The Synchronisation of E-Learning Courses Using Agent Technology

Aleksandra Aleksić, Milan Vidaković, Goran Savić, Aleksandar Kaplar, Milan Segedinac
University of Novi Sad, Faculty of Technical Sciences, Serbia
{aleksandra.a, minja, savicg, aleksandar.kaplar, milansegedinac}@uns.ac.rs

***Abstract*— The focus of this paper is on implementing a system for content synchronisation of e-learning courses in a distributed virtual educational environment. The system is implemented with the use of agent technology. The basic idea is bounding a software agent to each e-course. The agent manages its course content by exchanging data with other agents. An agent can integrate into its own course other course content that it obtained from other agents. An agent middleware named Siebog is used to implement synchronisation of data. Currently, the system works with Canvas LMS.**

I. INTRODUCTION

A learning management system (LMS) is a combination of software services that are used for the purpose of supporting and organizing online learning. [1] Due to the increasing number of enrolled students in educational programs, there is a rising problem with distributing material to the participants of various courses. LMS is an efficient way of providing online course materials in order to support student learning and faculty teaching. One example of an open source LMS system is LMS Canvas [2].

It is a quite common case that a lecturer teaches one course in two different educational institutions and both facilities use a LMS application in their teaching programs. The lecturer would have to update the material on both LMS platforms manually. This could lead to different problems such as increased time consumption, bad synchronisation, and potential loss of materials.

Technically, it is hard to provide different institutions or different sections of the same institution with the same technical platform. Lectures also have content that differs from institution to institution. On the other hand, there is a similarity in the content of different courses. For this reason, we need a technical solution for synchronisation of different courses in cases where it is needed.

A. Existing Solutions

One way to synchronise course content is to export it, and then to import it into the other system. A problem which can arise, with the import of an exported course, is that sometimes the format of the data does not match. If the data needs to be platform independent, then the data can be exported in another format that can cover selected aspects of a course. For example, SCORM [3] and IMS CP [4] contain only materials from the course, IMS QTI [5] contains tests, etc. In the Canvas LMS application, which is the focus of this research paper, automatic course synchronisation is possible on the same Canvas instance. The Canvas LMS system offers synchronisation options between two or more courses. The courses can be linked, or they can have the same blueprint in order to synchronise their content. Synchronisation is not possible in a distributed environment.

A course can be divided into sections. Cross-listing allows users to move section enrollments from individual courses and combine them into one course. This helps instructors that teach the same sections in different courses to manage course data in one location. To prevent data loss, cross-listing should be done while courses are unpublished. If cross-listing is done during the time period that the courses are active (published), it can cause certain associated data like grades or submissions to be deleted.

The advantage of course content synchronisation with reference to a defined blueprint is that courses are not dependent on each other. The possibility of overwriting data is reduced. This can also be a drawback. If a change is made in an associated course and if there is a need for it to be reflected on another course that is connected to the same blueprint, this change will not be made. This change will also affect the content that has been modified and the blueprint will no longer apply. Unlike cross-listing, for the change to apply the modification must be done in the blueprint.

Both solutions are also missing the possibility of synchronising data between multiple Canvas applications. By default, the content can be synchronised within one Canvas application only. This paper deals with the synchronisation between systems that are installed on different computers. Technically it is not feasible to integrate multiple independent Canvas applications into a single instance. The Canvas LMS cannot be implemented with the knowledge of other systems. A possible solution would be a separate system that is a mediator and handles the synchronisation of data between various LMS applications. With the aid of the middleware Siebog [6] we were able to incorporate a standalone system to handle the replication process.

II. SIEBOG

Siebog is a multi-agent middleware system which we use in this research as a middleware between LMS Canvas systems. Siebog represents an architecture for software agents displayed in Figure 1. Siebog consists of three modules on the server side: the Agent Manager, it controls the life-cycles of agents and can act as an agent directory, the Message Manager, which is in charge of communication between agents, and the WebClient Manager, whose role is to be an intermediary for server-to-client messaging, and to handle state persistence for client-side agents. [7]

Copyright 2019 by Information Society of Serbia - ISOS, Serbia | Creative Commons License: CC BY-NC-ND

