Scripted Step-based Visualizations: A Pilot Study

Aleksi Lukkarinen[a], Lassi Haaranen[b] and Lauri Malmi[c]
Aalto University, Department of Computer Science, P.O. Box 15400, FI-00076 Aalto, Finland

Keywords: Visualization, Client, Server, Messaging.

Abstract: Software visualization has numerous educational applications that focus on illustrating code, structure, behavior, and/or evolution of software. However, there are few available solutions that (1) illustrate arbitrary high-level concepts according to the scripts specified by the instructor and (2) can be easily integrated into various existing learning materials. To evaluate the feasibility of such a tool, we developed a proof-of-concept system that can be used as a part of many web-based course materials and that is supported by Acos content server. As a pilot study, we introduced the tool in an introductory web development course to visualize messaging between clients and servers. We analyzed the usage log data and student questionnaire data and the results were mostly positive, which prompts for further research on the subject.

1 INTRODUCTION

Software visualization is a broad research area that aims to develop useful visual presentations of, for instance, program code, structure, behavior, and evolution of computer software. Many applications of software visualization are targeted to professional software developers and system administrators who seek to investigate and visualize both statistics of large-scale software projects and relationships between their parts. On the other hand, educational software visualization has been actively researched in the fields of *program visualization* and *algorithm visualization*, where the applications are mainly targeted for introductory courses in programming, data structures, and algorithms. These are highly important target areas; however, computer science also deals with higher-level abstract concepts presented with many other types of visualizations, such as various process diagrams and *Unified Modeling Language* diagrams. Such diagrams are often static, and they are typically created and manipulated with generic drawing tools or tools that support certain specific diagram types. In this paper, we explore this more abstract domain for *dynamic* computer science visualizations.

We formulate our *research goal* as follows: We seek to develop a free and open-source software platform that empowers teachers to create visualizations of arbitrary high-level concepts in computing education, such that

- have interactive elements based on teacher-made scripts
- are easily integrable into web pages and learning management systems
- enable logging of students' activities
- are shareable between colleagues
- are versionable using regular version control systems
- can be created with a relatively low learning curve, and
- can be augmented and tailored to specific use cases.

According to the above goal, we developed a pilot application for a web development course: An interactive widget that visualizes messaging and the related processes between an Internet browser and a web server. In this paper, we present our visualization platform and discuss the teacher's point of view in using it—the advantages that the platform offers for creating novel interactive learning content. To support our proof-of-concept, we also present the results of a small pilot evaluation study, where we posed the following research questions:

RQ 1. How do students use the visualizations?

RQ 2. How do students experience the visualizations?

The rest of this paper is structured as follows: In Section 2, we discuss work related to our visualization

[a] https://orcid.org/0000-0002-3827-6243
[b] https://orcid.org/0000-0002-6500-6425
[c] https://orcid.org/0000-0003-1064-796X

Lukkarinen, A., Haaranen, L. and Malmi, L.
Scripted Step-based Visualizations: A Pilot Study.
DOI: 10.5220/0010454502400247
In *Proceedings of the 13th International Conference on Computer Supported Education (CSEDU 2021) - Volume 2*, pages 240-247
ISBN: 978-989-758-502-9

platform. We continue by discussing the platform and its usage in Section 3 as well as presenting the evaluation study and answering our research questions in Section 4. Finally, we conclude our work and present ideas for future work in Section 5.

2 RELATED WORK

In this section, we discuss related research and multiple existing software tools and contrast them with the set of requirements given in the Introduction.

Computer science education has a long tradition of using visualizations to clarify concepts and ideas for students: A well-known example of early educational applications of software visualization—turtle graphics and physical robots drawing with pens Papert (1980)—takes us back to the end of the 1960s.

Typical educational applications of software visualization include illustrating data structures as well as executing individual algorithms or entire programs, for instance, line-by-line, by the logical step, or as a whole (e.g., Shaffer et al., 2010; Sorva et al., 2013). Most of these solutions have been developed especially as specialized visualizations, but often other approaches are used for software visualization purposes, as well. Such approaches include (1) various static diagrams (e.g., flow, entity-relationship, and those of Unified Modeling Language), (2) charts (bar, line, pie, etc.) for statistical information of the software, (3) graphs for relationships, as well as (4) different forms of animations with varying options for interacting with the visualization. Three-dimensional models and virtual reality have been researched, as well (e.g., Averbukh et al., 2019; Milne and Rowe, 2004).

