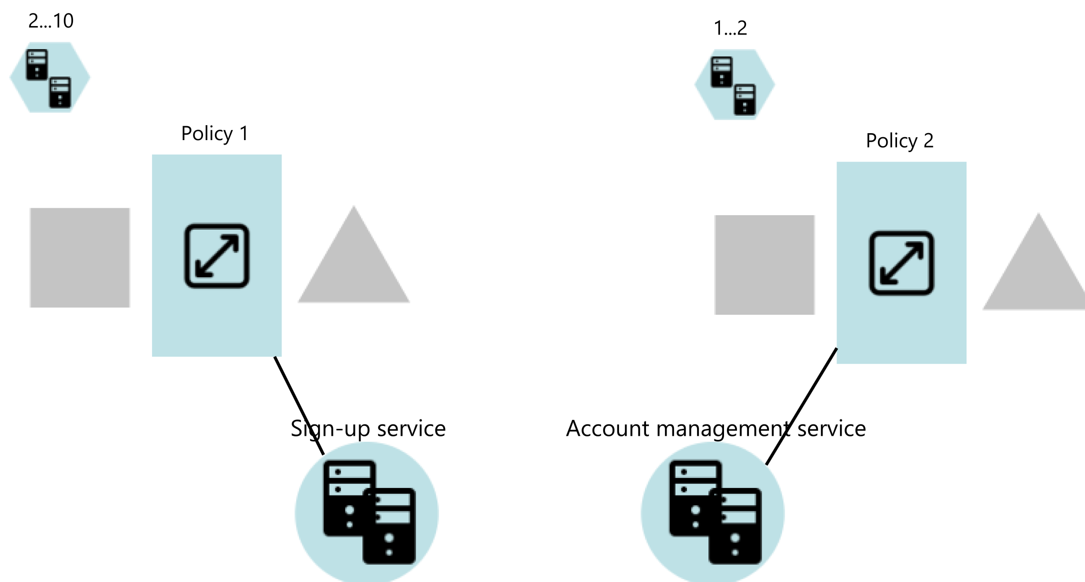


Figure 4.11: Example with constraints 1-2 fulfilled

For the third constraint, I add a scaling trigger and an adjustment type to policy 1. In order to activate at more than 70% CPU usage, I create a CPU scaling trigger with a threshold of 70 and the threshold direction “exceeded”. To double the amount of web server instances as a result, I add a relative adjustment type with a percentage value of +100% to policy 1. This can be seen in Figure 4.12.

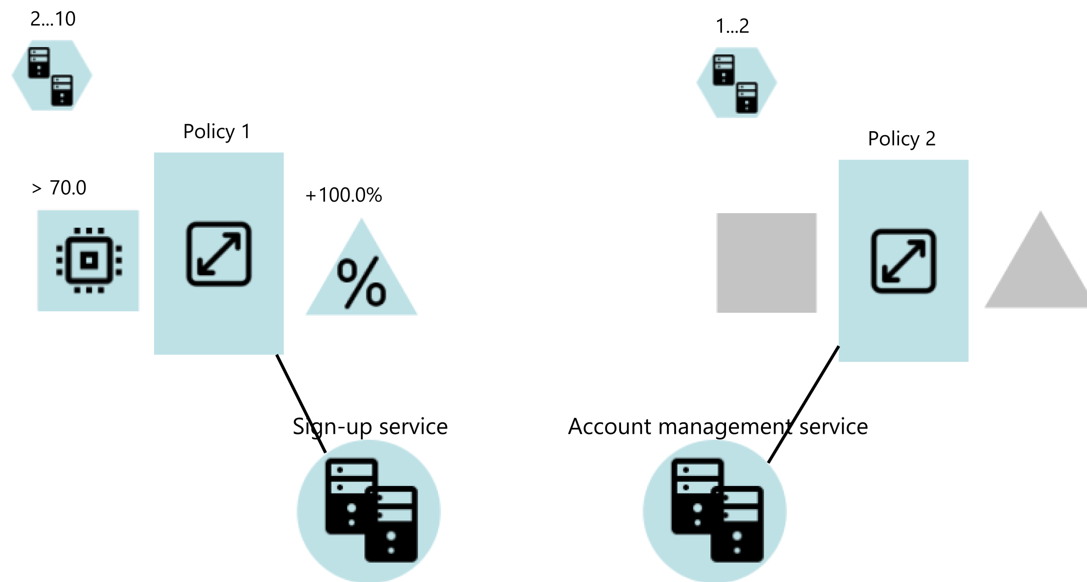


Figure 4.12: Example with constraints 1-3 fulfilled

Next, to fulfill the fourth constraint, I add policy 3 to the SPD and connect it to the sign-up service. To reduce the amount of instances by one when the CPU usage is below 20%, I add a CPU scaling trigger to the policy and set the threshold to 20 and the threshold direction to “undercut”. Furthermore, I create a step adjustment with a step value of -1 to reduce the amount of instances by one. This can be seen in Figure 4.13.

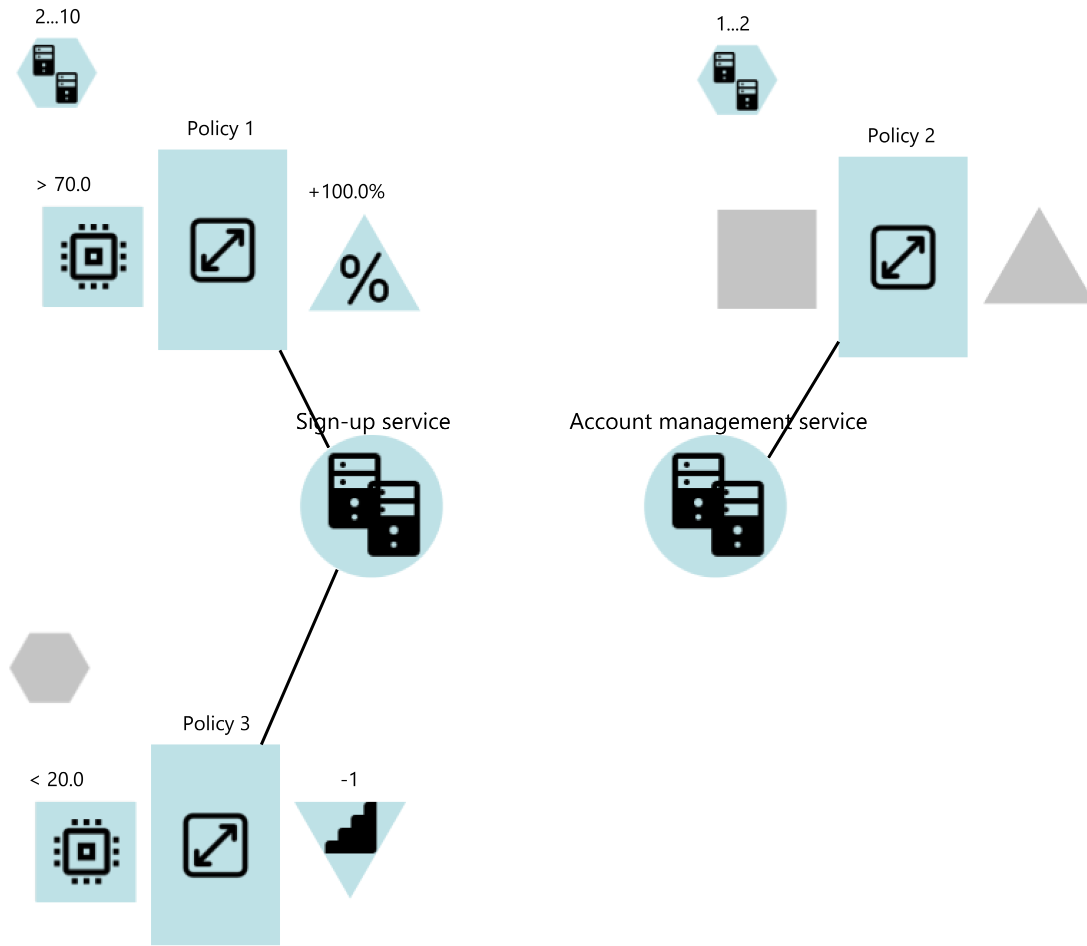


Figure 4.13: Example with constraints 1-4 fulfilled

For the fifth constraint, I add a scaling trigger and an adjustment type to policy 2. To add a second instance to the account management service when the RAM usage exceeds 80%, I create a step adjustment with an adjustment value of +1 and a RAM utilization trigger with a threshold of 80 and the threshold direction set to “exceeded”. This can be seen in Figure 4.14.

