New Acos Content Types

Ari Korhonen
Aalto University
Espoo, Finland
ari.korhonen@aalto.fi

Giacomo Mariani
Aalto University
Espoo, Finland
giacomo.mariani@aalto.fi

Peter Sormunen
Aalto University
Espoo, Finland
peter.sormunen@aalto.fi

Jan-Mikael Rybicki
Aalto University
Espoo, Finland
jan-mikael.rybicki@aalto.fi

Aleksi Lukkarinen
Aalto University
Espoo, Finland
aleksi.lukkarinen@aalto.fi

Lassi Haaranen
Aalto University
Espoo, Finland
lassi.haaranen@aalto.fi

Artturi Tilanterä
Aalto University
Espoo, Finland
artturi.tilantera@aalto.fi

Juha Sorva
Aalto University
Espoo, Finland
juha.sorva@aalto.fi

ABSTRACT

This paper demonstrates three new content packages recently published for Acos, the server for sharing smart learning content. The packages are aimed at 1) code annotation, 2) automatically assessed tracing exercises, and 3) scripted stepwise animations for stepping through explanations of various content. The content can be disseminated to different learning management systems by utilising several learning protocols, such as LTI.

Author Keywords

online learning, visualizations, automatic feedback, assessment, code annotations, visual algorithm simulation, algorithm animation, tracing exercises, stepwise animations

CCS Concepts

•**Social and professional topics** → **Computing education;**
•**Human-centered computing** → *Visualization toolkits;*

INTRODUCTION

Acos is a server for integrating Online Learning Activities (OLA) and interactive content to various learning management systems (LMS), first published in 2017 [12] and available in Github[1]. It provides interfaces for LMSs in various learning protocols, such as LTI [3] and local learning protocols, allowing content developers to integrate their OLAs to Acos and then distribute the contents in different LMSs. An overview of how exercises are embedded into an LMS is shown in Figure 1.

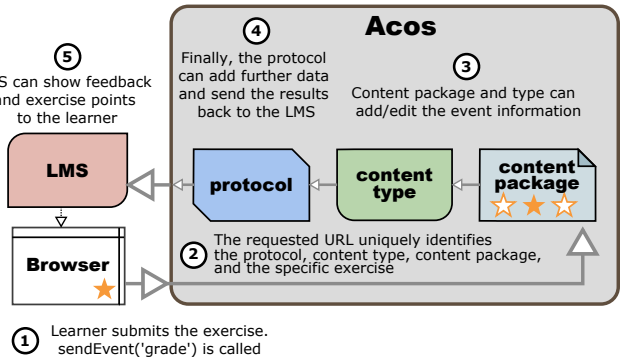

Figure 1. Flow of data in Acos exercise submission.

Originally, the content within Acos was focused on computing education, for example, providing interactive Parsons problems[2] and JSVEE program visualizations [11]. Since then, the scope of content developed for Acos has broadened to teaching academic English, which also benefits from similar pedagogical solutions as does computing education.

We have recently developed and published three content packages for Acos that could be of interest to the broader community. In this paper, we present 1) Acos code annotation, and complementary tool to create them, 2) Integration of Visual Algorithm Simulation Exercises to Acos ecosystem, and 3) scripted stepwise animations for stepping through explanations of various content.

CODE ANNOTATIONS IN ACOS

Acos code annotation is a content package designed for adding explanations to plain text and program code excerpts (other

[1] https://github.com/acos-server/acos-server

[2] https://github.com/acos-server/acos-jsparsons-python

Seventh SPLICE Workshop at SIGCSE 2021 "CS Education Infrastructure for All III: From Ideas to Practice", SPLICE'21, March 15-16, 2021, Virtual Event

