Usage Pattern Recognition in Student Activities

Maren Scheffel[1], Katja Niemann[1], Abelardo Pardo[2], Derick Leony[2], Martin Friedrich[1], Kerstin Schmidt[1], Martin Wolpers[1], and Carlos Delgado Kloos[2]

[1] Fraunhofer Institute for Applied Information Technology FIT, Schloss Birlinghoven, 53754 Sankt Augustin, Germany
[2] Universidad Carlos III de Madrid, Avenida Universidad 30, E-28911, Leganés (Madrid), Spain
{maren.scheffel,katja.niemann,martin.friedrich}@fit.fraunhofer.de, {kerstin.schmidt,martin.wolpers}@fit.fraunhofer.de, {abel,dleony,cdk}@it.uc3m.es

Abstract. This paper presents an approach of collecting contextualized attention metadata combined from inside as well as outside a LMS and analyzing them to create feedback about the student activities for the teaching staff. Two types of analyses were run on the collected data: first, key actions were extracted to identify usage patterns and tendencies throughout the whole course and then usage statistics and patterns were identified for some key actions in more detail. Results of both analyses were visualized and presented to the teaching staff for evaluation.

Keywords: attention metadata, usage monitoring, teacher feedback.

1 Introduction

Collecting data about how students interact with each other, the teaching staff and learning objects is becoming increasingly complex. Traditional approaches place a Learning Management System (LMS) at the center of these interactions [1]. Such systems are typically used as a container to store learning resources supplied by the teaching staff and offer a variety of services to capture the interaction between the users and other elements, e.g. discussion forums, quizzes, wikis, blogs, etc. This approach can be called a "LMS-centric" one. With the arrival of the Web 2.0, however, this approach is no longer functional [2]. Learning activities, sometimes even without the consent of the instructors, often take place within numerous applications that are hosted outside the LMS. Some of these activities may explicitly require the use of certain tools that must be installed outside the LMS, e.g. an editor, and others require students to search for auxiliary material that is not stored in the LMS.

The use of tools not hosted in the LMS is specially common in experimental sciences [3] where students are usually required to learn procedures with specific tools either installed on their personal computers or on equipment made available by the educational institutions. This fact and the tendency of students to access an increasing number of resources outside an institution's LMS [4] gives rise to the need for more sophisticated data collection mechanisms.

This paper presents an approach of collecting usage data from inside a LMS (i.e. forum interactions) and outside a LMS (i.e. use of a virtual machine as a self-contained course environment) and analyzing them to create teacher feedback about the course they were taken from. In section 2 we motivate our approach and present previous works connected to this topic while section 3 describes what kind of usage data we collect, how we collect it and how we store it. After explaining the analysis methodologies we employed in section 4, analysis results are presented in section 5. Finally, section 6 concludes our paper and discusses further work.

2 Related Work

Modifying a learning scenario based on user observations is a research topic that has been attracting significant attention. Areas such as intelligent tutoring systems [5] propose the design of applications that observe and react to the user actions. But these applications have the same drawbacks as LMSs: If a learning experience occurs in a context not only involving the tutoring system, the truly valuable observations would most likely be those of the student interacting with the whole environment, and not only the ones with the tutor.

Zinn and Scheuer [6] conducted a survey among teachers trying to identify requirements for student tracking tools. Among the information deemed mostly important were the students' overall success rate, the mastery level of concepts, skills, methods and competencies as well as the most frequently diagnosed mistakes. As for the reasons why they would employ student tracking at all, most teachers said that they would like to be able to respond to individual students, adapt their teaching and identify problems of understanding the material. One such approach to provide teachers with feedback about their courses is described by Jovanovic et al [7]. They present a tool called LOCO-Analyst that analyses user tracking data based on an ontological framework. It was developed specifically for online learning courses and thus only collects data from within the LMS. Kosba et al [8] also present a framework that generates feedback for teachers from tracking data collected within a course management system. Here, however, the generated advice bases on student, group and class models which are computed with the help of certainty factors and fuzzy set theories. Again, the system was applied to an online (distance) learning course. TADA-Ed (Tool for Advanced Data Analysis in Education) is a data mining tool by Merceron and Yacef [9] with the aim to support teachers in discovering pedagogically relevant patterns in user activity data collected in online exercises using a web-based tutoring system. Therefore, it offers visualizations of simple statistics (e.g. mistake frequency) and results of data mining algorithms (e.g. clustering students based on the concepts of their mistakes). However, TADA-Ed presupposes that the users (i.e. teachers) are familiar with data mining techniques and competent enough to choose and apply them as well as to analyze their results which can be quite complex.