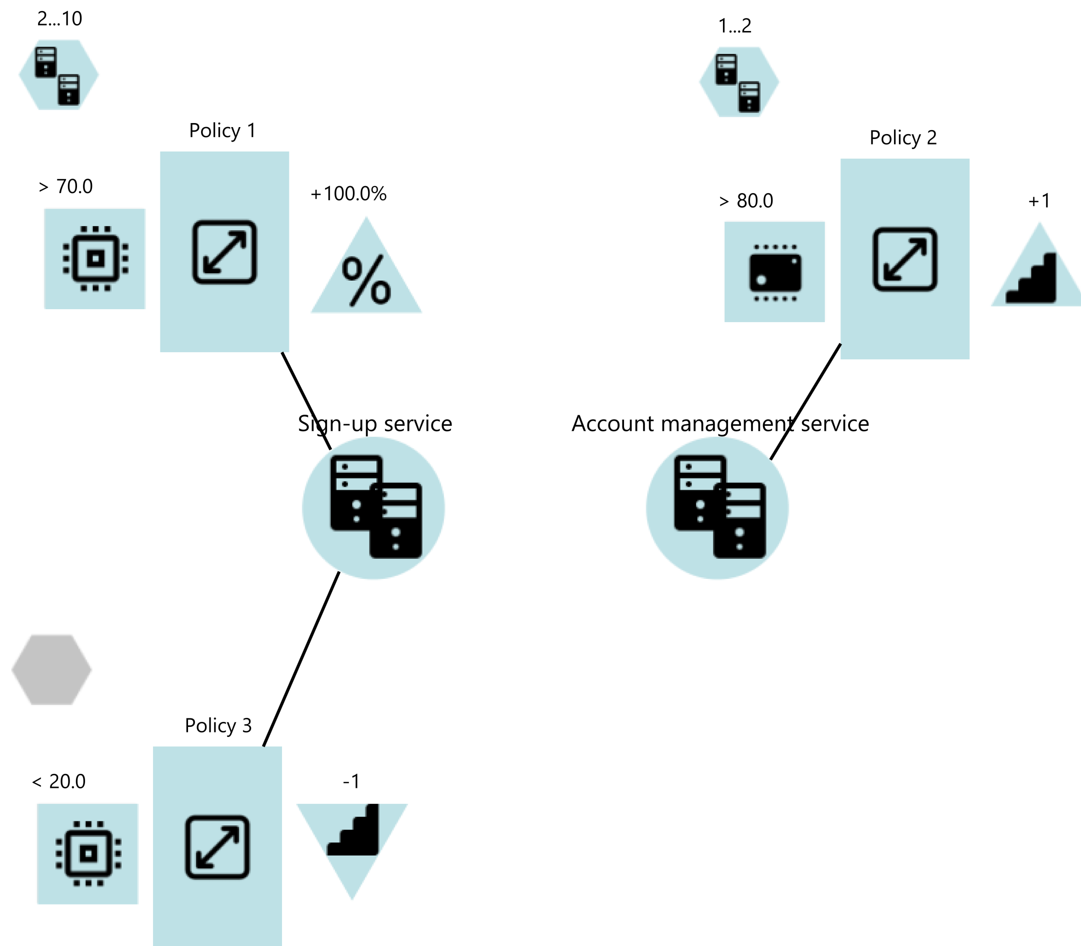


Figure 4.14: Example with constraints 1-5 fulfilled

Lastly, for the sixth constraint, I create scaling policy 4 which is connected to the account management service. In order to shut down the second server instance, when the RAM usage is below 35%, I configure the additional policy with a RAM utilization trigger with a threshold of 35 and a threshold direction of "undercut". Moreover, I specify an absolute adjustment type with a goal value of one. This can be seen in Figure 4.15.

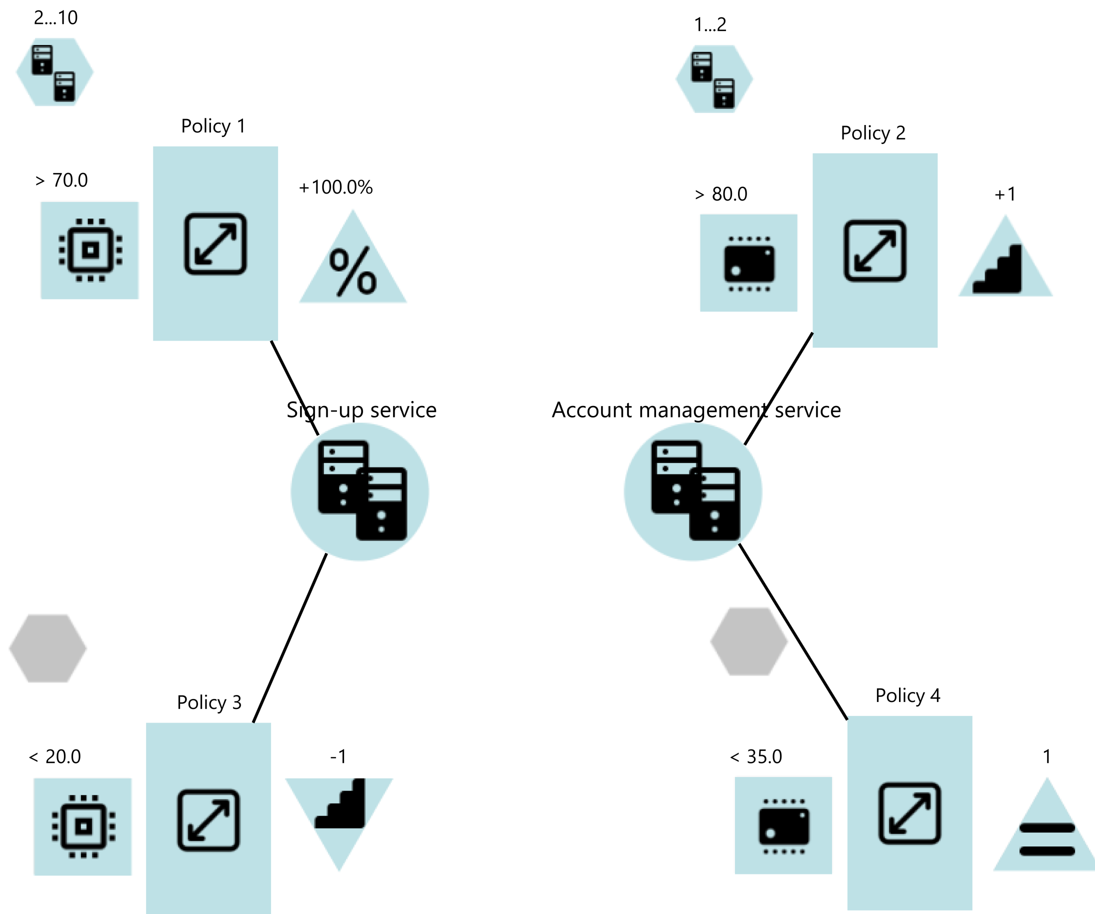


Figure 4.15: Example with all six constraints fulfilled

In summary, the example shows how all the different aspects of scaling policies are realized in the graphical editor. You have seen how to create target groups and how to add scaling policies. In addition, you have seen how policy constraints are used. Furthermore, several scaling triggers and adjustment types have been shown as well.

5 Evaluation

5.1 Study Design

The study was designed as an online live-session in which participants tried out the graphical editor and then filled out a questionnaire on their experience with the graphical editor. As a result, valuable empirical evidence as to how the graphical editor can help with the creation and understanding of scaling policies was gathered.

5.1.1 Preparation

In preparation of the study, I set up a slide deck and a virtual machine image containing the graphical editor for the participants to go through and set up in advance of the session.