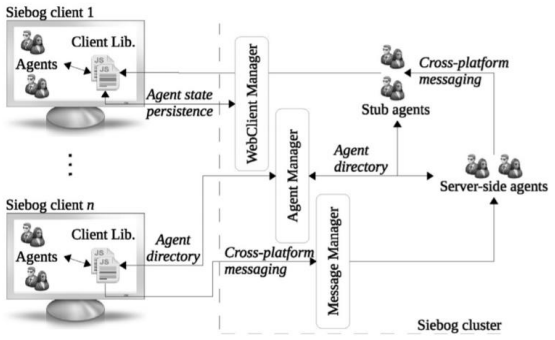

Figure 1. The overall architecture of Siebog multiagent middleware [7]

Siebog provides beneficial functionalities both on the server and client side, as well as between the server and the client. On the server it offers automatic agent load-balancing and fault-tolerance. On the client it offers platform-independence and runtime efficiency. Between the server and the client, it provides cross-platform messaging, heterogeneous mobility, and code mobility. [6]

For the purpose of this research Siebog is required to handle the exchange of the data between courses. Each course is tied to an agent and agents are observed as the key elements in the data exchange process. The proposed solution is a system that takes the idea of a blueprint course and selected course replication and applies remote synchronisation of data between two canvas applications. Both applications use the same Siebog Middleware system for the exchange of actions and data. Some key points are:

- data loss is reduced to a minimum, the user has control of which section/file/folder will be synchronised,
- a course does not need to exist for the replication to be applied, and all of this is done between selected remote systems.

With the Siebog middleware system instead of relying on direct communications among LMS Canvas systems and manually synchronising data, the intricacy and errors are decreased. The solution is based on agents deployed in the Siebog middleware. One agent is created per course. These agents communicate by exchanging messages. An agent can send requests and responses to other agents, in order to synchronise the content of the courses.

III. IMPLEMENTATION

The LMS Canvas API Library for Java [8] is used to wrap the Canvas LMS REST API [9]. For the purpose of this project, LMS Canvas API Library for Java was modified and new functionalities were implemented such as creating, deleting files and folders. Upload and download of files were also implemented.

A. Canvas API Library extension

The existing Canvas API Library was expanded with two new models the File and the Folder class, that respectively represent a file and a folder in a canvas system. The required fields for the resources file and folder are defined in the Canvas LMS API Documentation [9]. The file object is described with the following properties:

- id – the files identifier,
- uuid – universally unique identifier
- folder_id – the folders unique identifier in which the file is located
- display_name – the displayed named of the file in the Canvas application
- filename – the name of the uploaded/saved file
- content-type – the type of file
- url – the URL where the file was uploaded on the Canvas server
- size – size od the file
- created_at – the date of the file creation
- update_at – the date when the file was updated
- modification_at – the date of the file modification

The folder object is described with the following properties:

- id – the folders identifier
- context_type – the type of the context inside the folder
- context_id – the identifier of the context inside the folder
- files_count – the number of files in the folder
- folders_count – the number of folders in the folder
- create_at – the date of the folder creation
- updated_at – the date when the folder was updated
- folders_url – the URL location of the saved folders
- files_url – the URL location of the saved files
- name – the name of the folder
- full_name – the folders absolute path in the canvas application
- parent_folder_id – the identifier of the parent folder

Other classes that were added to the library are FileImpl and FolderImpl. These classes have the initial role of mapping the required parameters for the Canvas LMS File and Folder REST API. The following REST API calls are implemented in the File API:

- List files – calls for listing files in
  - a specific course (GET /api/v1/course/:course_id/files),
  - a specific folder (GET /api/v1/folders/:id/files)
- Get file – calls for retrieving files by:
  - file id (GET /api/v1/files/:id)
  - course id and file id (GET /api/v1/course/:course_id/files/:id)
- Update file:
  - update some settings on the specified file (PUT /api/v1/files/:id)

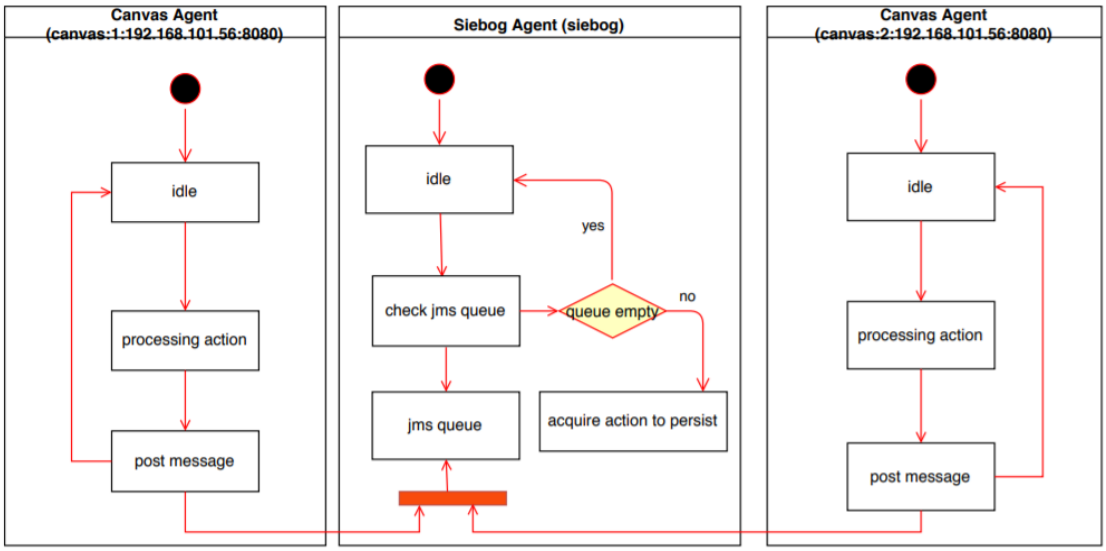

Figure 2. Communication between a Siebog (master) agent and Canvas agents. Both Canvas agents serve one instance of a Canvas server (192.168.101.56:8080) that is currently running on a VM. And they are bound two to different courses.

- Delete file:
  - remove the specified file (DELETE /api/v1/files/:id)

The file download and upload functionality are a combination of more than one API call and they are also implemented in this project. The other REST API calls that were added to the Canvas API support the Folder API:

- List folders – returns the paginated list of folders in the folder (GET /api/v1/folders/:id/folders)
- List of all folders – returns the paginated list of all folders for the given context:
  - by course id (GET /api/v1/course/:course_id/folders)
- Get folder – returns details for a folder by:
  - course id and folder id (GET /api/v1/course/:course_id/folders/:id)
  - folder id (GET /api/v1/folder/:id)
- Update folders – udpates a folder (PUT /api/v1/folders/:id)
- Create folder – create a folder in the specified context
  - by course id (POST /api/v1/courses/:course_id/folders)
  - by folder id (POST /api/v1/folders/:folder_id/folders)
- Delete folder – remove the specified folder (DELETE /api/v1/folders/:id)

All the previously listed calls in the File and Folder API have a list of parameters that accompany each method.

B. Siebog modification

Canvas servers communicate with a Siebog middleware, which has a finite number of agents associated with courses. Each agent has a course that it is bound to. An agent is described with a tuple: canvas name, course name and the date of the last scan. The previously mentioned elements represent the following:

- canvas name – an IP address and the port that the canvas server is running on,
- course name – represents the course that this agent manages,
- and date of the last scan – is the last date that a scan was performed in order to acquire any changes on the course content.

An agent's job is to manage the content of the course it is bound to. This is possible because agents communicate by exchanging messages. An agent can execute two types of operations: request and response. When an agent executes the request operation, it is trying to synchronise its content with other agents' content. The replication process requires parameters that indicate the name of the corresponding agent from the remote course that it is bound to and the last scanned date. With the request

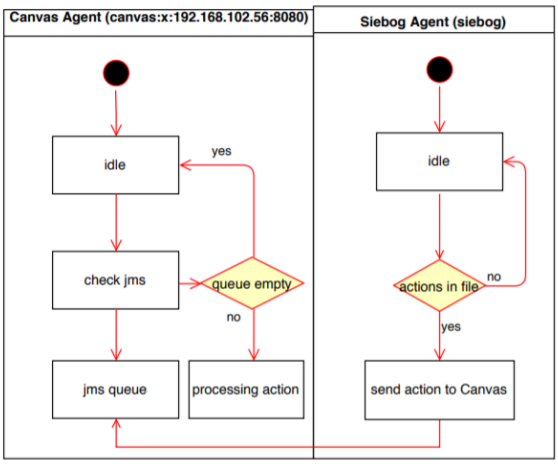

Figure 3. The synchronisation process between a Siebog (master) agent and a Canvas Agent that serves one instance of a Canvas server (192.168.101.56:8080) that is currently running on a VM

operation, an agent gets specific parts of the remote course content, according to a certain criterion. For now, the criterion is the date of the last performed scan. After this, the content of the course is updated. The response operation delivers a requested course content to a requester agent.

In Figure 2. a scenario of communication between two agents is presented. When the Sibeog application is started one master agent called Siebog will be started as well. This agent is in an idle state and it is awaiting requests from other agents. On startup or during runtime several Canvas agents are started in the initial idle state awaiting requests from clients (users). When a canvas agent receives a request from a user it will process the action and with the use of Canvas API will send a request to the canvas server in the acquire format depending on the request. If the servers' response is successful, then the canvas agent sends a request to the master agent to save this action locally in the Siebog applications' filesystem.

Figure 3. Represents the synchronization process between two courses that are in two different canvas applications. The replication process can be triggered right after the canvas agent completes the task that was defined from an end user, or it can be triggered in an arbitrary time during runtime. The process begins with the Siebog agent looking up actions in the file that contains list of previous actions on different canvas applications. Depending on the chosen corresponding canvas application the master agent will send inline actions to the canvas agents that are coupled with the corresponding canvas server. The canvas agents will obtain the actions from the master agent, and evaluate them, and with the use of Canvas API post a message to the Canvas application that it is tied to.

The proposed solution gives the user more control over the choice of data to be replicated. Another solution is the manual synchronization. The user can start the synchronisation process by selecting a running canvas server and a course name. Siebog stores dates of last performed scans of the chosen servers internally. These dates are compared with the modification or creation dates of the files and folders on the selected servers and courses. If the date of the last scan is before the modification (or creation) date, a list of actions will be queued in Siebog. During the process of course synchronisation, one course will always be a master course. A master course represents a blueprint, and selected changes in this course will overwrite other changes done in the associated course. The modifications made in the associated course after the scan date will not be acknowledged, and they will be overwritten. The user chooses which course will be the master and which one will be the associated course. This way we have more knowledge on which files or folders will be affected and possibly overwritten.

In comparison to other approaches, with the aid of the Siebog middleware, there is a stronger control over which content of a course will be synchronised. The job of replicating data handled by the proposed solution is easier for maintenance and improvement.

In this paper we gave an example of the Siebog application as the multi-agent middleware required for the communication between two Canvas applications. The exchange of messages between these two Canvas servers is possible through the Canvas API. The Canvas applications are started on two different VM that have the Linux OS installed, while the Siebog application is started on the Windows 10 OS.

IV. Conclusion

Siebog, our multi-agent middleware, is data and platform independent. As such, it can be easily integrated as a middleware for communication between more applications.

For the future work we plan to expand the current project with a greater number of regular actions sent toward Canvas servers and synchronisation actions for Canvas agents to perform. This will show that with the aid of a system that is agent-based like Siebog we can have more performance enhancement in handling multiple canvas instances and users that use the Siebog application through a simple user interface. Since Siebog is loosely tied to the Canvas application our goal also is to have this middleware system manage the replication of data between other LMS systems besides Canvas.

Acknowledgment

This work has been partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia (project III 44010).

References

[1] Long, Phillip D. "Learning Management Systems (LMS)." Encyclopedia of Distributed Learning. Eds. Anna DistefanoKjell E. Rudestam and Robert J. Silverman. Thousand Oaks: SAGE Publications, Inc., 2004. 291-293. SAGE Knowledge. Web. 21 Apr. 2019.

[2] Instructure, "Learning Management System | LMS | Canvas by Instructure". Retrieved from https://www.canvaslms.com/

[3] ADL, "SCORM Explained: In Depth Review of the SCORM eLearning Standard". Retrieved from https://scorm.com/scorm-explained/

[4] IMS Global Learning Consortium, "IMS Content Packaging Best Practice Guide". Received from https://www.imsglobal.org/content/packaging/cpv1p1p3/imscp_bestv1p1p3.html

[5] IMS Global Learning Consortium, "IMS Question and Test Interoperability Integration Guide". Received from https://www.imsglobal.org/question/qti_v2p0/imsqti_intgv2p0.html

[6] D. Mitrović, M. Ivanović, M. Vidaković, Z. Budimac, "The Siebog Multiagent Middleware", Knowledge-Based Systems, vol. 103, no. C, pp. 56-59, July 2016.

[7] D. Mitrovic, M. Ivanovic, M. Vidakovic, Z. Budimac, A scalable distributed architecture for web-based software agents, in: Seventh International Conference on Computational Collective Intelligence Technologies and Applications, in: LNCS, 9329, 2015, pp. 67–76.

[8] Kansas State University – ITS, "Canvas API Library for Java", GitHub repository, https://github.com/kstateome/canvas-api

[9] Instructure, "Canvas LMS REST API Documentation - Canvas Instructure". Retrieved from https://canvas.instructure.com/doc/api/