Book Title
Book Editors
IOS Press, 2003

Empowering researchers to detect interaction patterns in e-collaboration

Andreas Harrer [a,1], Rakheli Hever [b], Sabrina Ziebarth [a]
[a] *Universität Duisburg-Essen, Germany*
[b] *The Hebrew University of Jerusalem, Jerusalem, Israel*

Abstract. This paper describes an approach to support practitioners in the analysis of computer-supported learning processes by utilizing logfiles of learners' actions captured by the system. We enable researchers and teachers to identify and search for insightful patterns of the learning process in two ways: on the one hand patterns can be specified explicitly to be searched in logfiles; on the other hand automatic discovery of patterns that match configurable parameters is used to find most typical sequences in the logfiles. Both features have been realized in a stand-alone tool that accepts generic logfiles usable with a potentially wide variety of different learning systems. This is shown in an example of practical use of this tool in a project that supports researchers and moderators of graphical electronic discussions.

1. Introduction

CSCL-Systems (Computer Supported Collaborative Learning) provide a mediating function for collaboration, but recently also additional functionality for (pro-)active support of the collaboration for all people involved in CSCL, i.e. students, teachers, and researchers in learning processes. For this the system itself or additional tools have to incorporate computational techniques for analysis of the collaborative process to enable a better understanding of it. Some of these techniques consider the linguistic aspects of contributions, while others focus on activities on the domain level and the results of collaboration. The first line of research used either techniques for understanding of textual contributions [1] or classification of contributions on pragmatic level with semi-structured interfaces, such as sentence-openeners [2]. The second line investigated in analysis of domain-related actions in shared workspaces like analysis of the resulting states of collaboration [3] by checking structural properties. In CSCL-scenarios like classroom learning typically we cannot assume the system to be aware of the complete interaction, so that we tend to rely more on an understanding of actions and states on the domain-level of shared workspaces [4], taking also into account coordination acts, if available.

Learning Protocols are available in most CSCL-systems by logging the activities taking place in the collaborative environment. Logfiles are used in various ways [5,6] related to the AIED area. They have been used for manual inspection and interpretation [7], for exploration of mis-use of tutor support [8] and even for construction of tutors

[1]Correspondence to: Andreas Harrer, Abteilung für Informatik und angewandte Kognitionswissenschaft, Lotharstr. 63/65, 47057 Duisburg, Germany. Tel.: +49 203 379-3554; E-mail: harrer@collide.info

from real data [9]. In distributed architectures with a communication server these logfiles are produced automatically by logging the messages sent from learners' machines. Initially these logfiles are at a very low abstraction level and of little use for teachers, learners, and even the system to create an understanding of the ongoing collaboration.

In our collaborative applications the MatchMaker [10] communication server creates object-based logfiles, i.e. actions are related to the objects they operate on (e.g. creation, deletion, or modification of an object). These logfiles contain information about the user conducting the action, the original creator of the object and other aspects, that may be analysed for information about the ongoing collaboration. Even though this format is not as low-level as pure user-interface-events, the logfile itself is not useful for learners and teachers in its raw form. Our goal is to help users of CSCL systems to better understand the learning processes and to explore research questions. This is supported by providing a tool that enables the user to analyse logfiles from different CSCL systems.

2. Discovery of Patterns in Logfiles

Specific phases of collaboration or situations in the learning process, such as turn taking between learners or communication breakdowns, are indicated by typical sequences of user actions. The automatic discovery and visualization of these sequences helps teachers evaluating the process, researchers investigating their research questions, and students reflecting on their own activities. For that end, sequences have to be searched for matching a specific *pattern*; we will use the term pattern in this sense for a typical sequence of user actions. Depending on the type of educational application, logfiles of collaborative learning sessions contain usually an interspersed sequence of actions on coordination level, like chat messages, and domain-related actions, such as creating a UML-Class. This means that search for cohesive patterns (i.e. all actions directly succeeding each other) might not be sufficient: some interferences of actions, that are not relevant for the user, could be interspersed, so that a pattern can't be found in direct sequence. Thus the issue how to find patterns that are not strictly cohesive in the logfile has to be addressed in a tool supporting the user in logfile analysis. Another topic to consider when designing a tool for logfile analysis is that the researcher often has some hypothesis but cannot clearly specify how this hypothesis manifests itself on the low level of logfiles. For this it is desirable to automatically compute frequently occuring patterns that have not been specified before as a query. In the next subsections we will present our approach to tackle both the problems of dispersion of patterns and the extraction of unspecified patterns.

2.1. Explicit Specification of Patterns

Specified patterns can be found using standard algorithms or query engines if the patterns are cohesive in the logfile. Otherwise partial instances of patterns can be continued later in the logfiles. Because of this computational complexity we allow to filter out activities that are not relevant - if interspersed in a specified pattern - by choosing for each action type in the logfile if it is considered. The same can be done for specific users, e.g. when all actions of the teacher are filtered out to analyse just the learners' actions. By selecting actions in a suitable way incohesive patterns can be made cohesive for analysis purposes in the filtered logfile. To enable the user of our analysis approach to conveniently specify the patterns to be searched for, we provide three different methods for specification:

Rule specification by action types and rule composition: Based on the logfile to be analysed all action types (combination of objects and respective actions) that occur in the logfile are presented in a list. Thus actions, that do not occur in the logfile will not be part of the pattern to be searched. The user can specify a rule by selecting stepwise one or more of the action types in the order expected to occur in the log (see figure 1).

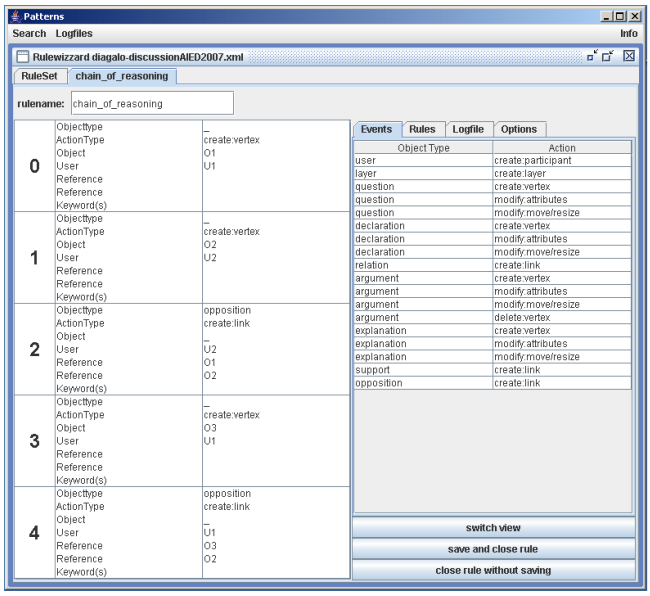

Figure 1. Rule specification by action types

Dependencies between the elements of a rule are specified by identical variables in the respective slot, e.g. in the User slot `U1`. The user can define rule sets and use already defined rules as subcomponents of new rules, to define complex rules by composition.

Specification by Example: The user also has the option to define a rule in a more convenient and informal way by selecting directly actions from the logfile as the pattern that is searched for. The tool automatically produces a rule by abstracting from the concrete values of the different slots of actions, but keeping the dependencies using the same variables if the values of slots were identical. Thus it is possible to review the logfile and use the system without any knowledge required about programming or rule specification.

Direct editing of rules: For the expert user there is also the option to directly edit and modify the rules (either created by rule specification or by example) at the level of the query language (cf. implementation section). This allows the user to add constraints or to the rule manually, but requires much more expertise than the previous methods, that hide the implementation level from the user who can define his rules on the action level.

Regardless of the method chosen for specification of rules the rules are compiled into rule sets that can be stored for later re-use. Thus a researcher loads a previously defined rule set, which compiles his standard rules for logfile analysis, chooses the rules from the set that should be actually used, and then conducts the pattern search (maybe after adding some specific rules for the current logfile/research question at hand).

2.2. Finding Unspecified Patterns

The much more challenging task of providing typical patterns that have not been specified before, demands a different algorithmic solution. For this we chose the so called "sawtooth" algorithm [11] that scans a string for the longest re-occuring substring starting from each position. This $O(n^2)$ algorithm iterates through a logfile and checks for each position in the log the maximal length of re-occurring sequences in the rest of the log. A completely un-supervised run will produce a lot of potential patterns of arbitrary length and frequency (minimum 2 to produce a re-occurrence at all); additionally this algorithm considers only directly succeeding patterns. To produce better results we provide several options for a user-guided pattern mining, i.e. the user defines general properties the patterns searched for have to comply to. These options are:

Objects to be considered: In a list the user can see all the objects used in the learning process captured in the logfile. For each item the user can inspect all the actions that have been conducted with this specific object. Based on this information she can choose if this object should be considered for the following pattern discovery run.

Object context: If the CSCL system producing the logfile represents relations between objects, as graphical modelling tools (e.g. Belvedere, CoLab, CoolModes, Digalo, ModellingSpace) do, an object's context is defined by its related objects. By configuring the maximal relational distance the size of the context can be varied; e.g. objects connected via a maximum path distance of 2 to the object can be selected for the search. This feature allows to detect patterns in specific regions of the learning products.

Users to be considered: Similar to the objects, a list of participants in the learning process is shown to the user of the pattern search tool. Choosing one participant in the list produces a list of actions conducted in the learning process by the participant. By reviewing these the user can decide which users should be considered for the search.

Action types to be considered: This globally allows or disallows action types associated with a specific object type in the pattern discovery. E.g. the user can decide, that moving a UML-Class object is not relevant for her research question who typically worked on content-level, but changing attributes of a UML-Class object is relevant.

Pattern length and Frequency: These options constrain the number of patterns to the ones that comply to minimum lengths of pattern and minimum frequency of hits in the logfile. Only the compliant ones are proposed to the user as candidate patterns.

By combining several of these options the user guides the pattern discovery process in the direction of the patterns he is interested in. The pattern discovery produces rules, each characterizing one pattern schema fulfilling all the option's criteria, which can be used as if they have been specified by the user directly (cf. previous subsection).

3. Tool Implementation

This section describes our practical implementation of the previously explained concepts. It is developed as a stand-alone analysis tool that is flexibly usable by analysts of learning processes with diverse learning systems. Important issues are:

Generic logfile format: To support a wide variety of different learning systems we defined a format that uses standardized attributes required for our analysis process (e.g. action type, user, timestamp), yet on the other hand is able to preserve and use

tool-specific information (e.g. for a graph-based tool relations between objects) in the analysis. The chosen logfile format is a XML representation of events triggered by the learner, so that other XML-based logfiles can be converted conveniently to this format.

Prolog runtime system as query engine: The patterns to be searched for can be considered as rules, especially when composing more complex patterns based on existing ones; this maps well to using a rule within a composition rule. Instead of implementing a search engine from scratch we explored the option of using existing rule-based engines for logical programming. Because of our use of the Java language for the graphical user interface and the logfile parsing, JESS (Java Expert System Shell) and SWI-Prolog were the main options: both provide interoperability with Java applications and offer optimized search engines. In practical tests we experienced performance problems with JESS and thus chose SWI-Prolog in conjunction with the JPL Java Interface.

Conducting the pattern search is a two-step procedure: first for every event in the logfile a Prolog fact is defined according to the general format:

```
event(eventid, objectid, action, objecttype,
      timestamp, user, additionalAttributes)
```

A pattern is described internally as a sequence containing event structures with variable elements and (for composition rules) subrules. The elements of an event structure are defined schematically: *user-defined values* are used as concrete values in the rule, *user-defined variables* are used in the rule consistently for each occurrence of the variable, and *elements the user didn't specify* are used as the unspecified Prolog variable "_". When composing rules, the representation of the subrule is added to the rule definition.

Implementation of the Sawtooth-Algorithm Due to the generic input logfile we need a generally applicable pattern search algorithm without domain-dependent optimizations. Our implementation of the Sawtooth algorithm operates on a sequence of action events: a section of the logfile is matched against the logfile using the *patternlength* and *frequency* options. If the section satisfies the two options, it is marked as an occurrence of a pattern and is expanded with additional subsequent actions in the next iteration; otherwise a new section will be tested. At termination the algorithm has marked possible patterns and their occurrence. To generate a suitable assignment of parameter values that represents the dependencies between different actions (such as same user, same object) an incremental processing similar to the *apriori* algorithm is conducted.

Visualization of hits: Found patterns are displayed in a diagram illustrating the timeline of the learning session. Every point of the graph is an action event within a pattern and is connected with its previous and succeeding action events. Every point has a tooltip window containing all information of its action event. Additionally all hits are arranged in a tree-structure: the tree allows the user to see an abstract representaion of the rules belonging to specific patterns and the parameter values of specific hits. Selecting one rule respective one pattern highlights the corresponding subgraph in the diagram.

4. Usage as a Research Tool for Discussion Analysis in ARGUNAUT

The approach presented has been implemented within a tool described in [12] with early results of the feasibility of the proposal based on research data from a case study [9]. Since then it was used in empirical research in the EU-funded Argunaut project (IST contract no. 027728). The project's goal is to provide moderators with computer-based

tools to support and increase their effectiveness and thereby the quality of monitored e-discussions. ARGUNAUT aims at delivering a unified mechanism of awareness and feedback to support moderators in multiple e-discussion environments.

One of the specific aspects of the Argunaut project is that the moderation support is not constrained to one dedicated discussion environment; versatile environments, currently the Digalo, FreeStyler, and Cool Modes applications[1] developed by different partners, can make use of the analysis and moderation features of the system. Thus, while initially the Pattern Discovery Tool was developed for the collaborative workspace environments FreeStyler & Cool Modes, the provision of a standardised XML-based logging format enables the tool to analyse all logfiles compliant to this format.

The pattern discovery tool was used as a research tool to help moderators or researchers identify relevant discussion patterns that occured in real discussions with visual graph-based languages (see 2). After a first practical use of the tool with real data and feedback from the practitioners, we added the functionality to search for texts (e.g. keywords) in the discussion cards; this enables us to combine the process-oriented dimension of actions on graphical discussion objects with content-based textual aspects, such as characteristic phrases for agreement, disagreement, explanation etc. We allows to utilize the diverse background of the project partners in pedagogy, linguistics, argumentation and interaction analysis in a combined analysis.

The pedagogical experts in the project then used the option of explicitly specifying patterns of interest by action types. The complexity and abstraction level of rules varied widely between relatively directly observable phenomena, such as *agreement*, to quite abstract and complex concepts, such as *change of opinion*. A selection of these rules is:

Agreement: a sequence in which two discussion shapes are created by different users and a "support" link is added between them.

Chain of reasoning (in response to attack): a sequence in which a user creates a discussion shape, another user creates a shape linked to the first shape using an "opposition" link, and the first user responds to this opposition by creating a shape and linking it to the other user's shape with an "opposition" link. A specification of this pattern can be seen in figure 1, while figure 2 shows 2 hits of the pattern manifested in a Digalo map.

Possible change of opinion: a sequence in which different users create two discussion shapes and a link between them. After this the same users create two discussion shapes with a different link (e.g. opposition becomes support or vice versa).

Revision / deletion: a sequence in which a user first creates a discussion object and later this object is modified or deleted. Discussion objects created by a specific user may be modified/deleted by the same user who created them or by other users, and this distinction (expressed by different sub-rules) is meaningful to the moderator or researcher.

Reasoned opposition: a sequence in which two discussion shapes are created by different users and an "opposition" link is added between them. The second shape is specified to contain reasoning keywords, such as "because", "example", etc. (shape types may also be specified).

Question and explanation: a sequence in which a user creates a discussion shape, another user creates a discussion shape of the "question" type (and/or one containing questioning keywords) which is later linked to the first shape. In the following steps, the first user responds to this question by creating another shape of the "explanation" or

[1] `http://dunes.gr , http://www.collide.info/downloads`

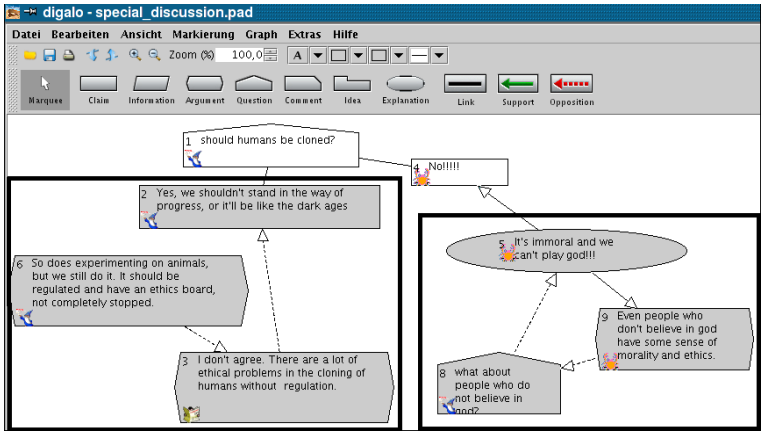

Figure 2. Digalo discussion map with two identifed hits of the chain of reasoning pattern

"reason" type (and/or a shape containing reasoning/explanation keywords) and linking it to the other user's shape.

The experts' approach in the ARGUNAUT project was to pre-define possible action sequences thought to be meaningful, i.e. representing specific theoretical or pedagogical phenomena (examples above) and to look for such sequences. The rules specified were applied to 582 XML log files of graphical discussion maps (created by small groups of school pupils with the Digalo environment). When applying a ruleset to a complete folder of logfiles, the results are shown in an ordered list containing the logfiles with most hits on top and then descreasing number of hits. This helps to give a first overview about the logfiles and the characteristics that are expressed by the numbers of each rule's hits. Typically a researcher would define her/ his own ruleset, apply it to the whole dataset and thus obtain a list of candidate XMLs (the discussions which show the largest number of hits) for deeper investigation at the fine-grained logfile level: then the temporal sequence, the actors involved in the hits can be inspected more closely.

The practical usage by the experts showed that they were capable of using the tool for specification of quite complex phenomena relevant to e-discussions. Yet, we also found out that the hits found by a discovery run sometimes were not considered as useful. This is due to over-generalization by the experts, which could have been avoided by applying the constraints and options offered in the tool. An example of this is that a rule for *repeated modification* of an object, that was specified with 3 modification actions, produced several hits for 4 modification events (all three-sets possible); this could have been avoided using options for time intervals or interspersed actions within the rule. To adress the issue of non-appropriate hits we added a feature to annotate the hits according to whether they truly represent the semantic interpretation of the expert ('good hits'), and save this information in XML format for further reference. Another observation from this experimentation is, that the experts frequently tried to define phenomena with structural aspects (e.g. shapes linked to 3 other objects) without the temporal dimension the tool allows for. Thus, while from these experiments of usage we can conclude that the tool is usable and useful, there is also a challenge in operationalizing the experts' intentions as rules. We assume that this requires close collaboration between the experts in the domain with people of formal background to clearly specify these rules.

5. Conclusion and Outlook

This paper presented our approach to support analysts of learning processes, i.e. researchers and teachers, in understanding the processes captured in computer-created logfiles. We provided functionality to find typical patterns of learning actions that can be either specified by the user via a convenient graphical interface or that are discovered automatically with configuration options for the user. We also adressed the issue, that especially in logfiles of collaborative processes, patterns might be interspersed with actions that are not so relevant for the analyst, but which inhibit the finding of meaningful patterns. This concept has been implemented within a tool and put to practical use in a European funded project with logfiles created by usage of a graphical discussion environment of one of our project partners. The analyis tool is designed to accept not only these logfiles, but provides a generic logfile format and suitable XSL-T transformation scripts to map the logfiles of arbitrary learning systems to the generic format. We will use our tool for analysis of lab and classroom experiments previously done manually [9].

Acknowledgements Our thanks go to Michael Vetter, Jens Brauckmann, Stefan Thür for their work on the pattern tool and our Argunaut partners for their comments.

References

[1] Carolyn P. Rose, Andy Gaydos, Brian S. Hall, Antonio Roque, and Kurt VanLehn. Overcoming the knowledge engineering bottleneck for understanding student language input. In *Proc. of AIED 2003*, pages 315–322, Amsterdam, 2003. IOS Press.

[2] Michael Baker and Kristine Lund. Flexibly structuring the interaction in a CSCL environment. In *Proc. of EuroAIED'96*, Lissabon, 1996.

[3] Maria de los Angeles Constantino-Gonzalez and Daniel Suthers. A coaches collaborative learning environment for entity-relationship modelling. In *Proc. of ITS 2000*, pages 325–333. Springer, 2000.

[4] Katrin Gaßner, Marc Jansen, Andreas Harrer, Kai Herrmann, and Ulrich Hoppe. Analysis methods for collaborative models and activities. In *Proc. of CSCL 2003*, pages 369–377. Kluwer, 2003.

[5] Cecily Heiner, Ryan Baker and Kalina Yacef (eds.). *Proc. of ITS2006 Workshop on Educational Data Mining*, Jhongli, Taiwan, 2006.

[6] Joseph E. Beck, Esma Aimeur, and Tiffany Barnes (eds.). *Educational Data Mining: Papers from the 2006 AAAI Workshop, Technical Report WS-06-05.* Menlo Park, California, 2006.

[7] Erica De Vries, Kristine Lund, and Michael J. Baker. Computer-mediated epistemic dialogue: Explanation and argumentation as vehicles for understanding scientific notions. *The Journal of Learning Sciences*, 11(1): 63–103, 2002.

[8] Ryan S. Baker, Albert T. Corbett, and Kenneth R. Koedinger. Detecting student misuse of intelligent tutoring systems. In *Proc. of ITS2004*, 2004.

[9] Andreas Harrer, Bruce M. McLaren, Erin Walker, Lars Bollen, and Jonathan Sewall. Creating cognitive tutors for collaborative learning: Steps toward realization. *User Modeling and User-Adapted Interaction: The Journal of Personalization Research*, 16: 175–209, 2006.

[10] Marc Jansen. Matchmaker - a framework to support collaborative java applications. In *Proc. of AIED 2003*, pages 529–530, Amsterdam, 2003. IOS Press.

[11] Heinz Ulrich Hoppe and Rolf Plötzner. Inductive knowledge acquisition for a unix coach. In *Informatics and Psychology Workshop*, pages 313–335, 1989.

[12] Andreas Harrer, Michael Vetter, Stefan Thür, and Jens Brauckmann. Discovery of patterns in learner actions. In *Proc. of AIED 2005*, pages 816 – 818, Amsterdam, 2005. IOS Press.