The slide deck began with a brief motivational introduction to the need of scalability in software systems such as those using Microservice architectures. Furthermore, the slide deck contained an overview of the SPD metamodel and, more importantly, an overview of the graphical metamodel.

The overview of the graphical metamodel entailed a visual overview of the graphical metamodel elements and documentation on example elements of the metamodel and their associated properties available in the editor.

Lastly, the slide deck contained a motivation task consisting of two requirements where, in a ten-minute video, I showed the participants how to realize these two requirements using the graphical editor.

Namely, these two requirements were the following.

1. Both the resources for the sign-up service and the account management service are to be scaled.
2. At least two sign-up service instances are to be always running. In addition, no more than 10 instances can be utilized for the sign-up service. The account management service will usually run on a single instance. However, an additional instance of the account management service can be enabled on demand.

The implemented example can be seen in Figure 5.1. The original version that the participants received can be found in Appendix A.2 as Figure A.1.

Furthermore, the task's requirements are identical to those depicted in the example in Section 4.4. There, the steps needed to create the example solution of the task can be found.

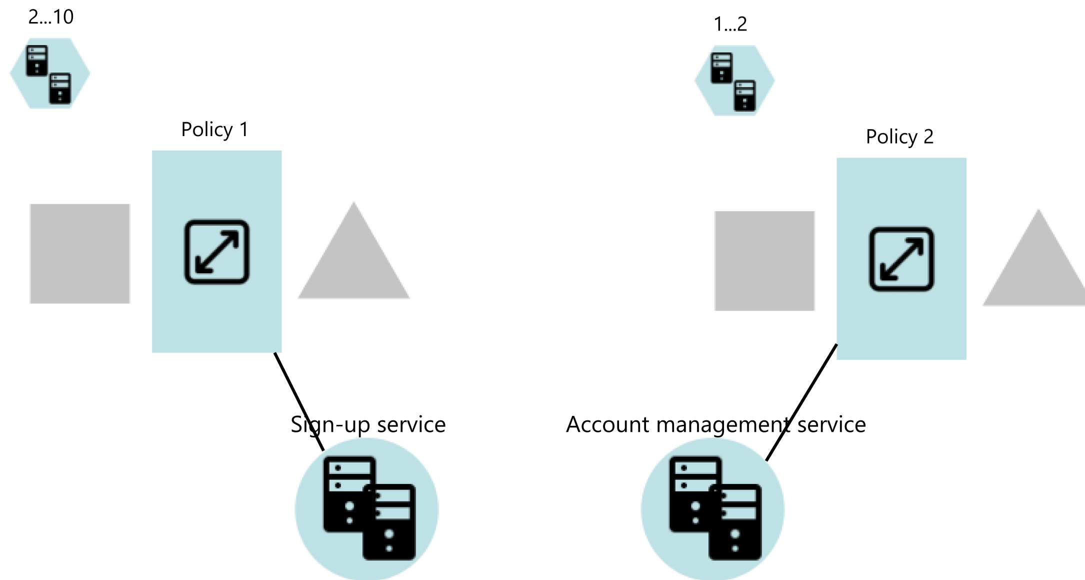


Figure 5.1: Given example for the evaluation study (latest graphical model)

To make sure, participants had a roughly-equal preparedness for the evaluation task and survey, they were asked to spend no more than 30 minutes on the given slide deck that they received a week in advance of the session.

5.1.2 Participants

With regard to whom I invited to participate, there were two groups.

The first group were interested members of the SQA department, the department that supervised this thesis. They were selected because of their expertise in related fields including (but not limited to) e.g., microservice architectures and software modeling (including graphical models). Furthermore, some of them had already previously participated in the feedback session on the graphical metamodel in Section 3.3 and were therefore already familiar with the concept of scaling policies.

The second group that was invited were experts from the industry including employees of AEB. AEB is a company that is concerned with providing solutions for logistics and trade on a global scale using IT[1].

They were invited to evaluate the graphical editor in a broader context, extending beyond academic experts. Moreover, as "experts in the field", they are able to better judge and give feedback on practical considerations of scaling policies and the graphical editor.

[1] https://www.aeb.com/de-de/ueber-aeb/index.php last access 25.4.2022

5.1.3 Session content

As stated in the beginning of this chapter, during the evaluation session, participants were given the task to try to implement four further requirements extending the given graphical model.

The goal of this exercise is to allow the participants to experience the graphical editor by first hand and, of course, form their own opinions on it.

The four requirements were the following.

3. If the CPU usage of the sign-up service exceeds 70%, the number of instances shall be doubled.
4. If the CPU usage of the sign-up service drops below 20%, the number of instances shall be reduced by one.
5. If the account management service RAM usage exceeds 80%, an additional instance shall be spun up for faster processing of incoming requests.
6. If the account management service RAM usage undercuts 35%, the additional instance shall be shut down.

The participants were given 30 minutes to try to extend the graphical model based on the additional requirements. The idea behind these requirements was to allow the participants to get an impression on as many (graphical) metamodel elements as possible.

As you can gather from the six requirements: target groups, scaling policies, policy constraints, adjustment types and scaling triggers are explored either by the warm-up task that was given or by the task during the session.

5.1.4 Questionnaire

After the 30 minute task, participants were given 15 minutes to fill out the following questionnaire about their experience of the graphical editor and some questions on their self-described competences.

The questionnaire was structured in four sections: two questions on the participants' competences and where they could see the graphical editor being used, nine questions directly regarding the graphical editor and the participants' experience with the task, an open question on whether they faced any problems and lastly an open question on suggestions for the future.

In each of these four sections, the questions were randomized to mitigate the order of questions leading to bias in the selection of answers [SH95].

For the second section i.e., the questions on the graphical editor and the participants' experience, the basis of these questions was based on the goal question metric (GQM) [BCR94] approach.

Again, the goal was to make creating/working with and understanding scaling policies better. Based on this goal, the following nine questions were asked.

- Did you perceive the graphical editor as helpful for you when working with scaling policies? (5-point Likert scale, not helpful - very helpful)

- Did you have enough time to solve the given task? (5-point Likert scale, too little - too much)
- Does the visual representation of the scaling policy seem appropriate? (yes/no)
- Does the visual representation of the adjustment type seem appropriate? (yes/no)
- Do you perceive the direction-dependent visual representation of adjustment types as useful? (yes/no)
- Does the visual representation of the scaling trigger seem appropriate? (yes/no)
- Does the visual representation of the policy constraints seem appropriate? (yes/no)
- Does the visual representation of the target group seem appropriate? (yes/no)
- Do you perceive the gray placeholders around scaling policies as helpful? (yes/no)

The answers to these questions resulted in subjective metrics either on a 5-point Likert scale [JKCP15] or the number of participants that found a particular detail of the graphical editor helpful or not.

5.2 Results

In the following I summarize the results gathered with the questionnaire. The raw results can be found in the appendix under Appendix A.1. The study was performed with nine participants.

5.2.1 Summary

When asked for their self-described competences: all participants described themselves as software engineers, seven described themselves as researchers while also seven participants described themselves as competent in microservice architectures. Moreover, five participants self-described the competence of DevOps and four participants described themselves as competent with other cloud related fields, as well. In addition, four participants described themselves as students and three as teachers, both in computer science or software engineering-related fields. One participant described him- or herself as a UI/UX developer.

Secondly, when asked about in which phase of the software development lifecylce [Rup10] participants' would see a potential use for a graphical editor for scaling policies (beyond the current prototype), the results tended to later phases of software development. The phases given as multiple-choice answers were loosely based on the stages of the waterfall model [Rup10].

Only three participants saw the requirements phase as a potential use. Five participants could see a use in the planning phase of a software project. Most participants, seven to be precise, were able to see potential use in the phases: design, documentation and communication and operations. Eight participants saw potential use during the deployment phase. Interestingly, every phase found at least one participant where the participant could see the graphical editor as potentially useful.

On a 5-point Likert scale regarding the question whether they found the graphical editor helpful when working with scaling policies, from not helpful (1) to very helpful (5). Five participants replied 4, one participant replied 5 and three participants gave a 3. Overall, six participants therefore gave a score tending towards very helpful while three participants decided for the middle between not helpful and very helpful.