Extending the range of observable events in learning environments is not a trivial task. In principle, a student may engage in a course activity interspersed with other activities or at unexpected times. In the rest of the paper the concept of *observability* will be restricted to the realm of computer use. Given the wide variety of applications and systems that can potentially be used in a course, it seems obvious that an important improvement would be to move the recording capabilities outside of the LMS and onto the students' personal computer. There are already certain applications that record these events which can then be shared in a community of users (e.g. Wakoopa[1]). In a learning environment though, the need is to monitor only those events that are related to the course.

3 Data Collection

This section describes what usage data was collected and how. First, the schema in which the collected data is stored is explained. We then present the data collection processes from several sources and finally address data privacy issues.

3.1 Contextualized Attention Metadata

The Contextualized Attention Metadata (CAM) schema allows modelling a user's handling of digital content across system boundaries [10]. As CAM was developed to describe as many types of attention metadata as possible, CAM records of a user cannot merely describe the user's foci of attention but rather his entire computer usage behaviour. The latest version[2] is better suited for evaluating and analyzing user observations due to its event centredness. Figure 1 shows an image of the new schema.

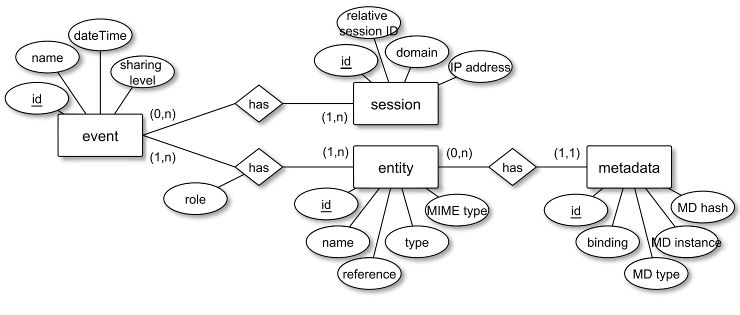

Fig. 1. Contextualized Attention Metadata (CAM) Schema

While the older versions of the CAM schema revolved around the user, the new schema has the `event` element as its entry point. Every user action is described by one `event`, e.g. *UserA sends MessageX to UserB in ApplicationY*

[1] http://wakoopa.com/

[2] More information and software about CAM can be found at https://sites.google.com/site/camschema/home

and several attributes define this event in more detail: the `name` attribute (name of the event, e.g. open - for opening a document) and the `dateTime` attribute (timestamp of the event, e.g. 20111504-123428). As opposed to these two attributes, the `sharing level` attribute is optional. Every `event` is connected to at least one `session` and every `session` can be defined by attributes for `IP address`, `domain`, and a `relative sessionID`.

An `event` can be connected to several `entities` that represent the participants of this event. The connected `entity` has to be further specified with a `role`, e.g. in the action mentioned above *UserA* has the role *sender*, *UserB* has the role *receiver*. `Entity` elements are defined by attributes for `name`, `MIMEtype`, `type` and `reference`. An `entity` in turn can have `metadata` attached to it. This `metadata` element is defined by a `binding`, `MDtype`, `MDinstance` and `MDhash`.

Due to the simplicity of this schema, a lot of knowledge has been moved to the `role` attributes. Also, the schema now requires rules to be enforced on the instances that go beyond simple cardinality restrictions, that is, the cardinality depends on the type of the value of `role`. For example: if there is a related `entity` with the `role` attribute *receiver*, there needs to be exactly one related `entity` with the `role` attribute *sender*.

CAM can be analyzed to provide an overview about where (i.e. with which application) and when an action takes place and what happens in the environment. CAM analyses enable the discovery of popularity, usage bursts and trends of tools. Information about when an action takes place can be useful in controlled environments such as formal learning environments where activities are usually scheduled. It is also quite useful in less controlled or more blended environments to understand when learners are actually active. The next sections therefore describe how CAM was collected during programming courses at the Universidad Carlos III de Madrid and then analyzed.

3.2 Collecting Data from the Virtual Machine

The concrete scenario where our approach has been deployed was a second year course of an engineering program. The course topic was C programming using mobile devices, and students were supposed to use a set of industry-type tools as well as work in teams for the course activities. The proposed approach consisted of offering the students a fully configured virtual machine containing all the tools required to carry out the course activities. A virtual machine is a self-contained computer captured in a single file that can be emulated by a client application in a different computer. This emulation is also called "virtualization" and has been considered in some learning scenarios to eliminate configuration problems [11].

The virtual machine was created by the teaching staff before the beginning of the course with a configuration specially tailored to the course activities. As part of that configuration, the main tools inside the virtual machines have been modified so as to record certain events in an internal database. Those recorded events were regularly relayed to a central server that collects them. The virtual machine made available to the students has been configured with the following tools: compiler, debugger, memory profiler, file editor, version control system, emulator, command line interface, browser and an integrated virtual environment (IDE).

A significant number of course activities require writing code fragments and testing their execution. The compiler, debugger and memory profiler are required to carry out these tasks. For the compiler the date and time when the compilation is invoked, the error and warning messages, and the final status of the compilation are recorded. The debugger is essential when developing C applications and students must be proficient in its use. The date and time the tool was started and closed, as well as all the commands used during the session are stored. The memory profiler executes and detects memory leaks. The invocation and the messages produced by the application are recorded together with a time stamp. The Command Line Interface is being used in the course to invoke most of the tools as well as carrying out regular operations such as creating folders, moving files, etc. Therefore, all commands are stored together with a time stamp. C programs can be edited either using a regular file editor or an integrated virtual environment. The programs are shared among team members and with the teaching staff using the version control system Subversion[3]. The date and time when the editor or IDE are invoked and closed are recorded. Additionally, the browser can be considered as yet another development tool that students will use frequently while working in the course. For that reason, it is also used to retrieve the visited URLs.

Each tool stores the recorded events at a fixed location in the user's home folder. Users should be able to easily enable/disable the recording mechanism. This feature has been implemented as a simple test for the existence of the folder where the events are recorded. If such folder is removed, the tools no longer record any event. The client application of the version control system Subversion was modified so that every `commit` operation that sends the current version of a file to the server would actually send two sets of changes: The normal set of changes in the files of the specified directory, and a second set of changes including all the files with the events recorded from the last transmission. As a consequence, every time a student submits a new version of a project, all recorded events are submitted to the server. The transmission has been deployed so that the system works incrementally and only those events produced from the last transmission are sent. Typically, a Subversion repository requires user authentication. As they are part of the same subversion session, the recorded events are submitted to the server with the same user authentication as the rest of files. Once the recorded events are submitted they are transformed into CAM instances.

3.3 Collecting Data from the LMS

Extending the observation of student tasks outside of the LMS does not mean that the data collected inside the LMS is no longer valid. On the contrary, the events taking place in the LMS are yet another ingredient to add to the overall observation mechanism.

[3] http://subversion.apache.org/

The analysis of usage behaviour within the LMS is made possible by reworking the Web server logs of the system. Every line in the log file represents one event and contains information such as date and time, the initiator's user id, type of event, etc. After collecting the logs a grammar consisting of regular expressions was constructed. With the help of this grammar the logs can be grouped into meaningful user actions. For example, the complex event of posting a new message in a forum consists of several single user-initiated actions in the logs (i.e. the user needs to click on a *new post* button which initiates the loading of a new page, where he can write the post to then click on the *send* button which initiates the loading of the forum page with the new message).

Given this file and the stored logs, an event was created whenever a pattern was identified and transformed into a CAM instance. Any static information needed for later analysis but not contained in the log files, e.g. name of a course, were pulled from the database the LMS is based on. Using this approach, events such as posting in a forum, replying to a post, looking at a message, logging in or logging out can easily be detected in the server logs.

Due to synchronized server time between the virtual machine and the LMS merging of the CAM instances from the two was unproblematic. The combined data were then stored in a relational database from which all further analyses were done.

3.4 Data Privacy Issues

Extending the observation of student activities with the described approach poses certain concerns on data privacy. The first observation is that by using a virtual machine, only those events that take place inside the machine (and not on the host) are being recorded. Students are instructed at the beginning of the course to use this machine *only* for course related tasks. By simply adhering to this policy (which cannot be strictly enforced), the recorded events are all relevant.

But current legislation about data privacy imposes additional constraints that have to be reflected in the deployed system. The first one is that users should be made explicitly aware of the type of data that is being collected. The virtual machine was configured so as to start with a user session with the browser showing a page explaining the details of the recording mechanism. Second, the recording system must be easily disableable by the user. As it was described in the previous section, the presence or absence of a concrete folder is used to switch the system on or off respectively. And third, and most importantly, users need to be made aware of how to contact the person/institution in charge of the data because they maintain the right to query and request the deletion of any of the data.

4 Analysis Methodologies

Two types of methodologies have been applied to the collected CAM. In a first step we extract key actions to identify usage patterns and tendencies throughout the whole programming course. Then, in a second step, some sequences of events are looked at in more detail and usage statistics and patterns for these sequences are identified.

Our first approach, extracting key actions, is to analyze CAM with techniques used in the research field of computational linguistics, that is, we transfer methodologies from text analysis to action analysis and try to find patterns within the recorded activities. Languages are rule-governed [12], i.e. they are based on patterns and structures. We exploit this fact and apply methodologies to detect and analyze such patterns to CAM by mapping linguistic concepts to the respective parts of CAM instances which is, however, not a simple one to one mapping. The concepts word and action can quite easily be seen as analogous but sentence and session cannot: While sentences are fixed linguistic categories, sessions describe a concept of time with a variable beginning and end (e.g. an hour, a few minutes, a month, a year, etc.). Mapping session to the concept of text, i.e. a collection of sentences, possibly makes more sense. As the word-to-action-mapping is very reasonable, we start our approach of detecting meaningful patterns from CAM by transferring methods from keyword extraction to key action extraction. The content of a document can be semantically represented by keywords [13]. We thus assume that key actions can semantically represent the session they are taken from. In order to find repeated string patterns we analyze the collected CAM with the so-called n-gram approach [14]. The following example illustrates the technique in a simplified way:

$$_0\,\mathtt{A}\,_1\,\mathtt{B}\,_2\,\mathtt{C}\,_3\,\mathtt{A}\,_4\,\mathtt{B}\,_5\,\mathtt{D}\,_6\,\mathtt{B}\,_7\,\mathtt{C}\,_8\,\mathtt{A}\,_9\,\mathtt{B}\,_{10}\,\mathtt{A}\,_{11}\,\mathtt{A}\,_{12}\,\mathtt{C}\,_{13}\,\mathtt{D}\,_{14}$$

The letters represent a sequence of actions while the numbers indicate the position of the actions within the sequence. In a first step, all possible monograms are extracted:

A	[0,1],[3,4],[8,9],[10,11],[11,12]	C	[2,3],[7,8],[12,13]
B	[1,2],[4,5],[6,7],[9,10]	D	[5,6],[13,14]

The merging of n-grams is possible if the frequency of the new key action is above a set threshold. Lets assume the threshold in this example was set to 2. As no monograms are below that threshold, none get discarded from further calculations. In a next step, all possible bigrams are extracted by trying to combine the monograms with one another:

AA	[10,12]	BC	[1,3],[6,8]	CA	[2,4],[7,9]
AB	[0,2],[3,5],[8,10]	BD	[4,6]	CD	[12,14]
AC	[11,13]	BA	[9,11]	DB	[5,7]

The bigrams `AA`, `AC`, `BD`, `BA`, `CD` and `DB` only occur once. Hence, they are discarded from further calculations and can consequently neither be a key action nor part of one. Additionally, if shorter sequences are contained in longer ones, the shorter ones are no longer part of further calculations and thus no key action candidate. In our example the monograms `A`, `B` and `C` are discarded but the monogram `D` stays as it is not contained in any of the remaining bigrams `AB`, `BC` and `CA`. These now pose as the basis for trigram extraction, those are turned into tetragrams, then pentagrams and so forth. Calculations end when no further n-grams can be merged. Our example ends with two n-grams, namely the twice occurring tetragram `BCAB` and the monogram `D`, and thus two key actions.

Once the key actions are identified, our second approach is to look at some of them in more detail. Depending on the domain of interest or the focus of the analysis, some actions can be deemed more important than others and a detailed analysis can help the teacher to become more aware of what problems students might run into. Once the sequences to be looked at in more detail are identified, they can be analyzed further and displayed to the teachers, e.g. in the form of text files, tables or diagrams. Interesting analyses can, for example, be the distribution of certain events over time or students, common subsequent actions of relevant actions in the current domain, anomalies, tendencies or regularities in the usage behaviour.

5 Analysis Results

To analyze and evaluate our approach we collected data from the C programming course mentioned above. First we extracted key actions from the collected data on several granularity levels and presented our results to the teaching staff. Then we ran several pattern detection analyses on those actions identified as worth looking at by the teachers in more detail and presented the results to the teaching staff again.

5.1 Key Action Extraction

The C programming course that our analyses base on took place from September 6th until December 16th, 2010 and 244 students attended the course. In that period of time, around 10,000 sessions were recorded comprising a total of 119,652 events which in turn contained 19 different event types. There are some events that do not contain any additional information as the event name itself as well as the date and time of occurrence (e.g. `startDebuggerGDB` or `endTextEditorKate`). However, most events contain one additional attribute, e.g. the ID of the viewed message for the event `viewMessage`, the error message for an unsuccessful compile event or the URL for the event `visitURL`.

In order to find key actions, i.e. frequent patterns in the data, we distinguish between two granularity levels of the data: a short and a long version. In the short version, only the events and their timestamps are considered but no further attributes. In the long version, all attributes (e.g. forum IDs and warning messages) are completely used for the calculation of key actions. There is one exception for the event `visitURL` that can be combined with the short or with the long version, i.e. only the domain of the URL can be considered instead of using the whole URL (long version) or no URL at all (short version). For example, we would assume the pattern of several students getting the same error and conducting a google search subsequent as very meaningful, even if they do not use the exact same query terms. If we want to detect such patterns, we need to use the long version, but shorten the URLs of the event `visitURL` to their domain.

As exploratory investigation, all variations were calculated with a threshold of 5 as well as 10. We extracted key actions on the basis of the whole course, that is, all sessions from all users were taken into account at the same time. Thus, if an action was identified as a key action with 15 occurrences, it could be due to several different users executing the action but also to one user executing it 15 times.

A visualization for the results was implemented and given to the teaching staff for evaluation. The version deemed most useful to deduce meaningful things from was the long version with the threshold set to 10. By analyzing the sequences of event patterns, the teachers discovered several situations interesting to them as they denoted incorrect procedures. Figure 2 shows an example of such a visualization. The darker boxes mark the first event of a key action sequence. The key action sequence in the upper right corner for example starts with a `gotoForum` event, followed by `viewMessage`, another `gotoForum` event and two more `viewMessage` events.

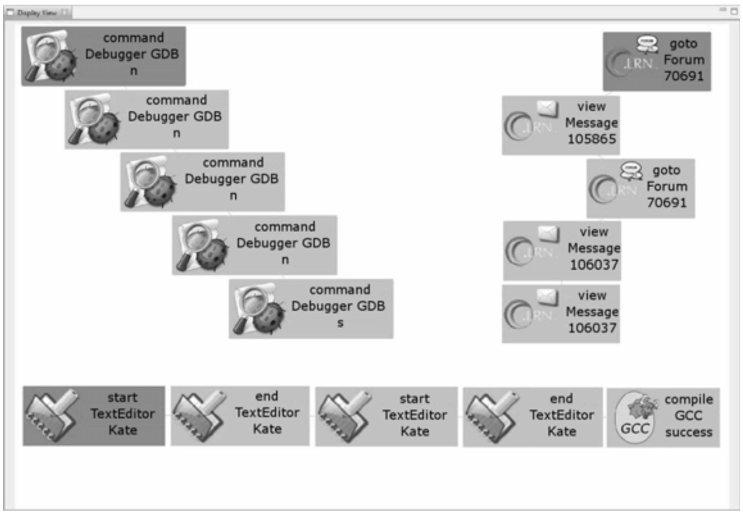

Fig. 2. Visualization of extracted key actions

Some sequences identified as interesting by the teaching staff contained events where the text editor was started and ended. This points to development flows in which for each compilation, students opened a file with the editor and closed it again before compiling. According to the teaching staff this translates into a significant increase in the development time and should be corrected. Other sequences of events to access the course material also pointed to some possible corrections. The course material was available through a direct link as well as through a redirection at a second official location. The second route was followed by a significant number of users, thus pointing to the need for a teacher to emphasize how to quickly access the course material. Another example identified as an important one were sequences where tools such as the debugger or the memory profiler were used several times which denotes the students' difficulty in understanding the anomalies detected by these tools. As the teaching staff considered error messages and the students' reaction to them as especially valuable, we analyzed these events in more detail.

5.2 Frequent Error Patterns

One of the most significant difficulties students face in introductory programming courses is how to understand the messages returned by the compiler when writing a program. There is an important gap between learning a programming language and quickly identifying an anomaly spotted by the compiler during development. This gap typically requires an unusually high amount of time from students to be closed. Tutor assistance in these situations is very important. However, even if the students are well aware of the fact that they are observed, we cannot assume them to be aware of the effects that the analysis of their data can have. We assume that showing a teacher some statistics of a user who for example compiled several times getting the same (trivial) error over and over again, might lead to the teacher getting negatively affected. Therefore, we only showed statistics of errors that occurred at least 50 times in total for a minimum of 10 users. Additionally, we made the students' usage patterns more abstract, e.g. `too few arguments to function usernameTest` was shortened to `too few arguments to function...` . This process helps combining activities when looking for patterns but also keeps the anonymity of the students.

Within the collected data there were 17,266 `compile` events. 6,443 of them were error messages, 3,190 were warnings and 7,633 events compiled successfully. Figure 3 shows those errors with the highest frequency and that at least 25 different user received ordered by the number of students that got it. From this diagram, instructors were able to easily identify those anomalies that require clarification. Teachers liked the fact that with the obtained data, they may even select a sample of the errors and use them as examples to guide students to detect and correct the most frequent anomalies.

We also calculated the distribution over time of the most frequent errors and asked the teaching staff to evaluate whether such a visualization can be useful. Figure 4 shows the distribution of the most frequent error, i.e. `'foo' undeclared (first use in this function)`, over time and the number of students that got this error. In total the error occurred 1048 times for a total of 92 users. As it can be seen, students encountered this error even in the final stages of the course (when there was an increase on student load due to a project deadline). According to the teaching staff corrective actions taken at the beginning of the course might have helped to lower the error's appearance.

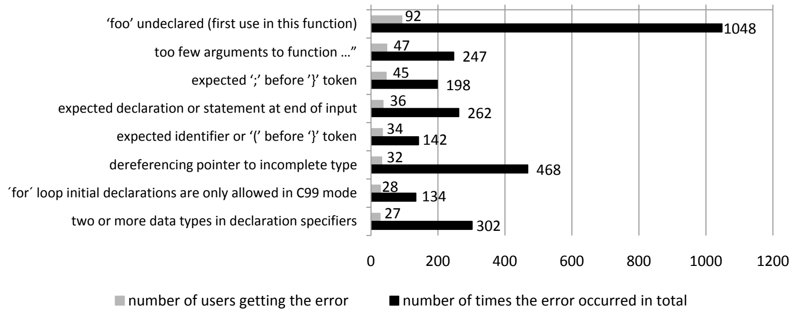

Fig. 3. Distribution of the most frequent errors

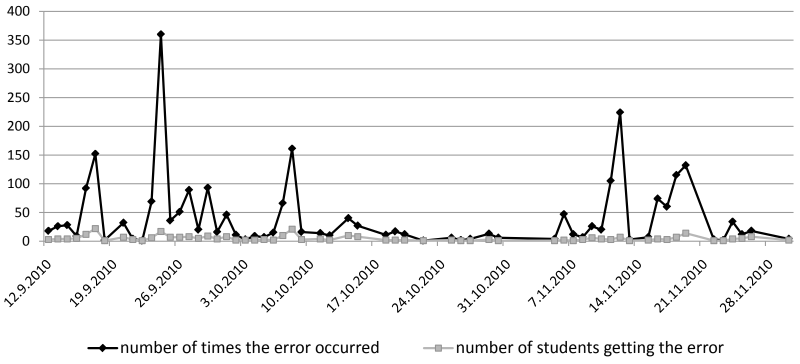

Fig. 4. Error `'foo' undeclared (first use in this function)` over time

Figure 5 shows the distribution over time of the error `two or more data types in declaration specifiers`. In total it occurred 302 times for a total of 27 users. The error disappears over time which, according to the teaching staff, shows that the main problem of understanding why the error occurred got solved by the students. However, as the error occurred very frequently at the beginning of the course, teacher support could have helped in improving the learning curve faster.

Another interesting aspect the teaching staff wanted to have more detailed information about was what the students did after getting error messages, i.e. what kind of events were `compile_ error` events followed by. We therefore created some diagrams to help identify more precisely the nature of the errors encountered by the students. Figure 6 displays the actions that students performed directly after receiving the error message `'foo' undeclared (first use in this function)`. It confirms that the error is persistent, that is, it tends to appear again in the next compilation. This information is essential for teachers to pinpoint those anomalies that students are not understanding, thus giving rise to important time delays.

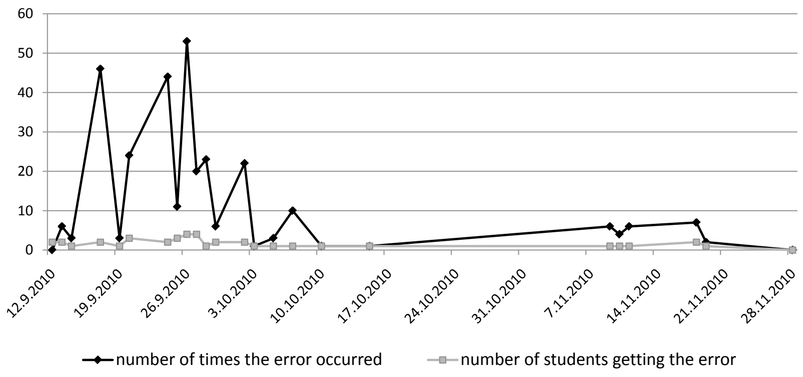

Fig. 5. Error `two or more data types in declaration specifiers` over time

Figure 7 shows the detail of a third compilation error, i.e. `expected ‘;’ before ‘}’ token`. It occurred 198 time in total for a total of 45 users. In this case we can see that the next events are sometimes a correct compilation, sometimes the appearance of some warnings, and in some other cases the error re-appears. This error clearly does not have the “road-block” nature, i.e. not knowing what to do next and trying random changes in the code, identified in the previous two.

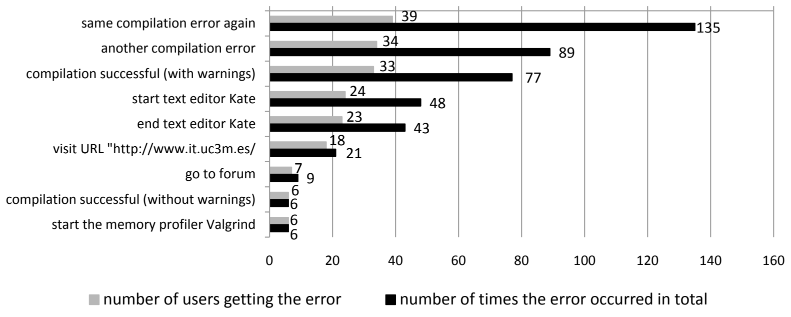

Fig. 6. First actions after error `‘foo’ undeclared (first use in this function)`

These diagrams show the power of offering visualizations of events occurring in every-day student work sessions to the teaching staff. If these events occurred in a face-to-face session, the instructor would quickly point the students to techniques to identify and solve the anomaly. With the proposed paradigm, these patterns can still be detected and brought to the attention of the instructor when they occur in any course working session and as a consequence of the analysis of these data, several corrective actions can be deployed by the instructors. They may range in complexity from discussing the most common mistakes in class and making sure students identify the errors correctly, to an automatic support system that detects when certain specific errors are produced by the compiler and some additional explanation is given live to the student (a pop-up message, an email, a URL pointing to a more detailed explanation, etc.).

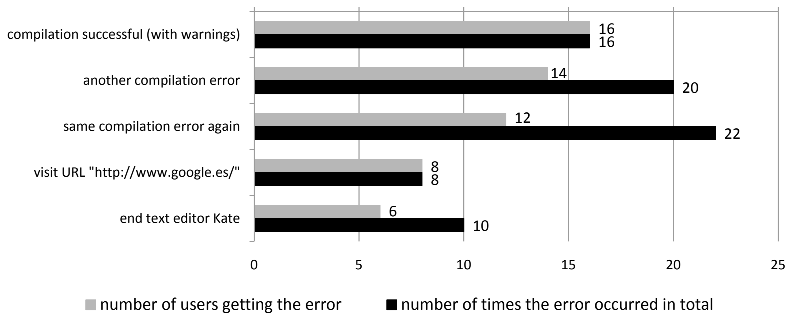

Fig. 7. First actions after error `expected ';' before '}' token`

6 Conclusion

In this paper an approach to analyze user activities and detect usage patterns in the context of a programming course has been presented. Contextualized Attention Metadata (CAM) was used to represent the multiple events occurring in a learning environment. The capturing mechanism employed extends the conventional methods that rely solely on those events recorded in the logs of a Learning Management System. A virtual machine specifically configured for the course was distributed among students for the purpose of being used in all the course activities. This machine deployed a high coverage recording mechanism for the events occurring in the previously installed tools. These observations were also complemented with the corresponding events derived from the LMS.

Once the events have been captured, techniques were applied to first extract key actions from the collected data, and then to identify usage patterns by exploring in detail sequences of events. With these techniques discovering patterns in large collections of events was successful where no manual technique is feasible.

The procedure was validated by applying it to an undergraduate engineering course on introductory programming in C and a large number of events was collected (almost $120,000$ events of 244 students). First a visualization of the key actions (i.e. frequent patterns of events) was produced. The visualization showed certain sequences that clearly point to corrective actions to be deployed. Second an analysis of the most frequent error patterns was performed. Based on this analysis, the teaching staff identified some errors as most problematic that should be discussed in more detail in the course.

Two approaches are considered as future research lines. The first is to support reflection and awareness of students. Self-reflection allows students to meta-cognitively assess, analyze and evaluate their learning processes. It is also an essential part of self-regulated learning [15]. Furthermore there is the need for direct feedback to the students in their learning environment [16]. This can be accomplished by using the version control system to transfer information directly into the user desktop.

For the second approach, the reflection and awareness support of the teachers, we will embed the graphical analyses that the teaching staff graded as especially helpful in our tool. This will enable the teachers to directly react in class on the usage patterns of the students. Once the merit of the approaches has been established by the experimental results, techniques for the automation of the corrective actions can be explored. Taking advantage of the use of the virtual machine, an algorithm that reacts when certain patterns are detected can be designed and deployed. Furthermore, the detection algorithms can be refined as to tackle specific problems within a course. The teaching staff may describe activities that might require a closer analysis so that more targeted corrective actions can be discovered.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no 231396 (ROLE project). Work was also partially funded by the Learn3 project (TIN2008-05163/TSI), the eMadrid project (S2009/TIC-1650), and the Acción Integrada DE2009-0051.

References

1. Ertugrul, N.: Towards virtual laboratories: A survey of LabVIEW-based teaching/learning tools and future trends. International Journal of Engineering Education 16(3), 171–180 (2000)
2. Chatti, M.A., Jarke, M., Frosch-Wilke, D.: The future of e-learning: a shift to knowledge networking and social software. International Journal of Knowledge and Learning 3(4), 404–420 (2007)
3. Auinger, A., Ebner, M., Nedbal, D., Holzinger, A.: Mixing content and endless collaboration–MashUps: Towards future personal learning environments. Universal Access in Human-Computer Interaction. Applications and Services, pp. 14–23 (2009)
4. Waycott, J., Bennett, S., Kennedy, G., Dalgarno, B., Gray, K.: Digital Divides? Student and Staff Perceptions of Information and Communication Technologies. Computers & Education 54(4), 1202–1211 (2010)
5. Woolf, B.P.: Building intelligent interactive tutors. Morgan Kaufmann, San Francisco (2008)
6. Zinn, C., Scheuer, O.: Getting to Know Your Student in Distance Learning Contexts. In: Nejdl, W., Tochtermann, K. (eds.) EC-TEL 2006. LNCS, vol. 4227, pp. 437–451. Springer, Heidelberg (2006)
7. Jovanović, J., Gašević, D., Brooks, C.H., Devedžić, V., Hatala, M.: LOCO-Analyst: A Tool for Raising Teachers' Awareness in Online Learning Environments. In: Duval, E., Klamma, R., Wolpers, M. (eds.) EC-TEL 2007. LNCS, vol. 4753, pp. 112–126. Springer, Heidelberg (2007)

8. Kosba, E., Dimitrova, V., Boyle, R.D.: Using Student and Group Models to Support Teachers in Web-Based Distance Education. In: Ardissono, L., Brna, P., Mitrović, A. (eds.) UM 2005. LNCS (LNAI), vol. 3538, pp. 124–133. Springer, Heidelberg (2005)
9. Merceron, A., Yacef, K.: TADA-Ed for Educational Data Mining. Interactive Multimedia Electronic Journal of Computer-Enhanced Learning 7(1) (2005)
10. Schmitz, H.-C., Wolpers, M., Kirschenmann, U., Niemann, K.: Contextualized attention metadata. In: Roda, C. (ed.) Human Attention in Digital Environments, pp. 186–209. Cambridge University Press, Cambridge (2011)
11. Gaspar, A., Langevin, S., Armitage, W.D., Rideout, M.: March of the (Virtual) Machines: Past, Present, and Future Milestones in the Adoption of Virtualization in Computing Education. Journal of Computing Sciences in Colleges 23(5), 123–132 (2008)
12. Sag, I.A., Wasow, T.: Syntactic Theory – A Formal Introduction. Center for the Study of Language and Information, Stanford (1999)
13. Rose, S., Engel, D., Cramer, N., Cowley, W.: Automatic Keyword Extraction from Individual Documents. In: Berry, M.W., Kogan, J. (eds.) Text Mining, pp. 1–20. John Wiley & Sons, Ltd., Chichester (2010)
14. Hulth, A.: Improved Automatic Keyword Extraction Given More Linguistic Knowledge. In: Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing, EMNLP 2003, Stroudsburg, PA, USA, pp. 216–223. Association for Computational Linguistics (2003)
15. Zimmermann, B.J.: Self-regulated learning and academic achievement: An overview. Educational Psychologist 25(1), 3–17 (1990)
16. Butler, D.L., Winne, P.H.: Feedback and self-regulated learning: A theoretical synthesis. Review of Educational Research 65(3), 245–281 (1995)