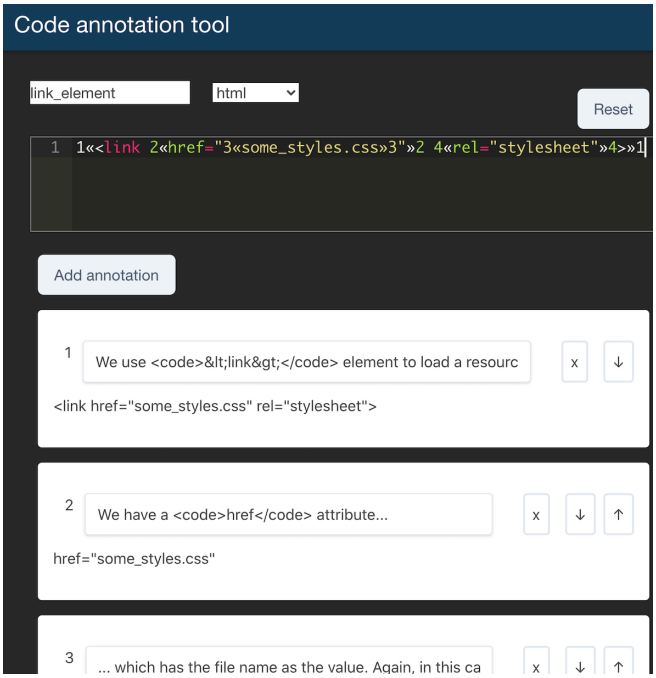

Figure 3. Creating the annotations in the provided tool.

similar tools exist as well, e.g. [2]). Traditionally, code samples in textbooks are explained separately in the main text, which can increase the cognitive load of the learner due to split attention [1]. Of course, the code excerpts themselves can include comments, which are indicated using various symbols, such as #, // or %. These traditional forms of explaining code features are inherently static, lacking dynamic interaction with the learning materials.

With Acos code annotations, the instructor can create annotated examples that highlight particular segments in the text when learners hover their mouse over the explanations. Figure 2 illustrates an annotated code example. This approach allows learner to locate the essential features in the code more easily than in the traditional textbook format, while keeping track with the location of explanations. This should reduce the risk of split attention and maintain learner control [10], making the code or text analysis a more dynamic and interactive experience for the learner.

Pedagogically, the Acos code annotations package is suitable for introducing concepts in similar fashion than a textbook, web page or video, but it promotes more possibilities for engagement and interaction for the learner. The interface can be embedded in an existing LMS and does not require installing any additional software or plugins, provided that the Acos server is already available.

Figure 2 illustrates a case in which a learner hovers the mouse over the second annotation (green text box) from the top. Here the annotation highlights one segment of HTML code in red. In addition, it is possible to highlight many separate segments within one annotation, such as the `href` and `rel`, to illustrate how separate features are related within an example text.

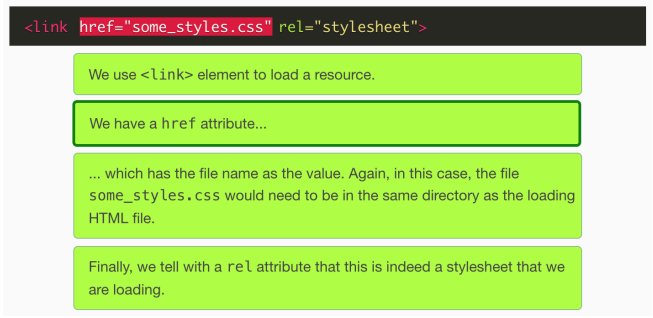

Figure 2. An example of code annotation with one annotation highlighted.

The code annotations are stored in a JSON file and then served by the Acos server. Since editing JSON files manually can be an error prone and tedious process, a teacher tool was developed for creating code annotations activities (see Figure 3). The teacher tool generates the exercise contents both in JSON and RST formats. The tool renders the annotations in real time, requiring no separate compiler. Each annotation can be linked to one or more segments in the text. The annotations (i.e., explanations) can include HTML tags, such as those for adding typographical emphasis or URLs to other resources online.

Since Acos code annotation can be used for annotating plain text documents, this tools has also been used in academic writing courses to annotate texts in natural languages. In these scenarios, code highlighting can be turned off. Currently, Acos code highlighting is available for Python, HTML, CSS, JavaScript, and C.

Acos code annotation also stores log data on the Acos server in the JSON format, such as when the users hover their mouse on the annotations or move the mouse off the annotations. This log data could be used for learning analytics or educational data mining (EDM) to further understand learner behavior or support learners in their learning process.