With regard to the amount of time given for the task, again using a 5-point Likert scale between too little (1) and too much time (5). Five participants reported they had too much time (5), three participants answered 4 and one participant gave a 3 i.e., just the right amount of time.

All participants found the visual representation of scaling policies, adjustment types and scaling triggers i.e., the mandatory elements of a scaling policy appropriate. Both for the visual representation of the target group and the policy constraints, eight out of nine participants found the representation helpful. Moreover, the direction-dependent adjustment types and the gray placeholders around the scaling policies were found to be helpful by eight of nine participants.

5.2.2 Problems

Furthermore, all participants reported at least one problem they encountered. In the following, I provide an overview of the reported problems.

Seven out of nine people reported being unable to add constraints to scaling policies. This was expected because constraints are attached on the invisible border of scaling policies as mentioned in Section 4.2.3. Surprisingly, one participant reported being able to add constraints to scaling policies but neither adjustment types nor scaling triggers. In addition, multiple participants reported the line connector between scaling policy and target group to sometimes become invisible. Furthermore, two participants mentioned the target group connector to be usable only one way. Another complaint that four participants provided, was lack of space for adjustment types and scaling triggers causing scroll bars that made using these elements difficult.

Two participants reported that the lack of a unit or percentage that, in their concrete case, caused them confusion about how to use scaling triggers. One participant mentioned that they were unsure about how to apply the group size constraint correctly. They mentioned a lack of clarity as to how constraints are synced between policies of the same target group. Another participant mentioned confusion that the name “absolute adjustment” is confusable with the “step adjustment”. Where the former sets a value for the amount of elements of a target group and the latter adjusts the amount of elements of a target group by a fixed value.

5.2.3 Future additions

In the following, I summarize notable future additions suggested by the participants.

An option for an automatic layout was suggested. Furthermore, a textual syntax or a textual summary of what scaling actions the scaling policies perform were desired. Swapping positions for scaling trigger and adjustment type, so that they appear in reading direction, was stated. Units for the scaling triggers were proposed to be added. Eclipse was mentioned to be something to consider alternatives

to for a production version of the graphical editor. Better explanations were wished for. Finally, the ability to have multiple scaling rules e.g., based on the same trigger type in a single scaling policy were suggested.

5.3 Discussion

The hypothesis I have been trying to prove or disprove is that the graphical editor helps with the creation, editing and understanding of scaling policies.

First, I present the arguments that speak for accepting the hypothesis.

The strongest argument for accepting the hypothesis is that at least eight of nine participants found the visual representations and design cues as helpful or appropriate. The fact that a group of participants with varying competences, ranging from students studying software engineering through experts in software architectures such as microservices, strongly agreed on the appropriateness of the design strengthens this argument.

In addition, participants saw multiple potential applications in the software development lifecylce with tendencies towards design, documentation, deployment and operations. This speaks towards the helpfulness of the graphical editor because participants found a broad spectrum of potential applications i.e., four out of six development phases were voted for by over 75% of participants.

Furthermore, when explicitly asked whether participants found the graphical editor *helpful* when working with scaling policies, six out of nine participants i.e., two thirds, tended towards a strong helpfulness while the other three participants chose a neutral stance between extra helpfulness and lack thereof.

Now, I will present the arguments against accepting the hypothesis.

On the other hand, one out of nine people i.e., roughly eleven percent, found parts of the graphical editor not helpful. This can be interpreted as an indicator that, in general, the graphical editor does not help everyone with the creation/editing or understanding of scaling policies. However, I must note that in order to assess this properly, another study, perhaps even with participants from the more general population, would be necessary.

In addition, three participants i.e., one third of participants, perceived the graphical editor as only somewhat helpful, scoring a 3 on a 5-point Likert scale on helpfulness when working with scaling policies. One can argue that if one third of users did not find the graphical especially helpful when working with scaling policies, the overall helpfulness for a broader audience is questionable, at least.

5.3.1 Conclusions

Based on the aforementioned arguments, I draw the following conclusions. It has been proven that the visual representations used by the graphical editor have been accepted as helpful by the study's participants. Furthermore, at least two thirds of participants gave the graphical editor a score in favor of the helpfulness of the graphical editor when working with scaling policies. In addition,

the fact that participants were able to see a use of a graphical editor for scaling policies in multiple phases of software developments also suggests a perceived usefulness of the graphical editor in a more general sense.

Therefore, the graphical editor does help with the creation and editing of scaling policies, at least for a majority of people asked in this case study. With regard to the understandability, I argue that the visual representations used, aided with the understandability of scaling policies for almost all the participants as they found the representations appropriate.

5.4 Threats to Validity

One inherent risk of allowing participants of a study to try out a software prototype is the risk of them running into a technical problem. To mitigate this as best as possible, participants have been provided with a Linux-based virtual machine with the graphical editor fully set-up and tested to be working.

Another potential risk is the order of questions affecting the answers participants' select [SH95]. To reduce this risk, the questions are being showed in a random order (per section).

Furthermore, the questions are subjective which in itself is not a risk. However, it is a fact that asking a subjective question yields a subjective answer dependent on the participant, their experience et cetera. As a result, conducting the same survey even under as identical as possible of circumstances will likely yield varying responses i.e., the results are not practically reproducible.

Albeit an obvious risk, the lack of experience on my part as to how to conduct a survey is a risk. To reduce it as best I could, I consulted frequently with my supervisor on decisions and details regarding this survey. I also feel it important to mention that, only by facing the risk of lack of experience, I myself am able to "reduce" this risk for future studies I might conduct.

Another potential threat to validity is a result of providing participants with a partial overview of the graphical metamodel and editor. As a result, it is possible that e.g., model elements relevant for the task given to participants might have been explained better than parts expected to be less relevant to the task. To reduce this risk, I intentionally provided information about scaling policies, target groups and three selected instances of scaling triggers, adjustment types and constraints. Out of this selection, some elements were expected to be used for the task while others were not intended to be needed by participants. Therefore, reducing the risk of favoring more relevant explanations over less relevant explanations for the given task.

Finally, since all participants took part in the same live session, there was a possibility of participants influencing each other e.g., via comments voiced during the session. With regard to this, one such comment was actually voiced at one point during the session by one of the participants. To mitigate this, it is best to ask participants to not make comments aloud during the session. If sessions are or rather had been performed as one-on-one sessions, the same risk still exists outside the sessions. Again, in this case, asking participants not to discuss their experience with other people for the duration of the study can, hopefully, reduce the risk.

5.5 Resulting Implementation Adjustments

Besides evaluating and validating my work, I also used the feedback provided by the participants to make final adjustments to improve the prototype.

As one participant and my supervisor suggested, the position of the scaling trigger and adjustment type have been swapped. As a result, the scaling trigger is now to the left of the scaling policy and the adjustment type is to the right of the policy. Therefore, trigger and adjustment type are shown in "reading direction". In other words, when a user "reads" a scaling policy from left to right, the trigger comes first on the left and then the adjustment is showed on the right.

Another small adjustment is that the "target group connector" is now functioning bi-directionally. As a result, it is possible to either click the target group first and then the scaling policy and vice versa.

In addition, since several participants ran into scroll bars when placing adjustment types and scaling triggers, the placeholder rectangles have been increased in size. The height of the entire scaling policy placeholder is now 140 pixels (px) high, instead of 120px. Both the placeholder boxes on the sides of the scaling policy are now 100px wide instead of the previous 80px. The overall dimensions of a scaling policy are now 290px (width) times 140px (height) plus, of course, any constraints added on top. As a result, I expect future users to be confronted less often with scroll bars than before.

Lastly, several participants mentioned their inability to add constraints to scaling policies. This can most likely be attributed to the fact that in order to add a constraint to a policy, one has to place the constraint on the invisible top edge of the scaling policy. This is a design decision made due to lack of time at the end of the implementation phase.

6 Conclusion

6.1 Summary

In conclusion, as a contribution of this thesis, scaling policies can now be created and worked with using a graphical editor which was previously not possible. In doing so, I have addressed the research question as to how graphical editors can help the software architect with the creation and understanding of scaling policies.