General-purpose Visualizations. Whereas the above software usually targets a single specific use case, our visualization platform is not tailored for any specific domain. Rather, it is a general-level solution for creating visualizations for the Internet browser. Obviously, many other more or less similar approaches are available. Probably the most primitive option would be to code a visualization as a series of *Hypertext Markup Language* (HTML) pages and related resources. Graphics could be static images, animated image files,[1] and video files. Furthermore, JavaScript libraries[2] can be used to create animation and dynamic visualizations directly to web pages. Visualizations as animated images and videos can be created and converted to suitable formats with regular graphics applications and video editors.

Developing visualizations by coding the necessary infrastructure from scratch or by tapping into existing libraries, however, requires significant effort: One must learn the necessary technologies and then apply them, which takes time, might be tedious, or might not even be worth the effort. An alternative option is to exploit technologies that have been designed for authoring learning material. Unfortunately, several solutions, which supported Internet browsers and were popular in the past, have become obsolete[3] during the recent years, partly due to emerging newer technologies, such as HTML5. Still, many other options are available.

One possibility for delivering visualizations that have low interaction would be to create slideshows using conventional presentation graphics applications, such as *Google Slides*, *LibreOffice Impress*, *Microsoft PowerPoint*, and *Apple Keynote*. The native file formats of these applications—or, for instance, *Portable Document Format* conversions of the slide sets—can be linked on web pages to be downloaded and viewed outside the Internet browser. Furthermore, PowerPoint for Microsoft Windows allows exporting presentations to both video files and animated *Graphics Interchange Format* files, that can be made available in a similar fashion.

Presentations can also be embedded to web pages. For instance, the cloud version of PowerPoint enables users to generate HTML chunks for embedding presentations to web pages in IFRAME elements. In addition, lightweight solutions, such as *WebSlides*, exist to aid creating slideshows by manual coding, and slide sets can be shared via (social) sharing platforms, such as *Scribd*, *SlideShare*, *SlideServe*, and *Speaker Deck*. For embedding videos, (social) video sharing platforms, such as *Google YouTube* and *Vimeo*, might also be helpful.

The above kinds of presentation graphics applications are available for most educators, but it might be impossible or impractical to customize the results, for instance, to implement a detailed usage logging facility for collecting learning analytics about students' usage of the generated dynamic learning resources. In addition, many of these tools are com-

[1] Examples: Graphics Interchange Format (GIF), Animated Portable Network Graphics (APNG), WEBP, and Free Lossless Image Format (FLIF).

[2] Examples: General-purpose libraries, such as jQuery and Anime.js, as well as more specialized ones, such as Chart.js and D3.js.

[3] Examples of obsolete authoring technologies: Adobe (FutureWave/Macromedia) Flash (Adobe Inc., 2021), Adobe (Macromedia) Shockwave (Adobe Inc., 2019), Microsoft ActiveX (in Internet Explorer 11, e.g., Microsoft Corporation, 2021a), Microsoft Silverlight (Microsoft Corporation, 2021b), Oracle (Sun) Java Applets (Oracle Corporation, 2020), and SumTotal Systems (Asymetrix) Toolbook (e.g., SumTotal Systems, LLC, 2021).

mercial whereas, in some contexts, using commercial solutions can be out of the question due to reasons such as principles, legislation, or a lack of funding.

A more advanced option for non-specialized visualization tools is using various *eLearning authoring solutions*.[4] They can enable creating non-linear course materials with multiple exercise types to be transferred to various learning management systems. The results often resemble slideshows or videos but are more interactive than conventional slide decks. However, also these solutions carry the above complications, including the time and effort necessary to learn the tool itself. In addition, they can be more expensive than their simpler alternatives, which is why free and open-source options might be preferred.

Protocol Visualizations. Whereas our visualization platform is not tailored for any subject area, we piloted it in a case study concerned with visualizing the concept of communication between web browsers and web servers. Tools that visualize protocols at a similar abstraction level than the visualizations in our pilot study include *NetPrIDE* (Crescenzi et al., 2005) for network protocols and *GRASP* (Schweitzer et al., 2006; Schweitzer and Brown, 2007) for security protocols. In addition, there are numerous professional-level network analysis and visualization tools (e.g., Awodele et al., 2015; Fuentes and Kar, 2005; Hall et al., 2003) that, unfortunately, do not lend themselves well for illustrating high-level concepts for novices due to their complexity as well as different design goals and features.