We plan on publishing `npm` packages later on. For the time being, the content is available at GitHub:

- **Content type:**
  `https://github.com/sormpe/acos-code-annotation`
- **Sample content for annotations:**
  `https://github.com/sormpe/acos-code-annotation-sample`
- **Tool for instructors to create annotations:**
  `https://github.com/sormpe/acos-annotation-tool`

The sample annotations and the annotation tool are demonstrated at `https://acos.cs.aalto.fi/`:

- **Sample content, e.g.:**
  `https://acos.cs.aalto.fi/html/codeannotation/code-annotation-sample/hello_world_python`
- **Annotation tool:**
  `https://acos.cs.aalto.fi/code-annotation-tool/`

VISUAL ALGORITHM SIMULATION EXERCISES

Visual Algorithm Simulation [6] (VAS) exercises build on the concept of Algorithm Animation (AA). An AA visualises an algorithm in action by displaying the data structures involved, their states and the operations performed. The visualisation is typically a step-by-step sequence, which also offers the possibility to step back and forth along the animation series [6, 7]. Compared to AAs, VAS exercises further engage the learner by requiring an active role in setting the sequence steps. According to the Engagement Taxonomy [9], the learner operates in a higher level of engagement than purely watching an animation. In a typical VAS exercise, the learner manipulates the state of the data structures through a graphical user interface and simulates step-by-step the actions of an algorithm. This can happen in different levels of abstraction. By means of this activity, learners trace the algorithm, and based on the immediate automatic feedback, learners can verify that they correctly understood the behavior of the algorithm, or they can retry if they failed to do so. Typically, all students retry an exercise until they reach a correct solution. In each trial, the input for the algorithm is different; thus, the system can also reveal the model solution between trials. Due to the varying input, the correct trace is also different in each trial. Figure 4 displays a VAS exercise where the learner is asked to simulate the Build-Heap algorithm by interacting with either the implementation-level binary heap array or logical-level binary tree: clicking two elements with the mouse swaps them.

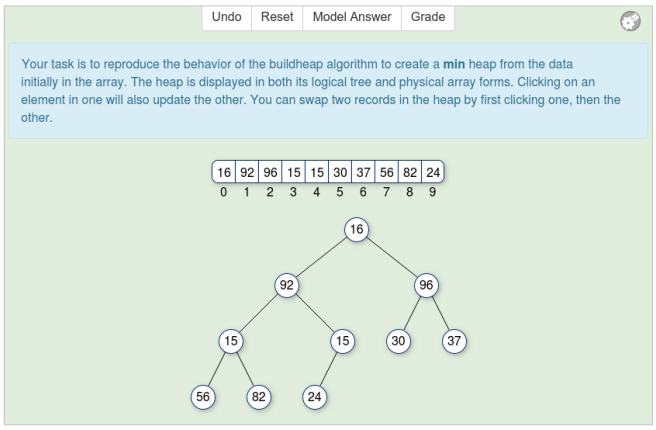

Figure 4. A VAS exercise on a linear-time Build-Heap algorithm from the OpenDSA e-textbook. The learner is expected to swap keys either in the array or in the tree to constitute a min heap.

The content type package *acos-jsav*[3] adds support for content created with the JavaScript Algorithm Visualization library [4] (JSAV), while the content package *acos-jsav-vas*[4] adds support for serving JSAV VAS exercises from an Acos server. Similar to other content hosted on an Acos server, instructors can visualize and test VAS exercises through the server homepage[5], and copy the URL to import the exercise into an LMS [12]. Running an exercise within an LMS may require allowing third-party cookies on the client browser. Developers wishing to add existing JSAV VAS exercises into the Acos server need to install the above type and content packages and follow the instructions given in the respective documentation. Guidance on how to create VAS exercises with the JSAV library is available at http://www.jsav.io. Potentially, the OpenDSA e-textbook[6] [5] offers an extensive collection of JSAV-based VAS exercises which could be served from Acos, thus enabling their use in various LMSs.

[3] https://www.npmjs.com/package/acos-jsav

[4] https://www.npmjs.com/package/acos-jsav-vas

[5] http://acos.cs.aalto.fi/html/jsav/jsav-vas/insertion_sort

We plan to continue developing VAS exercises and tools by

- broadening the exercise offering,
- developing further an existing prototype [8] for recording the trace produced by a learner and saving it to the LMS, and
- developing further the existing prototype player [8] to reproduce an algorithm animation from a saved trace.

SCRIPTED STEPWISE ANIMATIONS

Animated diagrams are useful for illustrating a variety of topics. To support their creation in Acos, we have created a general-purpose visualization framework that enables teachers to script animations as a sequence of *steps*. Each step corresponds to a distinct picture that is shown to learners.

The framework is generic in the sense that it is not customized to any particular application area, such as algorithms or a specific kind of an algorithm. Instead, the teacher is free to compose pictures from any images, text, and other static resources; thus, the visualizations may be useful for a broad range of purposes. However, as we initially created the framework for visualizing communication between clients and servers, it is, at least for now, dubbed *CSMV*, which is short for *Client-Server Messaging Visualizer*.

Student Interface. Figure 5 shows an example of a student interface of the CSMV: The animation area displays a step of an animation about asynchronous requests. Students can use the control buttons below to move linearly back and forth through the steps. Another example of an animation step is shown in Figure 6.

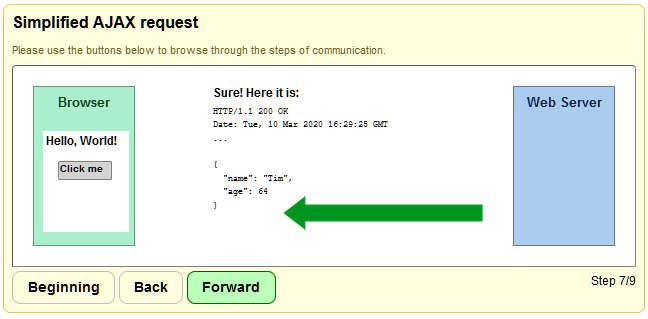

Figure 5. A step within a CSMV animation that illustrates making asynchronous requests in JavaScript. The green arrow has been loaded as an image; the other visual elements were constructed in the configuration file for the animation.

Teacher Interface. To create an animation, the teacher plans the content, creates or acquires the appropriate static resources

[6] https://opendsa-server.cs.vt.edu/

```
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 stub of a CSMV configuration file.

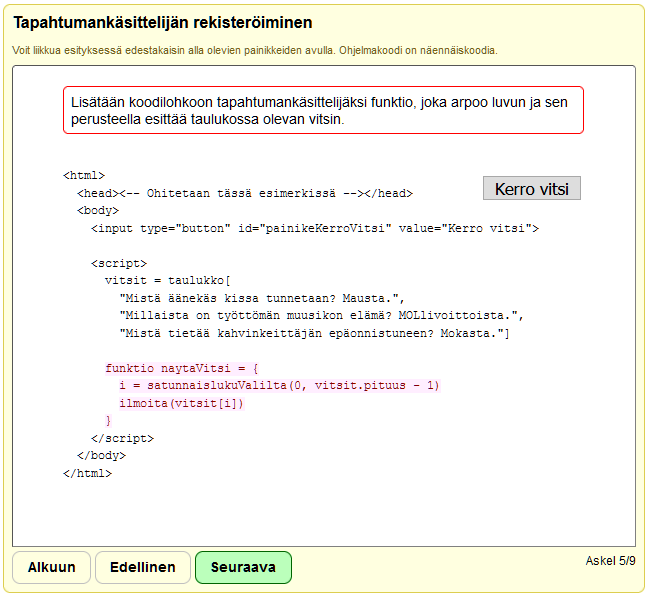

Figure 6. A step within an animation explaining (in Finnish) how to register an event handler. The animation gradually adds blocks of code (with a highlighted background) and describes each addition in the red rectangle at the top.

(such as images), and defines the visualization in a JavaScript file. Listing 1 displays a stub of such file. In addition to basic metadata and debugging settings, each animation needs three types of configurations: (1) *environment* settings, such as the animation's size and the visible buttons; (2) a list of *actors*, which are the visual elements that can be used in step definitions; and (3) a list of *step definitions*, which describe how each step is realized using the actors.

Actors can be defined as blocks of plain text, chunks of HyperText Markup Language (HTML), or with the aid of some presets, such as the boxes for the Browser and the Server in Figure 5. Cascading Style Sheets (CSS) can be used to define the appearance of the actors. At the moment, CSMV offers three operations for controlling the actors: (1) setting actor's position in absolute coordinates, (2) showing an actor, and (3) hiding an actor. Smooth animation is not supported yet but can be achieved in a limited form by exploiting animated image file formats.

At this prototype stage, no dedicated editor exists for creating CSMV animations. The creation process is manual, but an available configuration file template is relatively easy to use for anyone with basic HTML and CSS skills.

Prototype Implementation. CSMV is implemented using EcmaScript 6 and jQuery. At the moment, the data model is a state machine based on the list of steps read from the configuration file. Usage logging is based on events generated by CSMV and recorded by Acos. The source project lacks a proper documentation at this time, but it is publicly available[7] for use and further development.

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

In terms of the two-dimensional engagement taxonomy (2DET) of Sorva et al. [13], CSMV is currently on the *Controlled Viewing* level in *Direct Engagement* and on the *Given Content* level in *Content Ownership*. As CSMV offers predefined content only, its location in the latter dimension is fixed by design. However, its highest possible level of *Direct Engagement* can be raised, for instance, by implementing support for integrating exercises to the visualizations for both adding interactivity and analyzing learning.

Another possible direction for future development is to generalize the data model to directed graphs. This would make it possible to realize non-linear scripts, in which the user would be able to choose—or would be directed to—various paths during the visualization. Other potential features to be added include actions for the actors, with variables and conditions; actor and step templates; actor definitions using Scalable Vector Graphics; and smooth animation.

REFERENCES

[1] Tamara van Gog. 2014. The Signaling (or Cueing) Principle in Multimedia Learning. In *The Cambridge Handbook of Multimedia Learning* (2 ed.), Richard E. Mayer (Ed.). Cambridge University Press, Cambridge, UK, Chapter 11, 263–278. DOI: http://dx.doi.org/10.1017/CBO9781139547369.014

[2] Roya Hosseini, Kamil Akhuseyinoglu, Andrew Petersen, Christian D Schunn, and Peter Brusilovsky. 2018. PCEX: interactive program construction examples for learning programming. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*. 1–9.

[3] IMS Global Learning Consortium. 2021. Learning Tools Interoperability. (2021). Retrieved Jan. 29, 2021 from http://www.imsglobal.org/toolsinteroperability2.cfm

[4] Ville Karavirta and Clifford A. Shaffer. 2013. JSAV: The JavaScript Algorithm Visualization Library. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. Association for Computing Machinery, New York,

NY, USA, 159–164. DOI:http://dx.doi.org/https://doi.org/10.1145/2462476.2462487

[5] Ville Karavirta and Clifford A. Shaffer. 2016. Creating Engaging Online Learning Material with the JSAV JavaScript Algorithm Visualization Library. *IEEE Transactions on Learning Technologies* 9, 2 (4 2016), 171–183. DOI: http://dx.doi.org/10.1109/TLT.2015.2490673

[6] Ari Korhonen. 2003. *Visual algorithm simulation*. Ph.D. Dissertation. Helsinki University of Technology, Espoo, Finland. http://urn.fi/urn:nbn:fi:tkk-001030

[7] Ari Korhonen and Lauri Malmi. 2000. Algorithm Simulation with Automatic Assessment. *SIGCSE Bull.* 32, 3 (July 2000), 160–163. DOI: http://dx.doi.org/10.1145/353519.343157

[8] Giacomo Mariani. 2020. Design of an Application to Collect Data and Create Animations from Visual Algorithm Simulation Exercises. Master's Thesis, Aalto University, Espoo, Finland. (2020). http://urn.fi/URN:NBN:fi:aalto-202005313418

[9] Thomas L. Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J. Ángel Velázquez-Iturbide. 2002. Exploring the Role of Visualization and Engagement in Computer Science Education. *SIGCSE Bull.* 35, 2 (June 2002), 131–152. DOI: http://dx.doi.org/10.1145/782941.782998

[10] Katharina Scheiter. 2014. The Learner Control Principle in Multimedia Learning. In *The Cambridge Handbook of Multimedia Learning* (2 ed.), Richard E. Mayer (Ed.). Cambridge University Press, Cambridge, UK, Chapter 21, 487–512. DOI: http://dx.doi.org/10.1017/CBO9781139547369.025

[11] Teemu Sirkiä. 2018. Jsvee & Kelmu: Creating and tailoring program animations for computing education. *Journal of Software: Evolution and Process* 30, 2 (2018), e1924. DOI:http://dx.doi.org/10.1002/smr.1924

[12] Teemu Sirkiä and Lassi Haaranen. 2017. Improving online learning activity interoperability with Acos server. *Software: Practice and Experience* 47, 11 (2017), 1657–1676. DOI:http://dx.doi.org/10.1002/spe.2492

[13] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Transactions on Computing Education* 13, 4, Article 15 (Nov. 2013), 64 pages. DOI: http://dx.doi.org/10.1145/2490822