First, I gathered information about the contexts where scaling policies are applied. These, namely service oriented architecture (SOA), microservices and cloud native applications (CNA), I introduce in Chapter 2. Furthermore, I have informed myself and provide an overview of the metamodel of scaling policies.

In addition, I discuss related work in Section 2.2 where I go over Argon [SIA17], a graphical modeling tool to specify the provisioning of cloud resources, AjiL [RSW+20], a graphical toolkit for model-driven microservices and Stratus ML [HT15], a modeling framework for cloud applications.

To create the graphical editor, I have designed the model and editor on best practices for visual notations [Moo09] in Chapter 3. After creating a first draft, I gathered feedback from experts on what they liked/disliked and what they would like to see improved. The feedback, in conjunction with input from my supervisor, has been used to create a second draft.

The second draft, has then been implemented using Eclipse Sirius (see Chapter 4). In addition, in Section 4.3 I also discuss how changes to the Ecore metamodel can be handled and how e.g., constraints specified in the Ecore metamodel could be realized with the graphical editor.

To evaluate the graphical editor, I have performed a study with nine participants from the industry and academia where they have been asked to experiment with the graphical editor and to then give feedback on their experience using a questionnaire (see Chapter 5). In summary, at least eight out of nine participants have found the graphical metamodel to be appropriate. Furthermore, two thirds of participants have voted in favor of the graphical editor being especially helpful for working with scaling policies.

The study showed that overall the graphical editor does indeed help with the creation and editing of scaling policies.

6.2 Benefits

As of now, using the graphical editor for scaling policies can benefit any software architect who wants to design a scaling model for an application using e.g., microservices, cloud-native applications or a service oriented architecture.

For instance, for software architects used to using graphical notations the use of the graphical editor can help them to address scaling concerns faster and in a more understandable way than using the tree editor. In turn, by saving time, the graphical editor can even help reduce development costs. Furthermore, the graphical representation can also help with communicating scaling concerns with other stakeholders e.g., clients or project managers, to make the topic more approachable to them as well.

The findings of the evaluation study show that a graphical editor for scaling policies can benefit e.g., in the design phase of a software system. It can also be used to document and communicate decisions regarding scaling of a system e.g., in a DevOps use case, and it might also be used in the future during the operation of a software system for instance, to monitor scaling aspects of a software system. Furthermore, most participants found such a graphical editor to be useful for deploying software systems i.e., where the implemented system is put into production with scaling concerns addressed.

6.3 Limitations

The developed prototype of the graphical editor is limited to creating and editing scaling policies. It is not a general purpose graphical editor for modeling scaling concerns of cloud architectures. Neither does this thesis aim to go beyond scaling policies in that regard.

Another remark to keep in mind is that the graphical editor was evaluated in a setting of software engineers. Therefore, the results of the evaluation are, of course, limited to represent people with a software engineering or computer science background. In other words, one cannot expect e.g., to put a person completely unfamiliar with scaling policies or SOA/microservices/CNA in front of the editor and expect them to be able to create high-quality scaling models.

Another limitation is that the graphical editor does not enforce or check constraints that can e.g., be specified using OCL in the Ecore model. How constraints can be realized however, I have discussed in Section 4.3.3.

Finally, a more general limitation is the fact that the implemented graphical editor is a prototypical implementation. For instance, the rectangular boxes that serve as placeholders for triggers and adjustment types and the implementation of policy constraints as bordered nodes that have to be placed on the edge of the scaling policy container are not optimal with regard to usability. These trade-offs were a combination of relying on Eclipse Sirius in conjunction with the second draft and later adjustments of the graphical metamodel. These trade-offs have been made to reduce the risk of ending up without a fully working prototype. However, as a result, although fully functional, the practical usability of the graphical editor could be further improved.

6.4 Lessons Learned

One lesson I learned during the feedback session on the first draft was that if I ask three people for their feedback, I might get three different opinions that can go in different directions. For instance, when asked about the shapes used in the draft, one participant preferred less different shapes while the other two found the different shapes to be to their liking.

Another interesting takeaway from the feedback is that I got the impression that, especially in design, it is impossible to please everyone. Even if e.g., 90% of people like a graphical model better, there is still going to be people who will prefer something else e.g., a textual model.

Something I learned after the evaluation session was that it is important to provide participants with well-balanced options for answers. For example, to also ask for positive impressions when asking about negative impressions of e.g., a software artifact. While I argue that overall this was done in the questionnaire, at least one participant later informed me that they would have liked an open question on what they liked, in addition to being asked what problems they might have encountered.

6.5 Future Work

My first suggestion for futures work is to make use of color, to further enrich the graphical editor. Again, color is the most cognitively effective visual variable [Loh93]. Therefore I propose that when the graphical editor will be extended with functionality for analysis purposes, for example, that colors are used to e.g., visualize how often triggers fired et cetera.

Another suggestion is to specify units (e.g., percentages or units for absolute values) in the Ecore metamodel for scaling triggers. It was remarked by participants during the evaluation questionnaire that they were unsure how to correctly specify the given requirements with regard to e.g., how to specify a percentage on a scaling trigger correctly. When the units for scaling triggers are clarified, I also suggest extending the graphical editor appropriately. For instance, to allow the user to select the unit and then specify the desired value.

Furthermore, as the metamodel is specified now, thrashing and group size constraints are specified on scaling policies. It showed that this can lead to situations where constraints might not be synchronized (per target group) or even be specified in conflict. I propose addressing this e.g., by specifying group size and thrashing constraints directly "on" the target group because the two types of constraints operate directly on target groups and are therefore independent of individual scaling policies affecting the target group.

In addition, one participant found the naming of the "absolute adjustment" confusing because it could be interpreted as a "step adjustment". I propose considering an alternate name in the Ecore metamodel for the absolute adjustment, perhaps to something along the line of a "goal value adjustment" to make the two adjustment types more distinct and therefore more understandable.

With regard to empirical validation, I suggest performing a similar evaluation study with participants from other stakeholder groups of software projects. In a perhaps distant future, validating the graphical editor by stakeholders such as business managers and potential clients from the industry

would allow to put the graphical editor in a broader perspective as to how helpful with model creation and understanding it is. Therefore, increasing the research value of the graphical editor for scaling policies even further.

Moreover, handling constraints and validating them is a current limitation of the graphical editor that should be addressed in future work which I discussed by example in Section 4.3.3. Remedying the partially limited usability e.g., with regard to the attachment of constraints or the rectangular placeholders for triggers and adjustment types are also noteworthy future work.

Finally, I suggest automating updating the graphical metamodel when the Ecore metamodel is changed. Based on my discussion in Section 4.3.2, it is not feasible to try to automate all types of changes to the metamodel. However, a first step towards this automation can be automating renames of metamodel elements. Furthermore, it can be considered to also automate removals from the metamodel. The way the graphical metamodel is designed, the automation should also cover reflecting the changes in existing SPDs. Therefore, I suggest the ability to update the graphical metamodel and existing models. Lastly, an automated generator for documentation provided through the Ecore metamodel for the graphical metamodel is potential future work that can aid users understand scaling policies even better.

Bibliography

[Ayt+17] S. Aytac et al. "Using Color Blindness Simulator During User Interface Development for Accelerator Control Room Applications". In: *International Conference on Accelerator and Large Experimental Control Systems*. 2017, pp. 1958–1968 (cit. on p. 14).

[BCR94] V. R. Basili, G. Caldiera, H. D. Rombach. "The goal question metric approach". In: *Encyclopedia of software engineering* (1994), pp. 528–532 (cit. on p. 41).

[BFGÖ21] K. Baričová, C. Friedel, E. Grande, Y. Özbey. "A Metamodel for defining PCM-based Scaling Policies for Cloud Applications". In: (2021) (cit. on p. 29).

[CG12] J. Cabot, M. Gogolla. "Object constraint language (OCL): a definitive guide". In: *International school on formal methods for the design of computer, communication and software systems*. Springer. 2012, pp. 58–90 (cit. on p. 31).