Crescenzi and Innocenti (2002) have suggested a taxonomy specifically focused on tools for visualizing network protocols. However, as our need for visualizations is of more general nature instead of specializing (and being limited), for instance, to network protocol visualization, considering it using a specialized taxonomy, such as the above one, as well as comparing the tool itself to other network protocol visualization tools, is of little use.

Scriptable Visualizations. For our visualization platform, teachers create the visualizations as textual scripts (more of this in the next section) to fit their own teaching materials. Hundhausen and Douglas (2000) provide a differing example, in which, to increase the level of student engagement, students create scripts for low-fidelity visualizations themselves using the authors' *ALVIS system* and *SALSA language*. An obvious third possibility would be the scripts to be created by some third party, such as an Internet-based community or an external supplier of the visualization system.

[4]Examples: Adobe Captivate, Articulate Storyline 360, Lectora, and iSpring Suite.

Another example of scriptability could be the presentation file formats based on *Extensible Markup Language* (XML). Even though the `.pptx` files of PowerPoint are binary packages, they contain XML files that can be generated manually or programmatically to realize visualizations. However, binary packages, such as PowerPoint's native files, are not ideal to be stored into version control systems. Moreover, it can be tedious to learn the details of *PresentationML* (Microsoft Corporation, 2017) as well as to produce it and the binary packages manually. Finally, this specific format obviously enables the creation of PowerPoint presentations only.

Learning Objects. The visualizations realized by using our platform can be thought as *learning objects* (LO). Standards and other specifications exist for many aspects related to LOs. For instance, LOs can be described based on the *Learning Object Metadata* standard (IEEE Computer Society, 2020), and communication between clients and servers could exploit specifications such as *Shareable Content Object Reference Model* (Advanced Distributed Learning Initiative, 2009) and *Learning Tools Interoperability* (IMS Global Learning Consortium, 2021). However, our platform is intended to exist inside learning objects as a part that enables creating them, and such higher-level specifications are not of primary relevance here.

3 THE PLATFORM

Our visualization platform, currently dubbed as CSMV for *Client-Server Messaging Visualizer*, is based on *scripts* that 1) describe the content of the visualization exactly as it will be presented and 2) are used to configure the platform. As mentioned earlier, for our platform, the scripts are tailored by the instructor to fit their own teaching materials.

A second characteristic of CSMV is that the visualizations are divided into static *steps*, between which the student can move one at a time. This allows them to explore the animation on their own pace and backtrack, if needed, to understand some steps better. Continuous animation from the beginning to the end without interaction between the visualization and the student was not considered important in this phase and is not currently supported.

As the idea of CSMV is not to facilitate visualizing "data" in the sense of measurements from simulations or the real world, applying a data-based reference model (e.g., Aaltonen and Lehikoinen, 2005; Brodlie and Mohd Noor, 2007; Chi, 2000; Maletic et al., 2002; Schulz et al., 2016) is not beneficial in general. However, to position CSMV with other visu-

Figure 1: An example of a visualization step.

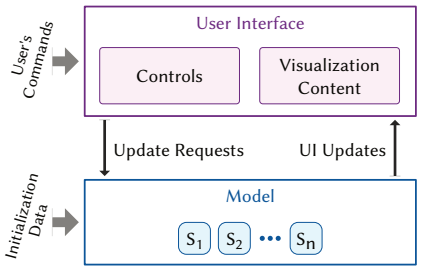

Figure 2: The essential architecture of CSMV.

alization solutions, we can consider it, for instance, in terms of the model of Maletic et al. (2002). From it, we only need the two final phases. The instructor prepares the *Visual Structures* (configured in the script) and the user transforms the *Views* by moving between the steps of the visualization.

We implemented CSMV to work in a modern Internet browser.[5] Figure 1 presents the user interface created by the visualization platform while a step of a visualization is being displayed. One web page can have several of them, each independent with their own visualization. One can add the platform to a web page by (1) including the necessary JavaScript and Cascading Style Sheet files as well as (2) adding an annotated DIV element for each visualization on the page as placeholders for their user interfaces.

The essential high-level architecture of CSMV is presented in Figure 2 below. The *User Interface* (UI) consists of the *Visualization Content* and the related controls, the latter of which are buttons for browsing the visualization. The UI is backed by JavaScript code that includes a *Model*—a state machine that contains, among other things, instructions for visualizing individual steps $(S_1, S_2, \ldots S_n)$ of the visualization and updates the UI based on user's input. When a web page that uses CSMV is loaded into a web browser, CSMV reads the provided configuration, finds the DIV elements that correspond the configured visualizations, and initializes both the UI and the Model for each visualization.

From the instructor's point of view, creating visualizations is relatively straightforward since they are defined as simple JavaScript files that set a value of one variable (see Listing 1 below). This value is a list of setting blocks—one for each visualization on an HTML page. For each visualization, there are essentially three types of information: general settings, *actors* (such as text, code, and images) to be used in the visualization, and *steps* that contain instructions for using those actors.

After initial planning, the instructor defines both the actors and the steps that form the visualization. For actors, there are a few configurable presets, such as the *Browser* and *Web Server* boxes visible in Figure 1 above. In addition, actors containing text and graphics can be defined as chunks of unrestricted HTML code. The visualization sequence is then composed step-by-step by applying available operations to the actor definitions. At the moment, three actor operations are available: hiding and showing them as well as setting their positions in terms of absolute coordinates. In the lack of a visualization editor, all of this is manual work. One can ease it a little by using the provided debugging settings for displaying borders of actors and ignoring visibility.

```
document.CSMesVisSetupData = [
  {
    name: "",
    title: "",
    description: "",

    debug: {},
    // Control size, buttons, etc.:
    environment: {},
    // Actors in the visualization:
    actors: [],
    // Instructions for the actors:
    steps: [],
  },
  { ... }, // next visualization
  :
  :
];
```

Listing 1: A configuration file stub of CSMV.

Due to the multiple learning protocols available in Acos content server[6] (Sirkiä and Haaranen, 2017), embedding the visualizations to the learning material itself is typically straightforward. Whether using LTI or some of the more university-specific protocols, the visualizations can be reused in different courses or repeated within a single course. Finally, because the visualization objects and steps are JavaScript entities,

[5] The development language is EcmaScript 6 that is transpiled into EcmaScript 5 by using Babel. Of third-party run-time libraries, CSMV uses jQuery. The build process is automatized with Gulp.

[6] See: https://github.com/acos-server/acos-server

it is possible to implement customized functionality. For example, clicking a particular object could affect the visualization in some way, such as by skipping certain steps or by launching a questionnaire.

4 THE PILOT STUDY

To evaluate the feasibility of the scripted step-based visualizations and the visualization platform CSMV (Section 3), we piloted them during Spring 2020 on an introductory course in web application development. The course is an open online course targeted mainly for adult learners interested in web development. Experience in programming is not required and, in general, the course is built with life-long learning in mind.[7] 127 students enrolled on this course implementation, including high-school students, professionals with jobs related to web development, and people who intend to change career.

Learning goals of the above course include the following: (1) Basic principles of web pages using Hyper-Text Markup Language and Cascading Style Sheets. (2) Event-driven programming in Internet browsers using JavaScript. (3) Client-server model in web development with Hyper-Text Transfer Protocol (HTTP) as well as Asynchronous JavaScript and XML (AJAX). (4) Simple server-side applications using NodeJS.

Study Arrangements. The course had eight rounds of content and exercises, of which two contained our visualizations. At the beginning of the reading material for the fifth round was a visualization called *Simple Get*, and at the end of the material there was a more elaborated *Detailed Get* visualization. At the beginning of round seven, the *Simple Get* was presented again as a recapitulation, after which the third visualization—*Simple AJAX*—was shown.

Simple Get visualization had four steps and illustrated a request–response pair when a client requests the default web page from a web server using HTTP (Figure 1). *Detailed Get* had essentially the same content but also included an additional step to explain the processing of the request on the web server. *Simple AJAX* had eight steps and continued from *Simple Get* by visualizing an AJAX call triggered by a pressed button on the downloaded web page.

We used Acos content server (Sirkiä and Haaranen, 2017) to host the client-side files and store the usage logs. The logging facility recorded the events generated by CSMV, so that we were able to examine the presses of the movement buttons in the user interface of CSMV. From the logged sessions, we had to filter out the valid ones—those that were inside the official availability times of the rounds,[8] were not empty, and contained events for movement button presses in addition to, for instance, other clicks as well as window focuses and blurs.

Usage of the Visualizations. Our first research question concerned the usage of the visualizations we incorporated to the above target course, and our results are based on analyzing the usage logs of the visualizations. This is useful, for instance, to discover whether visualizations are viewed in full and if students review the material later, as well as to assess the attentiveness of the students, as in an earlier study of Sirkiä and Sorva (2015).

To start with, for *Simple Get* we found 59 valid sessions. The shortest ones were approximately 5 seconds and contained only one button press. For the second and third step, the step-specific average duration was about seven seconds. In 34 sessions (58 %), the visualization was watched from the beginning to the end exactly once, and in 5 sessions (8 %) exactly twice. Two sessions (3 %) contained essentially three passes over the visualization, and the other sessions were some variations of going backwards and forwards. The realized step transition sequences varied between 2 and 16 steps in length. In eleven sessions (19 %), the student went back from the third step after they first arrived in it.

Next, *Detailed Get* was associated with 69 valid sessions starting from 10 seconds. The step-specific average length of the three middle steps was about twelve seconds. In 55 sessions (80 %), the visualization was watched once, and the other sessions contained variations of going back and forth. The length of the step transition sequences varied from 4 to 13 steps.

Finally, 65 valid sessions were recorded for *Simple AJAX*—the longest of the visualizations with 8 steps. Here the session durations increased from approximately 8 seconds upwards. The average step-specific duration of the six middle steps was slightly over three seconds. 32 sessions (49 %) represented watching the visualization exactly once and 2 sessions exactly twice. Interestingly, in 18 sessions (28 %) the user did not proceed beyond the fifth step, which was the first of the additional content compared to *Simple Get*. The rest of the sequences varied in step order.

[7] For the regular students of our university, we provide a more extensive course in web application development.

[8] The rounds had deadlines, after which the content was still browsable but no exercise points were given anymore. Thus, most students viewed the visualization during the time the round was officially active.

The above results demonstrate some simple possibilities for learning analytics that the platform will enable. However, this was a pilot study to test our framework as a proof-of-concept, and the available data logging had still restrictions for the analysis. First of them was that user identifiers were not recorded, which made us unable to relate sessions to each individual user and analyze the behavior of individual users over session boundaries. These limitations are currently being resolved and more complete logging will be carried out during the next spring implementation of the course.

Another limitation was the lack of specifying the beginning and ending states in the visualizations. The first step in our visualizations was automatically visible after the web page was loaded (i.e., the user did not have to indicate the moment they started to use the visualization), so we were unable to measure the actual time used for those steps. Similarly, as the last steps were the final steps of the visualizations, we were not able to identify the moments the users ceased using the visualizations. Consequently, the total time used for the visualizations could not be reliably measured. Furthermore, while we were able to measure the duration of the middle steps, we obviously cannot know what the user has really been thinking and doing during that time (they might not have been really studying at all). In our analysis, we limited the session duration to 700 seconds ($11\frac{2}{3}$ min). For each visualization, several sessions were longer than that.

Student Feedback. In our second research question, we were interested in what the students think about the visualizations, and we answer it based on an end-of-course survey. Out of the 127 enrolled students, 43 (34 %) both gave a research consent and answered the survey question regarding CSMV. We gathered feedback with a seven-point Likert scale (from *strongly disagree* to *strongly agree*, with a neutral option) with the statement *"I found the client-server visualizations useful for my learning."* 35 students agreed with the presented statement on some level: 25 *agreed or strongly agreed* and 10 students *somewhat agreed*. Seven students chose the neutral option, and only one disagreed with the statement with *somewhat disagree* option.

We also solicited open feedback regarding CSMV with *"Is there something else you'd like to say about the client-server visualizations?"* To this question, we received 12 answers. One of these we categorized as negative (*"I struggled with these a bit"*), seven as neutral (e.g., *"Concept is good but I didn't benefit from it that much"*), and four as positive (e.g., *"These were helpful for more abstract topics"*).

5 DISCUSSION

In this paper, we have described a visualization platform, CSMV, that was developed according to the research goal that we presented in the Introduction earlier. As a general-purpose platform, it allows the teacher to visualize a wide variety of subjects (e.g., network protocols). The teachers create visualizations as textual scripts, after which they can easily exploit them in learning management systems that allow the usage of web pages that integrate custom JavaScript and Cascading Style Sheets. For learning analytics, CSMV supports logging of students' activities, and is also open-source[9]—free to be used and customized.

To evaluate CSMV, we also described a pilot study in the context of visualizing messaging between clients and servers. Our results from the study show that from a technical perspective, CSMV functions as expected. A minor shortcoming concerned logging the durations of the first and the last steps—this is fixed by simply adding a "starting" step and a "completed" step.

Pilot Study. Our research questions were concerned with both how the visualizations in our pilot were used and what the students thought about them. Regarding the first question, the analysis of the log data confirms that the course participants actually used the visualizations. Moreover, the average step viewing time strongly suggests that they considered the steps separately, reading the available content, instead of just clicking through all steps quickly. Each intermediate step included a short text to read, and some steps included a brief JavaScript code snippet.

Based on this pilot study, we found confirmation that the visualizations are explored by students. Furthermore, our end-of-course survey related to the second question provided mostly positive feedback, even if the visualization content was only used in two rounds during the course.

What is particularly encouraging is that some of the students not only viewed the steps once in a sequence but also went back and forth in the visualizations. This seems to indicate that especially these students were actively engaged with the material instead of just "skimming it through." As *Detailed Get* visualization was only a slightly modified derivation of *Simple Get*, it was to be expected that a larger number of sessions (80 % compared to 58 %) contained only a single pass-through of the visualization.

On the next implementation of the course during Spring 2021, we will carry out a more comprehensive study about the potential learning gains with our vi-

[9] See: https://github.com/aleksi-lukkarinen/Client-Server-Messaging-Visualizer

sualizations and the platform. Then, we are able to combine the visualization log data with other course performance data (e.g., weekly exercise points) and investigate the actual learning gains.

Rich Learning Content. The type of visualization that interacts with the student is what Brusilovsky et al. (2014) call *smart learning content* (SLC). Their three-dimensional classification of SLC systems, as well as the position of CSMV in it, is illustrated in Figure 3. As CSMV accepts only pre-specified input in the form of button presses to switch from step to step, it is in the unintelligent end of the *Input* dimension. Similarly, CSMV produces generic output, that is, the same input produces the same output for all students; this positions it to the unintelligent end of the *Output* dimension. The characteristic that makes CSMV "smart" in terms of this classification is that the process to produce the output is—in its all simplicity—fully computational: Our scripts are self-sufficient, so while students study the visualizations, the underlying scripts can be executed by the visualization platform without any third-party intervention. Thus, CSMV is located in the "intelligent" end of the *Process* dimension.

The idea of scripts and steps might affect student engagement with CSMV. For instance, Sorva et al. (2013) suggested a two-dimensional engagement taxonomy (2DET) that constructs its first dimension on the basis of both an earlier taxonomy by Naps et al. (2002) and its extension by Myller et al. (2009). In its first dimension, *Direct Engagement*, CSMV currently enables achieving the third level: *Controlled Viewing*. In it, students can change the view—in this case, move between the steps—but more advanced activity (*Responding*, *Applying*, *Presenting*, *Creating*) is not supported yet. However, a simple extension would be to include, for instance, multiple choice questions between some steps, which would correspond to *Responding* level. The second dimension of 2DET is about *Content Ownership* with the expectation that a greater feeling of ownership increases caring and thus the engagement. As CSMV, by design, delivers only *Given Content* (the first level), it cannot reach the three higher levels (*Own Cases*, *Modified Content*, and *Own Content*) in this dimension.

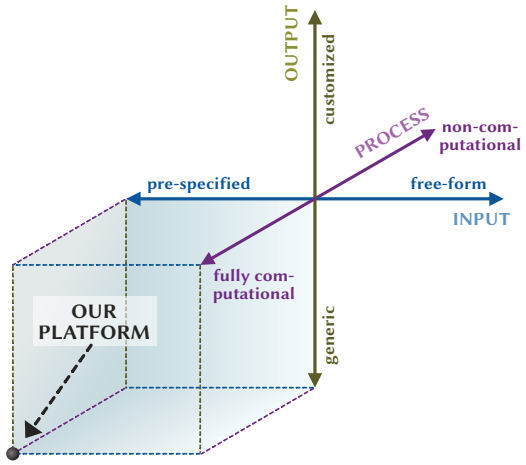

Figure 3: The position of CSMV in the smart learning content classification system of Brusilovsky et al. (2014).

Future Work. We intend to continue development and research with CSMV. We are developing a more comprehensive and overarching logging to overcome the limitations with the logging in our pilot study. After that, there are many directions to take CSMV. They include (1) integrating various exercise types to visualizations; (2) actions for elements, including variables and conditions; (3) non-linear scripts; (4) step and actor templates; (5) reusing steps; (6) defining actors using Scalable Vector Graphics; and (7) adding better animation capabilities.

In the future, we can study student engagement more closely, which could be supported by increasing the level of interactivity between the students and the visualization. Of the above directions of development, the exercises, actions, and non-linear scripts have the potential to affect positively in this respect. Also, using more elegant visuals and animating them might have the same effect. Furthermore, it would be of interest to understand more thoroughly how students use the visualizations to create profiles of typical usage. This information could be related with students' learning gains, which could be measured in terms of performance in added multiple choice questions and coding exercises incorporated into the corresponding rounds. The results could aid in further development of CSMV as well as in predicting students' success and identifying their points of struggle.

REFERENCES

Aaltonen, A. and Lehikoinen, J. (2005). Refining visualization reference model for context information. *Pers Ubiquit Comput*, 9(6):381–394.

Adobe Inc. (2019). End of Life (EOL) for Adobe Shockwave. https://helpx.adobe.com/shockwave/shockwave-end-of-life-faq.html; Accessed: Feb. 16, 2021.

Adobe Inc. (2021). Adobe Flash Player End of Life. https://www.adobe.com/products/flashplayer/end-of-life.html; Accessed: Feb. 16, 2021.

Advanced Distributed Learning Initiative (2009). SCORM 2004 4th Edition. https://adlnet.gov/projects/scorm-2004-4th-edition/; Accessed: Feb. 16, 2021.

Averbukh, V., Averbukh, N., Vasev, P., Gvozdarev, I., et al. (2019). Metaphors for Software Visualization Systems Based on Virtual Reality, AVR 2019. In De Paolis, L. T. and Bourdot, P., editors, *Augmented Reality, Virtual Reality, and Computer Graphics*, LNCS, vol. 11613, pages 60–70, Cham, Zug, CH. Springer International Publishing.

Awodele, O., Oluwabukola, O., Ogbonna, C., and Ajayi, A. (2015). Packet Sniffer—A Comparative Characteristic Evaluation Study. In *Proc InSITE*, pages 91–100, Santa Rosa, CA, USA. Informing Science Institute.

Brodlie, K. W. and Mohd Noor, N. F. (2007). Visualization Notations, Models and Taxonomies. In Lim, I. S. and Duce, D., editors, *Theory and Practice of Computer Graphics 2007, Eurographics UK Chapter Proc*, pages 207–212, Goslar, DE. Eurographics Assoc.

Brusilovsky, P., Edwards, S., Kumar, A., Malmi, L., et al. (2014). Increasing Adoption of Smart Learning Content for Computer Science Education. In *Proc ITiCSE-WGR*, pages 31–57, New York, NY, USA. ACM.

Chi, E. H.-H. (2000). A Taxonomy of Visualization Techniques Using the Data State Reference Model. In *Proc INFOVIS*, pages 69–75, Piscataway, NJ, USA. IEEE.

Crescenzi, P., Gambosi, G., and Innocenti, G. (2005). NetPrIDE: An Integrated Environment for Developing and Visualizing Computer Network Prot. In *Proc 10th ITiCSE*, page 306–310, New York, NY, USA. ACM.

Crescenzi, P. and Innocenti, G. (2002). Towards a Taxonomy of Network Protocol Visualization Tools. In Diehl, S., editor, *Software Visualization*, LNCS, vol. 2269, pages 241–255, Berlin, Heidelberg, DE. Springer Berlin Heidelberg.

Fuentes, F. and Kar, D. C. (2005). Ethereal vs. Tcpdump: A Comparative Study on Packet Sniffing Tools for Educational Purpose. *J Comput Sci Coll*, 20(4):169–176.

Hall, J., Moore, A., Pratt, I., and Leslie, I. (2003). Multi-Protocol Visualization: A Tool Demonstr. In *Proc MoMeTools*, pages 13–22, New York, NY, USA. ACM.

Hundhausen, C. and Douglas, S. A. (2000). SALSA and ALVIS: A Language and System for Constructing and Presenting Low Fidelity Algorithm Visualiz. In *Proc IEEE VL*, pages 67–68, Piscataway, NJ, USA. IEEE.

IEEE Computer Society (2020). IEEE Standard for Learning Object Metadata (1484.12.1-2020). Standard, IEEE, New York, NY, USA.

IMS Global Learning Consortium (2021). Learning Tools Interoperability. https://www.imsglobal.org/activity/learning-tools-interoperability; Accessed: Feb. 16, 2021.

Maletic, J. I., Marcus, A., and Collard, M. L. (2002). A Task Oriented View of Software Visualization. In *Proc 1st VISSOFT*, pages 32–40, Piscataway, NJ, USA. IEEE.

Microsoft Corporation (2017). Welcome to the Open XML SDK 2.5 for Office | Microsoft Docs. https://docs.microsoft.com/en-us/office/open-xml/open-xml-sdk; Accessed: Feb. 16, 2021.

Microsoft Corporation (2021a). Lifecycle FAQ - Internet Explorer and Microsoft Edge - Microsoft Lifecycle | Microsoft Docs. https://docs.microsoft.com/en-us/lifecycle/faq/internet-explorer-microsoft-edge; Accessed: Feb. 16, 2021.

Microsoft Corporation (2021b). Silverlight End of Support. https://support.microsoft.com/en-us/windows/silverlight-end-of-support-0a3be3c7-bead-e203-2dfd-74f0a64f1788; Accessed: Feb. 16, 2021.

Milne, I. and Rowe, G. (2004). OGRE: Three-Dimensional Program Visualization for Novice Programmers. *Educ Inform Tech*, 9(3):219–237.

Myller, N., Bednarik, R., Sutinen, E., and Ben-Ari, M. (2009). Extending the Engagement Taxonomy: Software Visualization and Collaborative Learning. *ACM Trans Comput Educ*, 9(1):Article 7.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., et al. (2002). Exploring the Role of Visualization and Engagement in Computer Science Educ. In *Proc ITiCSE-WGR*, pages 131–152, New York, NY, USA. ACM.

Oracle Corporation (2020). Java SE Support Roadmap. https://www.oracle.com/java/technologies/java-se-support-roadmap.html; Accessed: Feb. 16, 2021.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA. https://mindstorms.media.mit.edu/; Accessed: Feb. 16, 2021.

Schulz, H.-J., Angelini, M., Santucci, G., and Schumann, H. (2016). An Enhanced Visualization Process Model for Incremental Visualization. *IEEE Trans Vis Comput Graph*, 22(7):1830–1842.

Schweitzer, D., Baird, L., Collins, M., Brown, W., et al. (2006). GRASP: A Visualization Tool for Teaching Security Protocols. In *Proc 10th CISSE*, pages 75–81, MD, USA. University of Maryland.

Schweitzer, D. and Brown, W. (2007). Interactive Visualization for the Active Learning Classr. In *Proc 38th SIGCSE*, pages 208–212, New York, NY, USA. ACM.

Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., et al. (2010). Algorithm visualization: The state of the field. *ACM Trans Comput Educ*, 10(3):Article 9.

Sirkiä, T. and Haaranen, L. (2017). Improving online learning activity interoperability with Acos server. *Softw Pract Exper*, 47(11):1657–1676.

Sirkiä, T. and Sorva, J. (2015). How Do Students Use Program Visualizations within an Interactive Ebook? In *Proc 11th ICER*, pages 179–188, New York, NY, USA. ACM.

Sorva, J., Karavirta, V., and Malmi, L. (2013). A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Trans Comput Educ*, 13(4):Article 15.

SumTotal Systems, LLC (2021). Toolbook Knowledge Base: Does Toolbook support Edge? http://tb.sumtotalsystems.com/KBFiles/kb/EdgeSupport.html; Accessed: Feb. 16, 2021.