[GBU08] T. Goldschmidt, S. Becker, A. Uhl. "Classification of concrete textual syntax mapping approaches". In: *European Conference on Model Driven Architecture-Foundations and Applications*. Springer. 2008, pp. 169–184 (cit. on p. 10).

[Gee05] D. Geer. "Eclipse becomes the dominant Java IDE". In: *Computer* 38.7 (2005), pp. 16–18 (cit. on p. 19).

[Hit02] S. Hitchman. "The details of conceptual modelling notations are important-a comparison of relationship normative language". In: *Communications of the Association for Information Systems* 9.1 (2002), p. 173 (cit. on p. 6).

[HT15] M. Hamdaqa, L. Tahvildari. "Stratus ML: A layered cloud modeling framework". In: *2015 IEEE International Conference on Cloud Engineering*. IEEE. 2015, pp. 96–105 (cit. on pp. 10, 47).

[JKCP15] A. Joshi, S. Kale, S. Chandel, D. K. Pal. "Likert scale: Explored and explained". In: *British journal of applied science & technology* 7.4 (2015), p. 396 (cit. on p. 42).

[KHB21] F. Klinaku, A. Hakamian, S. Becker. "Architecture-based Evaluation of Scaling Policies for Cloud Applications". In: *International Conference on Autonomic Computing and Self-Organizing Systems*. 2021, p. 3 (cit. on pp. iii, v, 1, 5).

[KQ17] N. Kratzke, P.-C. Quint. "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study". In: *Journal of Systems and Software* 126 (2017), p. 13. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2017.01.001. URL: https://www.sciencedirect.com/science/article/pii/S0164121217300018 (cit. on p. 5).

[LMO95] G. L. Lohse, D. Min, J. R. Olson. "Cognitive evaluation of system representation diagrams". In: *Information & Management* 29.2 (1995), pp. 79–94 (cit. on p. 7).

[Loh93] G. L. Lohse. “A cognitive model for understanding graphical perception”. In: *Human-Computer Interaction* 8.4 (1993), pp. 353–388 (cit. on pp. 6, 49).

[LS87] J. H. Larkin, H. A. Simon. “Why a diagram is (sometimes) worth ten thousand words”. In: *Cognitive science* 11.1 (1987), pp. 65–100 (cit. on p. 6).

[Mil56] G. A. Miller. “The magical number seven, plus or minus two: Some limits on our capacity for processing information.” In: *Psychological review* 63.2 (1956), p. 81 (cit. on pp. 7, 13).

[MM03] R. E. Mayer, R. Moreno. “Nine ways to reduce cognitive load in multimedia learning”. In: *Educational psychologist* 38.1 (2003), pp. 43–52 (cit. on p. 7).

[Moo09] D. Moody. “The “physics” of notations: toward a scientific basis for constructing visual notations in software engineering”. In: *IEEE Transactions on software engineering* 35.6 (2009), pp. 756–779 (cit. on pp. 6, 7, 13, 14, 47).

[Obe95] J. Oberlander. “Grice for graphics: pragmatic implicature in network diagrams”. In: *Information design journal* 8.2 (1995), pp. 163–179 (cit. on p. 7).

[Pet95] M. Petre. “Why looking isn’t always seeing: readership skills and graphical programming”. In: *Communications of the ACM* 38.6 (1995), pp. 33–44 (cit. on p. 7).

[PL03] R. Perrey, M. Lycett. “Service-oriented architecture”. In: *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.* 2003, pp. 116–119. DOI: 10.1109/SAINTW.2003.1210138 (cit. on p. 4).

[RSW+20] F. Rademacher, J. Sorgalla, P. Wizenty, S. Sachweh, A. Zündorf. “Graphical and textual model-driven microservice development”. In: *Microservices*. Springer, 2020, pp. 147–179 (cit. on pp. 9, 10, 47).

[Rup10] N. B. Ruparelia. “Software development lifecycle models”. In: *ACM SIGSOFT Software Engineering Notes* 35.3 (2010), pp. 8–9 (cit. on p. 42).

[SH95] N. Schwarz, H.-J. Hippler. “Subsequent questions may influence answers to preceding questions in mail surveys”. In: *Public Opinion Quarterly* 59.1 (1995), pp. 93–97 (cit. on pp. 41, 45).

[SIA17] J. Sandobalin, E. Insfran, S. Abrahao. “An infrastructure modelling tool for cloud provisioning”. In: *2017 IEEE International Conference on Services Computing (SCC)*. IEEE. 2017, pp. 354–361 (cit. on pp. 7, 47).

[SIA20] J. Sandobalin, E. Insfran, S. Abrahao. “On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven Versus Code-Centric”. In: *IEEE Access* 8 (2020), pp. 17734–17761 (cit. on p. 8).

[SRH17] D. Shadija, M. Rezai, R. Hill. “Towards an understanding of microservices”. In: *2017 23rd International Conference on Automation and Computing (ICAC)*. IEEE. 2017, pp. 1–6 (cit. on p. 4).

[SWR+18] J. Sorgalla, P. Wizenty, F. Rademacher, S. Sachweh, A. Zündorf. “AjiL: enabling model-driven microservice development”. In: *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. 2018, pp. 1–4 (cit. on pp. 9, 10).

A Appendix

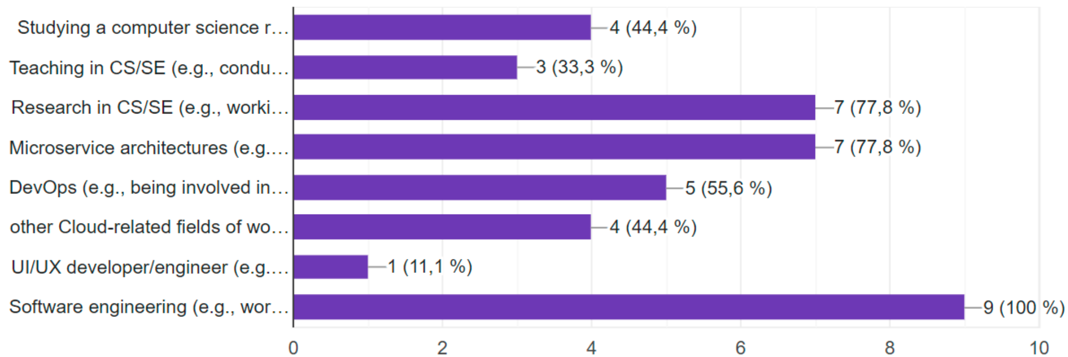
A.1 Summary of Study Results

On the following pages, the results of the performed evaluation study can be found. The study is discussed in Chapter 5 Evaluation.

The 5-point Likert scales have been annotated with the meanings of both ends of the scale. The answers to the two free-text questions have been aggregated by hand but kept unchanged otherwise. Individual answers are separated by dashed lines.

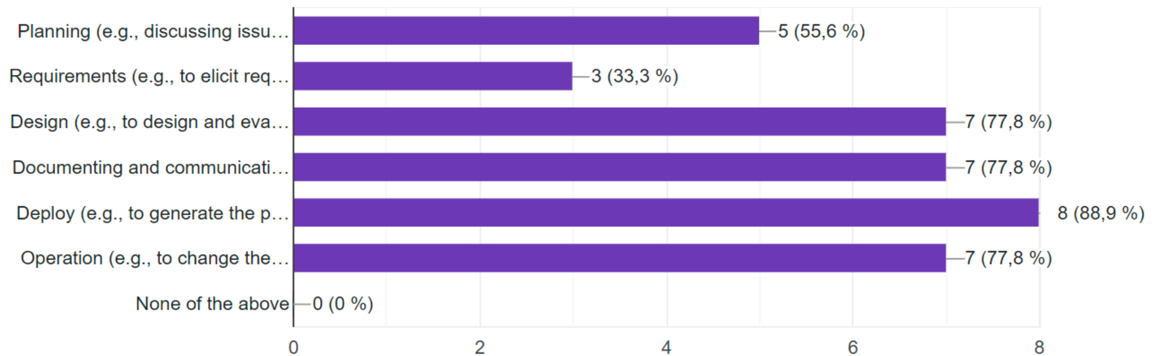
Out of the following selection, please select the competences that you feel you are capable of.

9 Antworten



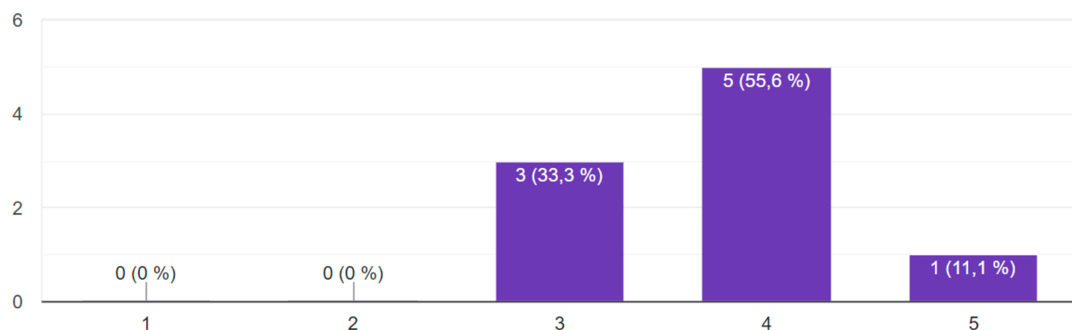
Beyond the current limitations of the tool, at your company and/or context and based on your experience in which phases of the software development lifecycle do you see the use for graphically modelling the scaling policies?

9 Antworten



Did you perceive the graphical editor as helpful for you when working with scaling policies?

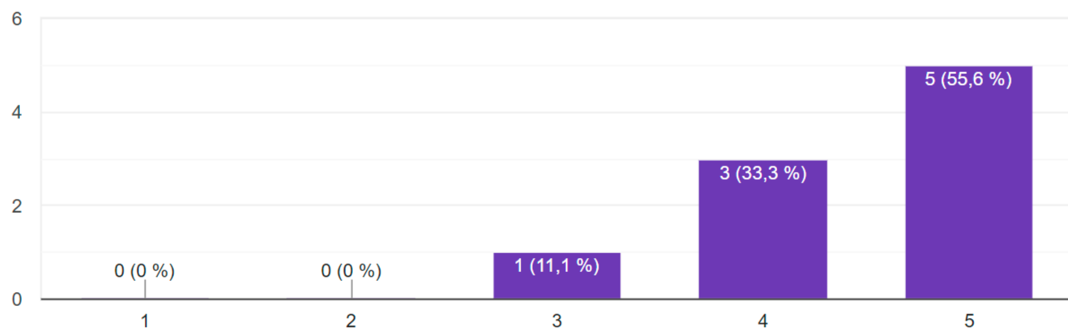
9 Antworten



(1: not helpful, 5: very helpful)

Did you have enough time to solve the given task?

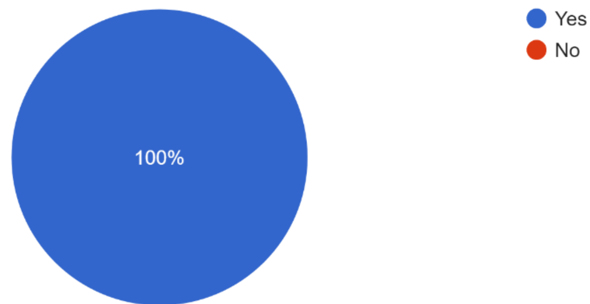
9 Antworten



(1: too little, 5: too much)

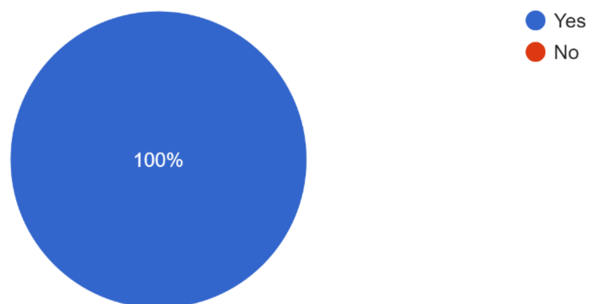
Does the visual representation of the scaling policy seem appropriate?

9 Antworten

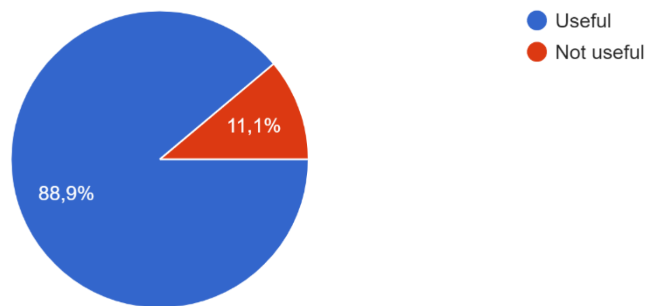


Does the visual representation of the adjustment type seem appropriate?

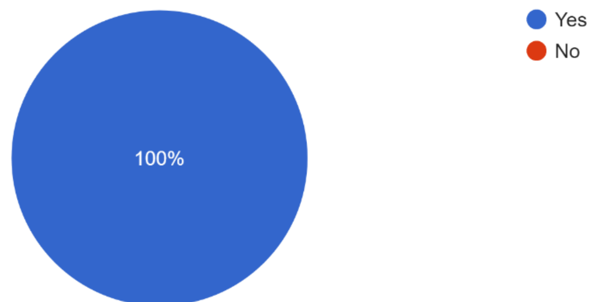
9 Antworten



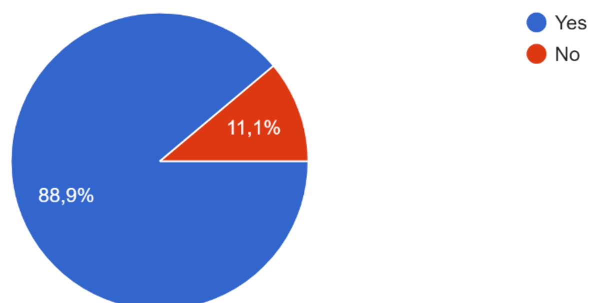
Do you perceive the direction-dependent visual representation of adjustment types as useful?
9 Antworten



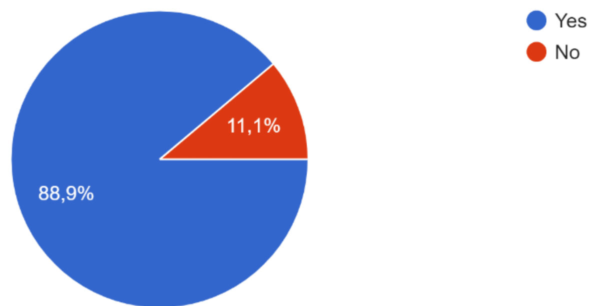
Does the visual representation of the scaling trigger seem appropriate?
9 Antworten



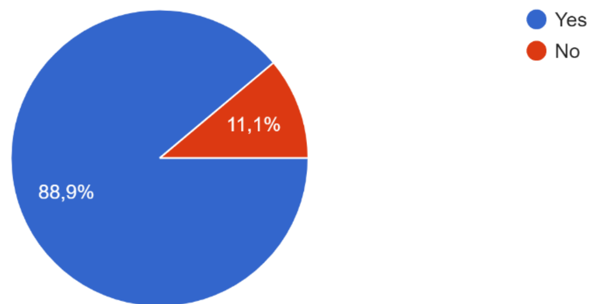
Does the visual representation of the policy constraints seem appropriate?
9 Antworten



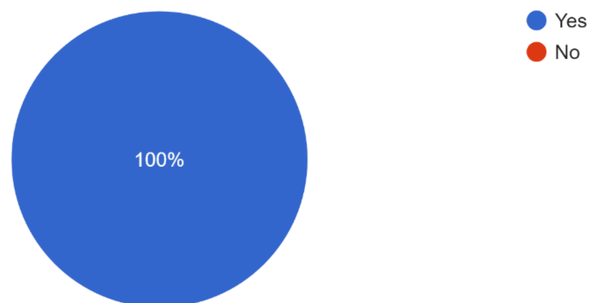
Does the visual representation of the target group seem appropriate?
9 Antworten



Do you perceive the gray placeholders around scaling policies as helpful?
9 Antworten



Did you have any problems when using the graphical editor?
9 Antworten



Free text question: Please describe the problem(s) you had.

- could not add the constraint hexagon to the new policies.

- could draw connector only one direction (policy to target group) but not the other way round.

- the entire graphical elemts "jump" around, after i insert the adjustment types / triggers. Probably because the size of thr triggers do not match the size of the place holders.

Also the placeholder's compartments are apparently to small for the actual elements, thus the actual elements are displayed with these scorlling bars at the side.

- until i read this questionaire, i did not understand the meaning of the adjustement triangles' orientation. I was also very confused when the triangle suddenly changed it's orientation after i entered a negative step size. And it was unclear, whether i should enter a negative value to decrease, or whether i should enter a positive value and tick some check box to make it decreasing.

- at the relative adjustment, it was at first unclear whether i should enter 100% because i want to increas _by_ that amount, or whether i should enter 200% because i want to increase _up to_ that amount, but the tiny "+" next to the value cleared that up and i entered the 100% percent.

- meaning of "Min Adjustment Value" in "Relative Adjustment" was (and is) unclear to me.

- connectors tend to dissapear when i select the background canvas

- not a problem, but i disliked that the two server icons in the target group icon are transparent, such that you can see the background server through the forground serve. it looks a bit weird. would look better if the server icon had no transparencies inside the icon.

Disclaimer:

- did not use VM, but directly install in obeo designer.

----

* I was not able to add constraints on any policy, even on the already given ones.

A little video about the usage (how to use the palette, how to find the properties to edit (properties view was not activated automatically at the beginning)) would have been very helpful

----

When creating a new policy, just an empty box appeared. Moving it did not solve it. I was only able to add constraints, but no triggers or adjustments via the visual editor. Linking did not work either. Finally, I used the textual editor to add the elements.

When removing policies, they were deleted from the visual editor, but were still present in the (textual) model.

When setting the thresholds for the triggers, I was not sure what the unit is. Is 80% utilization a 80.0 or a 0.8?

I had problem with disappearance of the connector. When I UN-checked the active checkbox in scaling policy, it appeared again. In addition, I could not put group size constraint in the respective gray box. An editor issue? or maybe it was only me

---

Lacking descriptions when hovering over the help (?) icons, Adjustment Icon created scroll bars around it making it unreadable when not moved manually, no idea how to add constraints to the second policy, as I was unable to drop one on the placeholder

---

The editor is completely buggy. Shapes jump around. Lines are hiding when clicking on the background canvas. I could not add the policy constraint to the policy because it was blocked by the editor no matter how I tried to add it. The policy adjusted itself by adding features (trigger, adjustment type) which were then shown outside the rectangle having unnecessary scroll bars. The line to connect the policy with the target group is only unidirectional when adding it, i.e., it can only be added from target group to policy. I would prefer connecting them by creating the policy directly out of the target group, e.g., right click -> add policy.

---

1. Percentages/relative values were designed inconsistently (e.g. in the relative adjustment type I have to enter percentage values e.g. 80% but in the scaling triggers I (assume) I have to use relative values i.e. 0.8 = 80%). This should be explained somewhere (the explanations in the question mark tooltips are no help)

2. After some time I was unable to place any Constraints i.e. I dragged them onto their respective fields but nothing happened, no matter if it was a new scaling policy or not

3. In general I find it confusing how I am supposed to apply multiple scaling policies to the same service. For example, for the CPU task I made two separate scaling policies, one for the high-usage upscaling and one for the low-usage downscaling. But does the constraint from one policy also apply to all other policies attached to the same service? Or do I need a separate constraint for each policy? And how are they synced? As a further example, I was unsure how to do the RAM requirement to ensure that the instance started during high-load was the same being shut-off during down-scaling (as the requirement implied). I, again, just made two separate policies and hoped that the constraints would sync up i.e. if I have a 1..2 group constraint on one policy, the other policies attached to the Account management service will also respect that constraint.

4. For Task 4, I first tried to use an "AbsoluteAdjustment" to reduce the number of instances by one until I realized that I should have used the "StepAdjustment". Maybe rename the "AbsoluteAdjustment" to something like "AbsoluteValue" to imply that you are not "adjusting" but instead "setting" the number of units

5. Nor sure if this is a limitation of the framework, but I sorely missed the ability to pan the view around. Using the scrollbars was really fiddly and there isn't enough margin around the edges e.g. the constraints of the initial scaling policies almost touch the edge of the window.

I could not drop Constraints into target.

Free text question: Is there anything that you would like to see added to a future graphical editor for scaling policies?

----

fix the problems i pointed out in the previous question :)

in addition, i would like a question in the qeustionaire where i can point out positive things.

I would have expected the trigger at the left and the adjustment on the left of the policy (in reading direction).

Units for the trigger.

Some symbols where hard to understand by intuition, e.g., target group and group size look too similar. The steps do not accurately represent the adjustment since actually only one step is performed. In general, the choice of shapes felt rather random, requiring to learn and remember their meaning.

In general, the Eclipse Editor felt really clumsy to use (although I understand the use of it for a research project). For production I would suggest to use different technology or invest time into usability.

The experienced bugs made it impossible to solve the tasks solely with the visual editor.

Maybe a combination of textual and graphical representation is more helpful. Something like a pseudo code mixed with graphical icons.

Autolayout or non-manual but useful static layout

A view for a textual syntax. Applying the policy to cloud resources. Better UX.

- Being able to set both up- and down-scaling rules in the same policy e.g. having one policy for all CPU related rules for a given service

- Better explanations for the property fields (i.e. what do they do, what type are the values (see my percentages feedback point), maybe an example)

- Being able to generate a textual summary of my scaling policies (i.e. If the "XY USAGE" of service "Z" goes ABOVE XX percent, the number of units will INCREASE by X, with a maximum of X and a minimum of Y units"). While the editor is lovely for creating policies, I think if you simply look at the result to understand the policies, it can be confusing because of all the icons

----

A.2 Screenshots

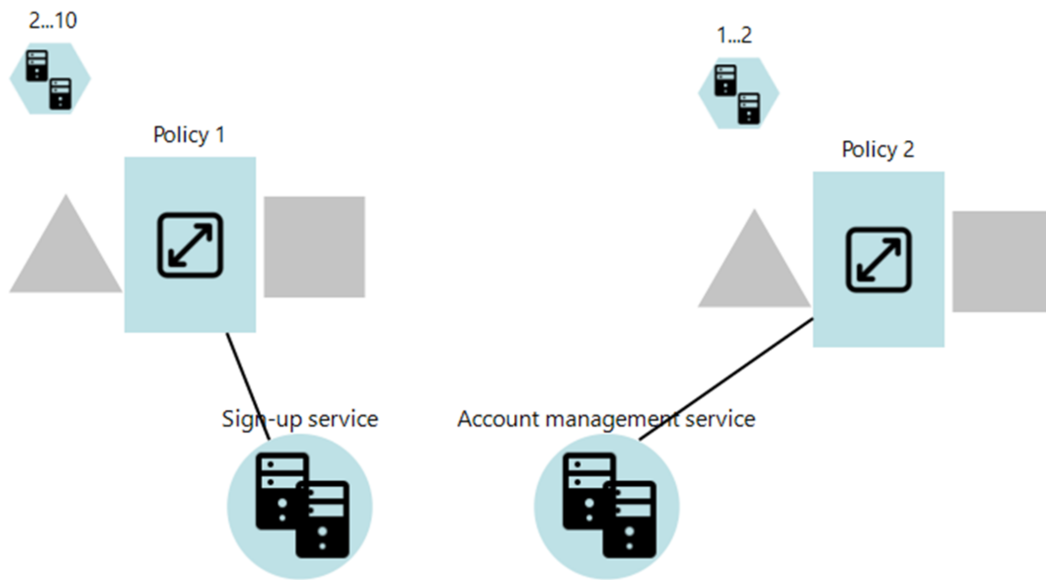


Figure A.1: Given example for the evaluation study (as provided to participants)

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature