CASE: A Configurable Argumentation Support Engine

Oliver Scheuer and Bruce M. McLaren

Abstract—One of the main challenges in tapping the full potential of modern educational software is to devise mechanisms to automatically analyze and adaptively support students' problem solving and learning. A number of such approaches have been developed to teach argumentation skills in domains as diverse as science, the Law, and ethics. Yet, imbuing educational software with effective intelligent tutoring functions requires considerable time and effort. We present a highly configurable software framework, CASE ("Configurable Argumentation Support Engine"), designed to reduce effort and development costs considerably when building tutorial agents for graphical argumentation learning systems. CASE detects pedagogically relevant patterns in argument diagrams and provides feedback and hints in response. A wide variety of patterns are supported, including ones sensitive to students' understanding of the domain, problem-solving processes, and collaboration processes. Teachers and researchers can configure the behavior of tutorial agents on three levels: patterns, tutorial actions, and tutorial strategies. The paper discusses design concerns, the architecture, and the configuration mechanisms of CASE. As a proof of concept, four showcases are presented each showing different aspects of CASE and thus demonstrating the flexibility and breadth of applicability of the CASE approach in supporting single-user and collaborative scenarios across different argumentation domains.

Index Terms—N.3.II Collaborative Learning Tools, N.4.I Intelligent Tutoring Systems, N.5.V Authoring tools

1 Introduction

Argumentation skills are vitally important in many respects but their teaching is not well established in our educational system [1]. One method employed in teaching and learning well-founded argumentation skills is *argument diagramming* [2]. Argument diagrams are based on a decomposition of arguments into their constituent elements (e.g., claims, statement, evidence) and relations (e.g., a claim is supported / opposed by a statement, a piece of evidence provides backing for a statement), represented in the form of node-and-link graphs. Students can acquire argumentation skills by creating or inspecting such diagrams, individually as well as in groups. The argument-diagramming paradigm has been adopted in a wide range of computer-based argumentation systems, in domains as diverse as the Law, science, and ethics [3], [4]. The computerization of the process brings important benefits over paper-and-pencil approaches such as: easy modification and revision of diagrams; adaptable visualizations, orientation, and navigation support (e.g., resizable display, overview maps, search functions); digitalization and persistent storage of created diagrams; remote collaboration and sharing of diagrams; and automated, system-generated support for students and teachers to guide the process and evaluate the result, which is the focus of this paper.

We present a software component, called *CASE* ("Configurable Argumentation Support Engine"), which supports the definition of tutorial agents to be deployed to argumentation systems. The tutorial agents analyze student activities and generated artifacts and provide hints and feedback in support of argumentation learning activities. CASE has been designed for usage in a wide variety of argumentation domains and learning scenarios. Therefore, a special focus has been placed on flexible configuration mechanisms that allow support to be tailored to specific conditions and pedagogical approaches. CASE has the potential to considerably reduce efforts in the development of adaptive support mechanisms by providing a reusable and easily extended framework. CASE works in tandem with the *LASAD* argumentation system, which itself is highly configurable across domains and settings [5].

- *O. Scheuer is with the Center for e-Learning Technology, Saarland University, Campus, 66123 Saarbrücken, Germany. E-mail: o.scheuer@mx.uni-saarland.de.*
- *B.M. McLaren is with Carnegie Mellon University, Human-Computer Interaction Institute, 2617 Newell-Simon Hall, 5000 Forbes Avenue, Pittsburgh, PA, 15213-3891, and also with the Center for e-Learning Technology, Saarland University, Campus, 66123 Saarbrücken, Germany. E-mail: bmclaren@cs.cmu.edu.*

2 Background

Although CASE can be used to support individual student learning activities, its real focus is on analyzing and supporting collaborative learning arrangements. An early comprehensive overview of Computer-Supported Collaborative Learning (CSCL) systems is provided in [6]. Systems are classified into one of three categories depending on the "locus of processing" (student / teacher versus system): *Mirroring tools* support students or teachers by collecting, aggregating, and presenting interaction data faithfully, e.g., in a visual display, yet without hinting at how a good or ideal mode of collaboration would look like. The mirrored data aims at raising students' or teachers' awareness; interpretation and use of the data is left to the student or teacher. *Metacognitive tools* provide, in addition, a normative model of ideal or desired collaboration. The model serves as a point of reference for interpreting and assessing the quality of interactions. Yet, the

diagnostic task itself remains under the control of students and teachers rather than the system. Finally, *guiding systems* also diagnose collaboration problems and suggest remedial actions. That is, the locus of processing is shifted in large parts from the users to the system. CASE is aimed at supporting precisely such guiding systems.

Under the rubric of *Adaptive and Intelligent Systems for Collaborative Learning (AICLS)*, a recent review of such guiding systems for CSCL is given in [7]. According to their scheme, systems can be categorized (among other dimensions) according to the target of intervention (group formation, domain-specific support, peer-interaction support), modeled aspects (user/group, domain, activity), and modeling techniques (ranging from AI techniques, such as Bayesian Networks, to Non-AI techniques, such as user-defined preferences). CASE supports the development of systems that provide domain-specific and peer-interaction support, based on domain and user activity models, realized through rule-based pattern matching techniques. If needed, external analysis modules of any kind (e.g., machine learned classifiers) can be integrated with CASE through a well-defined extension API.

Automated analysis and feedback techniques to support argumentation learning are reviewed in [4]. Following [8], a distinction is made between *argument modeling systems*, which support the analysis and structural representation of arguments, and *discussion-oriented systems*, which provide a medium for argumentative exchange between discussants. While discussion-oriented systems often aim at a broad set of communication and collaboration skills, such as balanced participation, topic focus, and leadership, argument modeling systems focus on the logic of arguments and domain-specific argument structures. Accordingly, the two system classes employ different analysis approaches.

The following analysis approaches are used in argument modeling systems: *(1) Syntactic analyses* check whether the created argument representation complies with a set of given syntactic constraints (e.g., data supports hypotheses and not vice versa). *(2) Problem-specific analyses* check whether the created argument representation adequately models a given problem case (e.g., a transcript of an existing argument). *(3) Simulations of reasoning / decision-making processes* determine whether a claim is believable / acceptable based on the created argument representation. *(4) Assessments of content quality* determine the quality of the textual content of individual argument components. *(5) Classifications of the current modeling phase* determine whether the student is, for instance, in an orientation, modeling, or reflection phase (i.e., problem solving is conceived of as a multi-phase process).

The following analysis approaches are used in discussion-oriented systems: *(1) Analyses of process characteristics* identify the function of discussion moves and speaker intentions, for instance, counterarguments and question-answer interactions in dialogues. *(2) Analyses of discussion topics* identify the current topic of a discussion. *(3) Analyses of interaction problems* identify, for instance, unanswered questions and failed attempts to share knowledge. *(4) Assessments of collaboration quality of longer sequences of time* aggregate and summarize students' behaviors over time, for instance, the level of group responsiveness and agreement. *(5) Classifications of the current discussion phase* determine whether the group is, for instance, in a confrontation, opening, argumentation, or conclusion phase (i.e., a discussion is conceived of as a process that unfolds into multiple phases).

Support mechanisms in these systems are classified according to the following dimensions: *(1) feedback mode* (e.g., text, highlighting, meters), *(2) feedback content* (e.g., self-reflection prompts versus explicit directives), *(3) feedback control* (student-driven, moderator-driven, system-driven), *(4) feedback timing* (on-demand, immediate, summative), and *(5) feedback selection and priority* (e.g., select / prefer messages that refer to recent events).

CASE can principally support all of the previous mentioned analysis approaches, either through CASE's rule-based pattern matching mechanism or through connected external analysis modules. Examples are discussed in section 4. With respect to support mechanisms, CASE allows the configuration of textual messages and highlighting of diagram elements. Feedback is provided on-request; a configuration option for proactive, system-triggered feedback is currently under development. The configuration of feedback selection and prioritization strategies is supported as well.

3 LASAD Argumentation System

CASE has been developed in context of the LASAD project ("Learning to Argue – Generalized Support Across Domains"), which aims at developing a software framework and methodology to build argumentation-learning systems for a range of domains. Most past argumentation systems have been designed with specific domains and learning scenarios in mind, resulting in systems that cannot be ported to different application settings. Yet, on a conceptual level, these systems share many features in terms of the user interface and underlying functionality. In principle, it should be possible to develop a more general framework that can be used as a basis for building specific argumentation systems in a more simplified fashion, based on well-defined configuration and extension mechanisms. Within the LASAD project, this is precisely our objective and what has been developed. The generality of LASAD has been shown through its use in a wide variety of differently targeted argumentation-learning applications and empirical studies (e.g., [9], [10]).

The LASAD system [5] is based on the argument-diagramming paradigm, an approach that has gained considerable popularity during the last two decades for reasons described in the introductory section. Fig. 1 shows a screenshot of the user interface. In this instance, an Intelligent Tutoring System for legal argumentation, LARGO [11], has been implemented using the LASAD framework. In LARGO, students analyze a given transcript of a U. S. Supreme Court oral argument (Fig. 1, left panel) by creating a diagrammatic representation of it (Fig. 1, right panel). When students are stuck they can

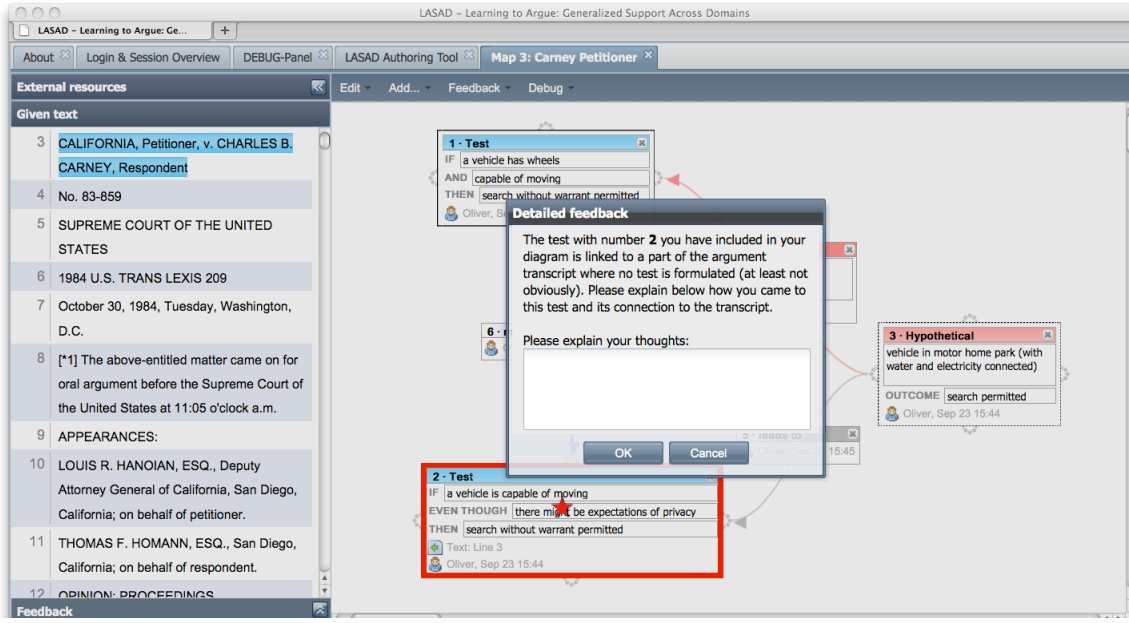

Fig. 1: LASAD user interface

request hints, which will be provided in the form of a text message and highlighting of the portion of the diagram the message refers to (Fig. 1, message window on top of diagramming area). LARGO will be discussed in greater detail in section 4.1.

Many aspects of the LASAD user interface can be configured through XML or an authoring tool, which facilitates the process considerably, especially for novice users. The type and makeup of boxes and links can be set up. For instance, the example in Fig. 1 uses "Test" and "Hypothetical" box types. "Test" boxes comprise a number of text fields such as "IF," "AND," "EVEN THOUGH," and "THEN," some of which are predefined, while others can be added and removed dynamically by the user. Besides text fields, other widget types can be used, such as dropdown menus, rating elements, and radio buttons. The diagramming area can be enhanced with other tools and displays. In Fig. 1, a transcript panel has been added. Other options include displaying the list of active users, adding a chat tool, possibly enhanced with "sentence openers" [12], or adding tutorial agents that support students while creating diagrams.

4 CASE Applications

To illustrate the generality and breadth of applicability of the CASE framework, the main objective and driving force in the design of the system, we now discuss four CASE applications (*LARGO*, *Science-Intro*, *Metafora*, and *ARGUNAUT*). These applications support argumentation-learning activities in different domains (the Law, science, group deliberation, and ethical discussion), focus on different argumentation facets (analysis, planning, and discourse), and use different features of the CASE framework (structural patterns, process-based patterns, and integration of external analysis modules).

In *LARGO*, students analyze and structurally represent legal argumentation processes using argument diagrams. In *Science-Intro*, students use diagrams as an outlining tool to prepare the writing of research reports in the domain of psychology. Both applications are primarily designed for single-user activities. Adaptive support is provided on request and based on structural patterns defined by domain experts. In *Metafora*, students jointly work in an inquiry environment for mathematics and science. They use LASAD diagrams to discuss, in a structured way, findings obtained in microworld simulations, with the aim of arriving at a joint, agreed solution. In contrast to LARGO and Science-Intro, in which the CASE framework is used to detect domain-specific structures in diagrams, the focus is on interaction patterns to support students in "learning to learn together." *ARGUNAUT* also focuses on interaction patterns but uses a different analytical approach. Rather than relying on expert-defined patterns, machine-learned classifiers are utilized to categorize qualitative aspects of e-Discussions about controversial ethical dilemmas.

4.1 Legal Argumentation: LARGO

The Intelligent Tutoring System *LARGO* [11] was developed to teach beginning law students a particular model of legal argumentation based on hypothetical reasoning [13]. The model semi-formally describes argumentative processes as they can be observed in U. S. Supreme Court oral trials. According to this model, lawyers propose some test (i.e., if-then rules) how to decide certain legal situations and argue that this test also applies to the case under discussion. Proposed tests are based on an interpretation of legal statutes and precedent cases and are chosen in a way that leads to a favorable outcome for the proposing party. To challenge a proposed test, the opposing party may cite hypothetical situations that put the validity of the test into question (i.e., the test would lead to some undesirable result). The first party might respond to such challenges by withdrawing the proposed test or modifying it in some reasonable way. Typical moves in-

clude analogizing or distinguishing between the current facts and hypothetical situations.

To practice this model of legal argumentation students are tasked with analyzing a given transcript of a U. S. Supreme Court oral argument within the LARGO system. They "translate" the given textual argument representation into an argument diagram based on the model of hypothetical argument described above. The argument ontology reifies important concepts of that model using "Facts," "Test," and "Hypothetical" boxes and "leads-to," "modified-to," "analogized-with," and "distinguished-from" links. While modeling arguments in LARGO, students can use a "Hint" button in the user interface. The system is capable of identifying more than 40 different patterns in the argument diagrams, which are used as a basis for hint generation.

The current LARGO version has been re-implemented based on the LASAD framework to be deployable over the web and to benefit from other LASAD assets (look-and-feel, maintainability, configurability). The LARGO help system, including all analysis rules, has been ported to the CASE framework. A screenshot is shown in Fig. 1. The following three patterns illustrate the kind of patterns used in LASAD:

(1) a "Test" node with some content in the "if" text field but none in the "then" text field: That is, the test has not been completely specified. If an instance of this pattern has been detected a feedback message can be triggered that prompts the student to enter some text into the "then" text field.

(2) a "Hypothetical" node that is distinguished from or analogized with a "Facts" node, but is not related to any "Test" node: Since hypotheticals are typically used to challenge proposed tests, the structure is incomplete, so a student could be prompted to connect the "Hypothetical" node to some "Test" node.

(3) a circular structure of nodes, in which each node "leads-to" or is "modified-to" the next node: The semantics of a "leads-to" or "modified-to" transition often involve a temporal progression, which is counteracted by the pattern's circularity. However, if interpreted as logical consequence, a circular structure can make sense. This pattern can be used to prompt students to rethink their diagram model (temporal or logical relation?) to identify possible mistakes.

Other rules not discussed here make use of expert annotations of the given transcripts, which mark passages in the transcript as "test," "facts," or "hypothetical." Since students create explicit references from diagram elements to transcript passages (through a specific GUI widget), it is possible to check whether they have misclassified certain passages (e.g., a student creates a "Test" box to model a transcript passage annotated as "Hypothetical").

4.2 Scientific Argumentation: Science-Intro

The *ArgumentPeer* project ("Teaching Writing and Argumentation with AI-Supported Diagramming and Peer Review") [14] aims at developing an Intelligent Tutoring System to teach students how to write argumentative texts. One component of the system is the LASAD diagramming environment, which students use to outline arguments in a diagram in advance, as preparation for the actual writing of the text.

Besides the legal domain, the project tackles the writing of scientific arguments in psychology. The students' task is to write a report, which motivates and defines a new research study based on a review of relevant literature and reports on the study results. The text should indicate hypotheses and claims that the current study is based upon and cite previous literature to either support or oppose those claims and hypotheses. The current study should be compared to previous studies to point out analogies and distinctions. Citations that lead to contradictory results (e.g., citation x supports a claim while citation y opposes the same claim) should be compared to one another in terms of similarities and differences.

To prepare the writing of such research reports students use LASAD to outline the basic structure of the text. Equipped with a selection of relevant background literature provided as part of the assignment, students create a LASAD diagram using an ontology specifically designed to support the outlining of scientific arguments, including boxes such as "current-study," "claim," "hypothesis," and "citation" and links such as "support," "opposition," and "comparison." Similar to LARGO, a "Hint" button is offered in the user interface.

The ArgumentPeer project includes a help system used to identify patterns in student activity and provide feedback accordingly. Since the ArgumentPeer project only recently started, the help system is still under development. The three patterns below have been defined and implemented within the CASE framework in a first design iteration, along with other preliminary patterns. The project team is currently investigating a first version of the help system in pilot tests.

(1) a node of arbitrary type with an empty text field: This pattern is a generalization of the first LARGO pattern discussed in section 4.1. It can be used to check whether students filled in every text field in their argument diagrams and, if not, prompt them to do so.

(2) a "Hypothesis" node with fewer supporting than opposing inbound links: In general, it is good if students also consider evidence that contradicts a hypothesis rather than only searching for confirmatory evidence, a well-documented psychological phenomenon ("confirmation bias"). However, sometimes students also neglect supporting evidence, or might neither pay attention to supporting nor to opposing evidence. This pattern identifies such situations in order to prompt students to search for positive evidence. Confirmation bias, i.e., neglecting contradictory evidence, could be detected analogously.

(3) unconnected node clusters in the diagram ("argumentation islands"): Since the goal of the outlined argumentative text should be to present a current research study in a coherent way, all discussed hypotheses, citations, claims, etc. should be related to one another in some way, if not directly, then at least indirectly through some other elements. In the end, all components must integrate smoothly into one coherent line of argumentation in support of the overall aim of the research report.

This pattern is also potentially useful in other applications that require a well-integrated representation of knowledge, such as pre-writing activities in other domains, or group deliberation and discussion, which will be discussed next.

4.3 Group Deliberation and Discussion: Metafora

The Metafora project aims at developing a pedagogy and technical platform to support students in "learning to learn together" (L2L2; [15]). In the context of Metafora, complex, challenge-based learning scenarios, involving a variety of learning tools and stretching over a timespan of two to three weeks, have been researched and developed. The project tackles the domains of science and mathematics and targets students between ages twelve and 16. While the effective teaching of conceptual domain knowledge is certainly one goal of the project, the emphasis is put on L2L2 skills, such as distributed leadership, mutual engagement, help seeking and giving, and reflection of group learning processes. A Metafora challenge is defined in terms of an ill-defined problem that requires students to gain a mutual understanding of the given challenge, jointly plan and coordinate activities to address this challenge, explore ideas in simulation environments, discuss findings, and draw conclusions. Student activities are mediated through the Metafora platform, which integrates and provides access to a number of tools, such as a graphical planning tool to jointly outline and monitor learning activities, microworld simulations to experiment with mathematical and scientific models, and discussion tools, including LASAD, to coordinate collaboration efforts and reflect on finding.

Through LASAD, students can share and discuss models they have individually created in a microworld. Students "publish" their individual models to the group through a "My-Microworld" box, which displays a thumbnail image from the created model and grants fellow students access to this model in the original tool through a hyperlink. Students can then compare and discuss their models to decide whether they are "correct" or helpful to solve the given challenge, whether two models are equivalent to one another or not, or in what respects two models differ from one another. LASAD also serves as a forum for help seeking and giving. When students are facing problems in a microworld, they can seek help through a "Help Request" box, which displays a thumbnail image of the model the student is struggling with, together with a hyperlink to visit the model in the original tool. Help requests can be answered through a "Microworld Action" box, in which the help-giver can suggest, via two dropdown menus, specific microworld actions to address the problem.

In Metafora, CASE has been repurposed as a middleware to integrate LASAD with the Metafora platform. A CASE agent has been developed that exchanges messages between LASAD and Metafora. One of the next steps will be to use CASE for its original purpose, which is identifying patterns in LASAD diagramming in the context of Metafora.

The following three Metafora patterns are identified to support important aspects of student collaboration. They have been defined in CASE but not yet fed into the Metafora platform, since the concrete design of remedial tutorial actions and strategies is still under discussion:

(1) a "Help-Request" box not older than 10 minutes, unattended for more than 3 minutes (3 minutes passed by and no other box has been connected to the help request): The patterns indicates a situation, in which a student requests help regarding a problem encountered in a microworld. Yet, the help request went unnoticed, or is deliberately ignored, since three minutes have passed without a response. Because the help request is still recent – it has been published within the last ten minutes – it might be worthwhile to draw the attention of other students to this request in order to elicit help.

(2) one student has not contributed at all (no boxes) while the rest of the group has already contributed considerably (at least five boxes each other student): This is a heuristic approximation to one of the central problems in collaborative learning, imbalanced participation among group members. There are several possible reasons for a lack of participation [16]: Some students may know that others, who are interested in a good group result, will compensate for their lack of participation *(free-rider effect*; [17]). A possible consequence is that these others, who were mainly responsible for the group progress up to this point, are becoming increasingly upset and reduce or stop their participation as well (*sucker effect*; [17]). Overall, productive activities in the group may come to a halt. Also, low performers may refrain from participation because they feel less competent and thus miss opportunities to practice and develop, a vicious cycle (*Matthew effect*). The pattern could be further extended to focus on a recent time window.

(3) a student did not interrelate any of her own contributions with those of fellow students, even though she had an opportunity to do so (own and others' contributions exist in the diagram): This pattern suggests a lack of mutual engagement and transactivity (i.e., reasoning on the reasoning of others [18]), which are important prerequisites for collaborative meaning making. An appropriate tutorial intervention may be to prompt students to check how their contributions (e.g., their microworld model) relate to contributions of others (e.g., microworld model of fellow students).

4.4 Ethical Argumentation: ARGUNAUT

In the EU-funded *ARGUNAUT* project, an e-Moderation environment for graphical e-Discussions was developed [19]. The project tackled classroom-based scenarios in which one teacher-moderator monitors and supports multiple computer-mediated discussions (e.g., six groups with four or five students each). In graphical discussions students create and interrelate contributions (boxes) in a shared workspace. Typically, the teacher prepares the workspace with an initial box that presents the discussion topic, usually in the form of some controversial or ethical question (e.g., "Do you think it is ethical to perform experiments on animals?"). Students choose, depending on the kind of discussion move they want to make, a specific box

type, such as "claim," "argument," "question," or "explanation" and enter some text into this box. They connect boxes through graphical links to indicate that contributions "relate-to," "support," or "oppose" one another. Several LASAD-like graphical discussion tools are supported, including *Digalo* [20] and *FreeStyler* [21].

The process of monitoring and supporting multiple synchronous discussions in parallel is inherently difficult. The teacher-moderator must track and maintain a mental model of multiple discussion threads at a time. Important events in different discussions may occur in rapid succession, sometimes even in parallel. While monitoring or supporting one discussion thread important events in other discussions might pass unnoticed. The ARGUNAUT project investigated how the moderation process could be facilitated by means of a computer-based "Moderator's Interface," which provides awareness indicators and alarms to highlight noteworthy situations, and feedback tools to intervene and remediate identified problem.

From a computational perspective, two different kinds of awareness indicators are provided in ARGUNAUT. "Shallow" indicators are computed in relatively straightforward ways, e.g., through keyword search or descriptive statistics of box and link type usage. "Deep" indicators are more complex to compute but also potentially more meaningful to teachers. They are based on classifiers built using Artificial Intelligence techniques, in particular, machine learning, case-based reasoning, and natural language processing. The classifiers analyze structural, temporal, and textual data to categorize e-Discussion contents on three levels of granularity: single boxes, box pairs, and arbitrary clusters of boxes. Examples are discussed below.

Shallow and deep indicators are analyzed through different components of the ARGUNAUT architecture. The classifiers for deep indicators have been incorporated into an independent component (*Deep Loop*), accessible through a web service interface. As a proof-of-concept, the Deep Loop classifiers have been integrated into the CASE framework and can be applied in a LASAD setup that emulates the ARGUNAUT e-Discussion environment. Patterns in e-Discussions that can be identified by means of Deep Loop classifiers include:

(1) off-topic contributions: A few off-topic contributions may be acceptable but if an e-Discussion goes astray, with a considerable number of contributions not addressing the topic at hand, a human or artificial moderator might want to intervene.

(2) question-answer pairs: Questions serve important functions in discussions. For instance, they can challenge ideas, disclosing weaknesses and misconceptions of others, thus triggering processes of reflection and conceptual change. Answering questions is important as well, not only to satisfy a specific information need but also to maintain a healthy working atmosphere in the group. In sum, a fair amount of question-answer pairs indicates well-functioning group dynamics.

(3) chains-of-opposition: The pattern consists of an initial claim, followed by an objection, followed by a rebuttal of the objection, followed by a rebuttal of the rebuttal, and so forth. The sequence has at least a length of three. A chain-of-opposition indicates that student "negotiate" the meaning of ideas and arguments. It "deepens the space of debate" [22] through successive scrutiny and refinement of proposed ideas and arguments, thus representing exactly the kind of interaction valued in collaborative argumentation. Similar to "question-answer," "chain-of-opposition" is an indicator of good discussion quality.

Machine-learned classifiers that have been induced from a corpus of annotated examples can detect the first two patterns. The third pattern can be detected by means of a novel case-based graph matching technique, developed within the ARGUNAUT project, which searches clusters that are similar to prototypical examples and ranks these clusters according to their similarity scores. Other patterns not discussed here include "building-on" and "new-perspective," which is related to creative reasoning [23].

5. CASE System Architecture

In this section, we discuss the CASE system architecture, in particular, software design concerns (section 5.2), components and their interactions (section 5.3), and knowledge representation and inference processes within CASE (section 5.4). We start with the overall LASAD architecture, which CASE is one component of (section 5.1).

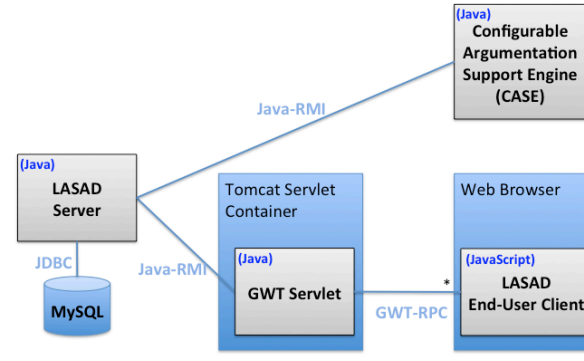

Fig. 2. LASAD architecture.

5.1 Overall LASAD Architecture

Fig. 2 shows the overall LASAD architecture. The *LASAD-Server* uses a database to maintain the history and state of all LASAD sessions (a session essentially corresponds to a LASAD diagram) and distributes messages between connected LASAD clients (of which two sorts exist: *End-User-Clients* and *CASE*) in order to synchronize their states. For instance, when a user creates a new box, the user interfaces of other connected *End-User-Clients* must be updated as well. *End-User-Clients* are JavaScript-based applications, built with the Google Web Toolkit (GWT) and executed in a standard web browser. The *LASAD-Server* cannot directly speak to the JavaScript-based client-side; an intermediary *GWT-Servlet* mediates the communication between *LASAD-Server* and *End-User-Clients*. *CASE* is just another client of the *LASAD-Server* that uses the same infrastructure and interfaces as *End-User-Clients*. Yet, since *CASE* is implemented in Java ra-

ther than JavaScript, it can directly talk to the *LASAD-Server* without the indirection of the *GWT-Servlet*.

To illustrate how LASAD operates let's have a look at a processing iteration, starting with a user creating a box in the LASAD user interface. The *End-User-Client* then sends a "create box" message though the *GWT-Servlet* to the *LASAD-Server*. The *LASAD-Server* updates its database and forwards the message to all connected clients. *End-User-Clients* display the new box on the screen; *CASE* updates its internal data representation, searches for meaningful patterns (examples are discussed in section 4), and possibly generates feedback messages. These feedback messages are sent through *LASAD-Server* and *GWT-Servlet* to one or multiple *End-User-Clients*, depending on the specific *CASE* configuration, to be displayed on the screen. While *CASE* provides the textual content of feedback messages and control flags, e.g., to specify whether diagram elements should be visually highlighted or not, the actual realization of the feedback presentation is done by the *End-User-Clients*.

5.2 Software Design Concerns

We turn now to the internals of CASE. The design of CASE addresses a number of key software system attributes, in particular, portability, maintainability, extensibility, efficiency, stability, availability of service, and configurability. In the following we describe how these concerns have been realized in the CASE architecture.

Portability

CASE is implemented in Java, which has been specifically designed for portability. Java binary code is executed in Java Virtual Machines (JVM), which encapsulate dependencies to the operating system, making platform specifics largely transparent to the application program. JVMs are available for virtually any platform, so CASE can be easily deployed in nearly any IT environment. A recompilation of binary code for the specific platform is not needed.

Maintainability and Extensibility

CASE has been designed in a modular fashion. The system unfolds into several components, each having well-defined responsibilities and interfaces (for details, see section 5.3). A loose coupling is achieved through event-based communication. Commands and status updates are encapsulated in message objects that are exchanged between components and processed asynchronously. The modular design makes it easy to adapt the system to a different environment. For instance, the communication to the *LASAD-Server* is encapsulated in a *DataService* component. If the communication protocol or message format changes, only the *DataService* component will be affected. The modular design also contributes to the attainment of an open architecture that can be easily extended with new analysis and feedback functionality. New functionality is encapsulated in *software agents*, which can be easily hooked up with the CASE framework. The CASE framework provides the basic infrastructure to handle generic tasks, such as the provision of events from and to agents, so developers can focus on the analysis and feedback logic when implementing new agents. An example that showcases the integration of already existing, external analysis functionalities is the ARGUNAUT Deep Loop, discussed in section 4.4. Deep Loop has been integrated into CASE using the *remote proxy design pattern* [24]: a *DeepLoopAgent* has been implemented to provide a CASE-compliant interface to the remote Deep Loop web service; the agent essentially serves as an adapter, making data formats and interfaces compliant with one another.

Efficiency

The most critical part in terms of runtime performance is the computational analysis of argument diagrams. CASE uses the *Jess Rule Engine* to continuously check for patterns in diagrams. Jess has been shown to be highly efficient. Used on an outdated machine (800 MHz Pentium III, Sun HotSpot JVM) Jess could fire up to 80,000 rules, match up to 600,000 patterns, and add up to 100,000 facts to the Jess knowledge base within one second [25]. This performance appears to be sufficient to support relevant use cases, such as class-based scenarios and research studies. A second performance-related aspect is the possibility to distribute system components on multiple machines to overcome the performance limitations of a single physical unit. First, CASE can be physically separated from other LASAD components (e.g., the *LASAD-Server*). Second, if CASE makes use of additional, computationally demanding analysis modules (e.g., involving natural language processing and machine learned models) these modules can be encapsulated in independent services, deployed on other server machines, and integrated through the remote proxy design pattern (similar to the above described *DeepLoopAgent*).

Stability and Availability of Service

One CASE installation might support many learning sessions. To avoid the situation where runtime errors or time-consuming analyses in one session affect other sessions, the processing of each session is isolated in an independent set of threads. Since CASE is only loosely coupled with the *LASAD-Server*, an outage of CASE for whatever reason would not affect the functioning of the main LASAD service.

Generality and Configurability

A key objective in the design of CASE was to achieve a high level of generality to support a wide range of learning scenarios and domains. Besides extensibility, which has been discussed above, the provision of configuration mechanisms that allow tailoring support to the specific constraints and requirements of a given application scenario is another important cornerstone in achieving generality. CASE allows configuring *FeedbackAgents* in terms of patterns they can identify, tutorial actions they can take in response to patterns, and tutorial strategies that govern their overall behavior. Configuration settings can be specified in XML files or through an API that allows changes to the configuration during runtime. A currently developed graphical configuration front-end, which allows teachers and researchers to manage and author feedback behavior in a simplified fashion, makes use of this API.

Section 6 describes the available configuration options in greater detail.

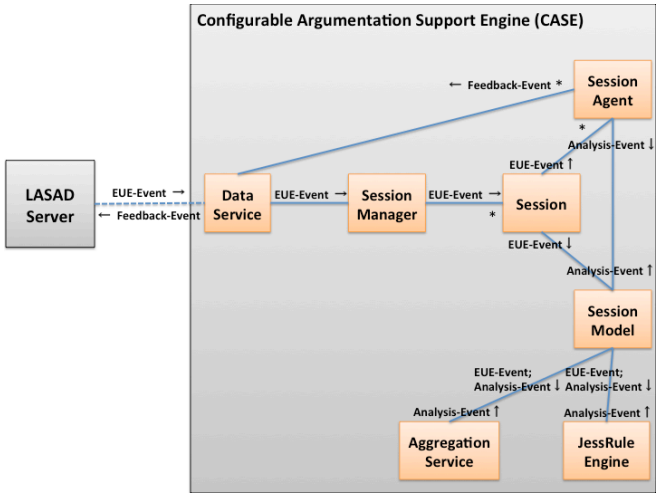

Fig. 3. CASE components and interactions.

5.3 System Components and Interactions

Based on Fig. 3, we discuss now the different CASE components (section 5.3.1) and their interactions (section 5.3.2). For the sake of brevity we will not cover the CASE configuration sub-system, which manages configuration settings and allows changing them at runtime.

5.3.1 Components

The **DataService** encapsulates all communication with the *LASAD-Server*, translating back and forth between the message format used by the *LASAD-Server* and the one used internally in the CASE framework. Messages include notifications (e.g., a diagram has been modified) and commands (e.g., display a given feedback message to a given user).

The **SessionManager** keeps a record of all sessions (i.e., diagrams) that exist on the *LASAD-Server* and distributes incoming messages to the relevant *Session*.

Sessions comprise all information that is associated with a specific LASAD diagram, including (1) a *SessionConfiguration*, which represents invariable aspects such as the used ontology (e.g., available box and link types) and available tools (e.g., chat tool), (2) a *SessionModel*, which represents fluent aspects such as currently active users and the current state of the diagram, and (3) *SessionAgents*, which are processing units employed to analyze and support session activities.

The **SessionModel** serves as a central data repository for *SessionAgents* to access and exchange session-related information. It uses the Jess rule engine to maintain and continuously update a representation of the current session state (e.g., active users; boxes and links in the current version of the diagram) and of analysis results, i.e., patterns and interpretations derived from the current session state. Analysis results can be produced in three ways: (1) Declarative Jess rules, which operate directly on the Jess fact base, identify patterns of interest (e.g., a circular argument in the diagram). (2) An *AggregationService*, which keeps track of the number of boxes, links and patterns that follow given specifications, detects that a predefined condition is fulfilled (e.g., more than x boxes of type t created by user u within the last y minutes). (3) *SessionAgents* conduct customized analyses (by e.g., applying machine learned classifiers) that lead to the detection of meaningful patterns.

SessionAgents are processing units that perform specific tasks related to the analysis of sessions and / or the generation of feedback. They interact with the *SessionModel* by adding or removing analysis rules (at service startup) and analysis results (during operation). Vice versa, the *SessionModel* informs *SessionAgents* about new or invalidated analysis results. The *SessionAgent* interface provides an extension point in the CASE framework to add new analysis and feedback capabilities. Already existing agents include a configurable *FeedbackAgent* (section 6), and the *DeepLoopAgent*, which integrates AI-based classifiers to analyze e-Discussions (section 4.4).

5.3.2 Interactions

We illustrate now a typical processing iteration in CASE, which is triggered by a student action in the end-user environment (EUE; the graphical LASAD user interface), such as creating a link, entering text into a box, sending a chat message, or requesting feedback.

The user action is encapsulated in an *EUE-Event*, sent through the *LASAD-Server* to the *DataService* component of the CASE framework, and there converted to the data format internally used in the CASE framework. From there, the *EUE-Event* is transmitted, via the *SessionManager* and the responsible *Session*, to the *SessionModel*.

The *SessionModel* distributes the *EUE-Event* to all listening analysis modules, including (1) the *JessRuleEngine*, which updates its internal session representation (Jess Fact Base) and performs pattern-matching operations according to predefined rules (Jess Pattern Matcher), (b) the *AggregationService*, which updates its internal tallies and checks whether predefined conditions on these tallies are fulfilled, and (c) *SessionAgents*, which update their internal models and perform their proprietary analyses, according to their specific implementation. The processing within the analysis modules may lead to the detection of salient patterns, such as structures in argument diagrams (e.g., a circular argument; two boxes connected by a link of the "wrong" type), conditions on tally counts (e.g., no box of type t in diagram), or text classifications (e.g., a box is categorized as an off-topic contribution based on the contained text). Detected patterns are encoded as *AnalysisResults*, packaged in *Analysis-Events* and re-distributed between the analysis modules for a second processing cycle. For instance, the *AggregationService* may check whether the number of patterns of a given type exceeds a threshold. The *JessRuleEngine* may check for logical combinations of patterns. The processing may involve further cycles.

One specific *SessionAgent* instance, the *FeedbackAgent*, responds to the presence of predefined *AnalysisResults* by generating *Feedback-Events*, which specify feedback to be displayed in the LASAD user interface. Generated *Feedback-Events* are delivered, through the *DataService* and *LASAD-Server*, to all relevant *End-User-Clients*.

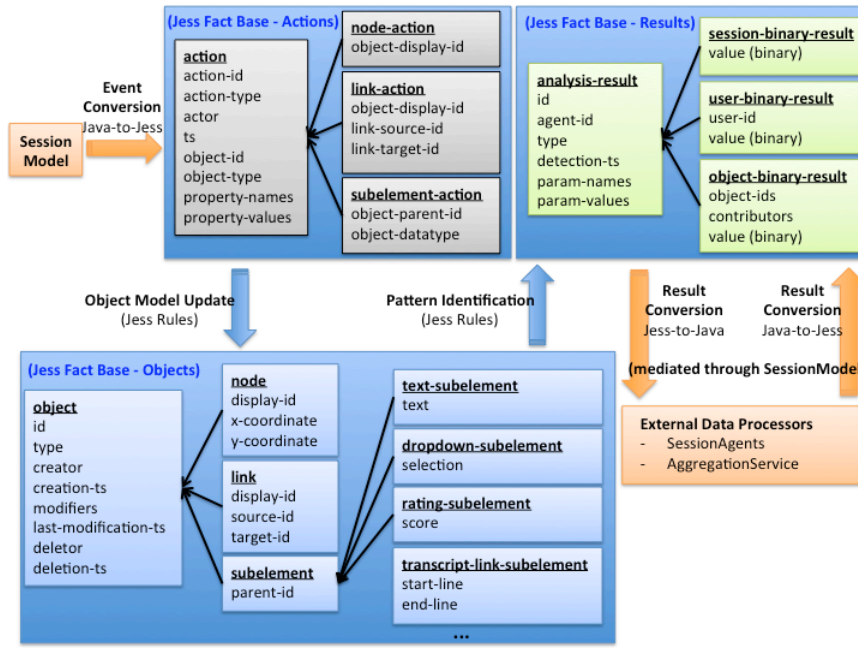

Fig. 4: Knowledge representation and inference processes

5.4 Knowledge Representation and Inference

A centerpiece in the CASE architecture is the *SessionModel*, which employs the *JessRuleEngine* to model the current state of LASAD sessions including *AnalysisResults* inferred from the current session state. The *SessionModel* provides a central place in the CASE infrastructure where processing components, such as the *AggregrationService* and *SessionAgents*, can retrieve session-related information and exchange their interpretations of the current session state. They can utilize the Jess pattern matching mechanism, which is based on declarative production rules, to identify salient structures and situations in order to trigger tutorial support or to facilitate subsequent (meta-level) analyses.

Fig. 4 depicts the knowledge representation scheme used in the Jess fact base (blue shaded areas), conversion procedures to translate between Java and Jess-based object representations (orange arrows), and inference procedures to derive new knowledge facts from existing ones (blue arrows). We now describe the processing of an incoming *EUE-Event* (i.e., a user action) within the *JessRuleEngine*: from its insertion, to the computation of derived facts representing the current diagram state, to the detection of patterns in the current diagram state, to the notification of listening processing modules about detected patterns.

EUE-Events arriving at the *SessionModel* are translated into Jess *action* facts and added to the fact base. Jess *action* facts structurally correspond to *EUE-Events*; they represent crucial information about user actions in the end user environment including: an action ID, the action type (e.g., "create," "modify," "delete"), the actor, a timestamp, and a description of the manipulated object in terms of its ID and semantic type (e.g., "hypothesis," "data," "support," "oppose"). Depending on whether the manipulated object is a box, a link, or a sub-element of a box / link (such as a text field or a dropdown menu), three action subtypes with additional information are defined: *node-actions*, *link-actions*, and *subelement-actions*.

Jess *action* facts are analyzed through a set of Jess rules to reconstruct the current state of the argument diagram in terms of its constituent objects (i.e., nodes, links, subelements). Besides state information, Jess *object* facts hold relevant process information, such as the object creator, modifiers, and corresponding timestamp information. Thus, the chosen knowledge representation format permits the definition of structural patterns (e.g., a node *n1* of type *t1* is connected to a node *n2* of type *t2* through a link *l* of type *t3*) that are further constrained through process-related properties (e.g., node *n1* and *n2* were created by different users; node *n1* was created before node *n2*). The incorporation of process-related information enables analyses that take into consideration temporal sequence (e.g., recent patterns versus old patterns) and patterns of group interaction (i.e., how diagram structures emerge from a joint effort of a group).

The current diagram state is analyzed through application-specific Jess rules to identify patterns of interest. The specific rules are defined as part of the *FeedbackAgent* configuration (for details, see section 6; for examples, see section 4). If a pattern has been identified a corresponding *analysis-result* fact is added to the fact base. Patterns might refer to specific sets of objects (*object-binary-result*), specific users (*user-binary-result*), or the session as a whole (*session-binary-result*). CASE allows the definition of meta-patterns, that is, patterns defined in terms of other patterns (e.g., logical combinations of patterns).

Finally, through a two-way conversion procedure between Jess and Java object representations, external data processors, such as the *AggregationService* and *SessionAgents*, read out and write *AnalysisResult* objects from and to the fact base.

The described data model can be, and has already

Application	Patterns	Actions	Strategy
LARGO (analysis of legal argumentation transcripts)	– *Count-Patterns* focusing on task progress (e.g., no relations in workspace but at least three nodes; no "Test" nodes in workspace) – *Jess-Patterns* focusing on domain structures (e.g., a "Hypothetical" node isolated from "Test" and "Fact" nodes; a "Test" node without text in the "Condition" text field) also including problem-specific aspects represented in an expert model (e.g., important text passages not yet included in the diagram)	– *Text message* (parameterized with diagram references) focusing on problem-solving support – *Highlighting* of diagram elements	– *Feedback-on-Request* – *Delivered to* the requestor – *Prioritize* based on current problem-solving phase (5 phases considered) – *Select top-5* hints – *Filter out* all but one message per type
Science-Intro (preparation for writing argumentative texts in science classes)	– *Jess-Patterns* focusing on domain structures (e.g., unconnected node clusters ["argumentation islands"])	– *Text message* (parameterized with diagram references) focusing on problem-solving support – *Highlighting* of diagram elements	– *Feedback-on-Request* – *Delivered to* the requestor – *Prioritize* based on current problem-solving phase (2 phases considered) – *Select top-5* hints – *Filter out* all but one message per type
Metafora (group deliberation about science and math problems)	– *Jess-Patterns* focusing on process characteristics (e.g., unattended help requests)	– *Text message* (parameterized with diagram references) focusing on collaboration support – *Highlighting* of diagram elements	*unattended help requests:* – *Automated-Feedback* – *Delivered to* the entire group – *No prioritization* – *Select* one message – *Filter out* instance that have already been pointed to
ARGUNAUT (argumentation about ethical controversies)	– *External-Patterns* (analyzed by machine learned classifiers) focusing on process characteristics (e.g., off-topic contributions; question-answer pairs)	– *Highlighting and labeling* of diagram elements to support the awareness of moderators regarding salient events	– *Feedback-on-Request* – *Delivered to* the requestor – *No prioritization* – *Select* all messages – *No filters*

Table 1: Configuration settings used in the four example applications.

been, extended with additional data structures to allow more complex analyses, for instance, by representing tallies, paths, cycles, or predefined expert models within the Jess fact base. Such data structures can be added either globally, to extend the entire CASE framework, or locally, as part of one *SessionAgent* and restricted in scope to a specific LASAD application. Moreover, the framework can be easily extended to represent user activities not related to argument diagrams, such as chat contributions.

6. CASE Configuration Mechanism

The behavior of *FeedbackAgents* is defined on three levels: Firstly, *patterns* describe salient structures or situations in a session, which call for tutorial support. Secondly, *tutorial actions* describe the specific reactions of the *FeedbackAgent* in response to detected patterns including textual statements and highlighting of relevant diagram contents. Thirdly, *tutorial strategies* describe when exactly to provide tutorial actions and which tutorial actions to choose when multiple ones are possible. For instance, a tutorial agent can provide feedback only when a student explicitly requests help, or act proactively, checking in predefined intervals whether relevant feedback is available. It can prefer messages that refer to more recent structures in the diagram and refrain from sending the same message a second time. Table 1 summarizes the specific configuration settings of the four previously discussed CASE applications. The following subsections describe the available configuration options in greater detail.

6.1 Patterns

Three different ways of defining patterns are currently supported:

(1) Jess-Patterns can be directly defined using the Jess rule language. This option offers the full expressive power of the Jess production rule system but also requires basic knowledge about Jess syntax and knowledge representation and understanding of the functioning of rule-based systems more generally. By modifying existing prototypical patterns it should also be possible for non-experts to define patterns of limited complexity without much effort.

Table 2 shows an XML element that defines a pattern. The *pattern* element specifies a pattern ID (*id*) and indicates that the pattern is defined in the Jess rule language (*type="jess-rule"*). In general, the *type* attribute determines how the body of the *pattern* element is interpreted. Accordingly, the other pattern types described below specify different values for the type attribute. CASE can be easily extended to support further pattern types.

The actual Jess-Pattern definition is enclosed in another XML element (*jess*). The pattern (LHS of the rule) comprises a node of type "hypothetical" and a second node of type "fact." There is a link of type "general" pointing from the "hypothetical" node to the "fact" node. When the pat-

tern on the rule's LHS is matched, the rule's RHS will be executed. Here, an *object-binary-result* fact that holds important information regarding the detected pattern (agent-id, pattern-id, matched objects) is added to the fact base.

Additional pattern filters can be defined (*pattern-filters*), for instance, to limit the scope to recent or old patterns, or patterns the user under consideration has contributed to. The example XML snippet in Table 2 does not define additional filters.

```
<pattern id="..." type="jess-rule">
  <jess>
   (defrule R35
     (logical
      (node
        (id ?hypo_id)
        (type "hypothetical"))
      (node
        (id ?fact_id)
        (type "fact"))
      (link
        (id ?link_id)
        (type "general")
        (source_id ?hypo_id)
        (target_id ?fact_id)))
     =>
     (assert
      (object-binary-result
        (agent-id "largo-default")
        (type "hypo_facts_relation_general")
        (object-ids ?hypo_id ?fact_id ?link_id))))
  </jess>
  <pattern-filters />
</pattern>
```

Table 2: Jess-Rule (simplified for illustration purposes)

(2) Count-Patterns define conditions on the number of boxes, links or other patterns in a session. They can be specified directly in XML (not shown here) and are processed by the *AggregrationService*, which adds new analysis-results automatically to the Jess fact base when a counter condition is fulfilled. A counter pattern consists of a *counter definition* (e.g., "count all links of type *t*") and a *counter condition* (e.g., "count >= 3"). Counter definition options include: (a) count all nodes or links; (b) count specific node, link, or pattern types; (c) count only recent or old instances; (d) maintain a counter for the entire group or for each individual user; and (e) count only instances the user under consideration has created, modified, or not modified. These options can be combined in a variety of ways to specify counter definitions.

(3) External-Patterns are analyzed by external components that connect with CASE over a well-defined API. The CASE framework acts as a mere "consumer" of these patterns, indifferent to how these patterns are defined or computed, so there are also no restrictions in this respect (e.g., machine-learned models can be used).

(4) XML-Patterns are based on a XML language that we have developed to reduce the complexity inherent in the original Jess rules. We are aiming at a tradeoff between expressiveness and ease of use. XML-Patterns are automatically translated into operational Jess code.

6.2 Tutorial Actions

Table 3 shows an XML element that defines a tutorial action. Analogously to the pattern definition described in section 6.1, the *type* attribute determines how the body of the action element is interpreted. That is, CASE can be easily extended with other types of tutorial actions that support different parameterization options. Feedback actions are activated by a specific pattern (*trigger*).

```
<action id="..." type="feedback">
  <trigger pattern-id="..." />
  <message>
   <short>...some text...</short>
   <long>...some text ...</long>
   <highlighting />
  </message>
  <priority>
   <default priority="1" />
   <phase idref="3" priority="10" />
  </priority>
</action>
```

Table 3: Tutorial Action Specification (simplified)

The feedback message itself (*message*) has three components: (1) a *short* message, which provides feedback in a concise way, (2) a *long* message, which provides a more detailed explanation, only displayed when the user clicks on the short message, and (3) a *highlighting* flag, which indicates whether objects that are part of the pattern should be visually highlighted in the user interface. The message texts can be formatted through HTML tags (e.g., bold, italics). Moreover, messages themselves can be parameterized through a control flag "[##*parameter-name*##]." When the message is delivered, this placeholder will be substituted by the actual value of the parameter, for instance, the box number displayed in the LASAD user interface to help students identifying diagram elements mentioned in the message text. In general, all information represented in the Jess fact base can be declared as a parameter and used in text messages.

The *FeedbackAgent* supports the prioritization of tutorial actions. There might be many messages activated at the same time. For instance, in some situations more than 100 messages were relevant and could have been provided in LARGO. To not overwhelm students, an informed selection of the most critical message is required. This decision should also consider the current problem-solving phases, since in each phase some messages may be relevant (and others not) and some tutorial actions may be preferable over others. For instance, at the beginning of creating a diagram, we might expect students to represent relevant statements. Only in a later stage might we expect them to interrelate these statements. A tutorial strategy might be to prompt students in the early stage to create boxes (statements; *pattern*: #boxes < X) and in the later stage to create links (relations; *pattern*: #boxes ≥ X ∧ #links < Y). The first part of this pattern ("#boxes ≥ X") defines a "relevance condition," which ensures that the pattern will not be activated in the earlier stage.

The phase-based prioritization procedure is governed by corresponding annotations of each tutorial action (*pri-*

ority). There is a default priority value (*default*) that can be overwritten by phase-specific priority values (*phase*). The assigned priority values serve essentially two purposes: In a first step, probabilities are assigned to each phase, based on all patterns (and associated priorities) that are detected in the current diagram state. That is, if many patterns that are strongly associated with a phase *X* are detected, then there will be a high probability for phase *X*. In a second step, all possible tutorial actions are prioritized according to their priority values for the most likely phase (which is assumed to be the current phase). This approach is based on the prioritization procedure developed in LARGO [11]. Whether the phase-based prioritization is ultimately activated, maybe as one component of a more comprehensive prioritization procedure, can be configured as part of the tutorial strategy specification described in section 6.3.

We are planning to enhance the available configuration options in future CASE versions. For instance, alternative modes of displaying feedback messages can support different levels of obtrusiveness (e.g., a separate feedback message panel, for which the user decides if and when to pay attention to, versus popup windows, which force the user to acknowledge that she has read the message before proceeding). Such enhancements enable an experimental investigation of different feedback realizations. A further step would be to *adaptively* decide whether to provide feedback obtrusively or not, depending on the current situation and the urgency of the detected problem. Corresponding decision heuristics could be configured as part of tutorial strategies, discussed next.

6.3 Tutorial Strategies

Table 4 shows an XML element that defines a tutorial strategy for tutorial actions. In this example, tutorial actions are triggered when the user explicitly requests feedback (*provision-time="on-request"*). To request feedback, the user can select a menu item in the user interface, labeled "Get hint" (*display-name*). We are currently working on a second option to deliver tutorial actions proactively, in predefined intervals. Tutorial actions are targeted at individual users (*recipient type="individual"*), as opposed to broadcasting them to each member of a learner group (*recipient type="group"*).

```
<provision id="..." type="sort-and-filter">
  <provision-time type="on-request">
    <display-name>Get hint</display-name>
  </provision-time>
  <recipient type="individual" />
  <provided-actions all-own-actions="true" />
  <action-filters>
    <action-filter type="one-instance-per-type" />
    <action-filter type="no-instance-twice" />
  </action-filters>
  <sort-criteria>
    <sort-criterion type="phase-priority" />
  </sort-criteria>
  <number-of-actions>5</number-of-actions>
</provision>
```

Table 4: Tutorial Strategy Specification (simplified)

In general, all tutorial actions defined in this *FeedbackAgent* are considered in this strategy (*all-own-actions="true"*). Alternatively, a selection of tutorial actions that are relevant can be enumerated. The set of relevant tutorial actions can be further reduced using a list of filters (*action-filters*). Here, each tutorial action is provided at most once at a time (*one-instance-per-type*). For instance, if the same pattern matches multiple structures in a diagram, only one feedback message is considered, rather than messages for each pattern instance. Which message this is depends on the given prioritization criteria, discussed below. The second filter (*no-instance-twice*) ensures that the same feedback message is never provided twice, based on the history of previous messages.

The resultant set of tutorial actions is then sorted according to a list of predefined criteria (*sort-criteria*). In the example, the phase-based prioritization procedure discussed in section 6.2 is activated (*phase-priority*). It is possible to combine different prioritization heuristics such as *phase-priority*, *prefer-recent-structures*, or *prefer-structures-not-yet-pointed-to*. The list of sort criteria is then processed from the top to the bottom. Criteria lower in the list are used as "tie-breakers." Only when preceding criteria cannot decide which tutorial action is most important subsequent criteria will be applied [26]. Finally, a cut-off point is defined (*number-of-actions*). In this specific example the first five tutorial actions of the sorted action list are delivered. Through its modular design, CASE can be easily enhanced with further filter and prioritization criteria.

7 Knowledge Engineering Model

In this section we describe how to design, implement, and evolve adaptive support for argumentation diagramming activities in a systematic manner using the CASE framework. We will hint at several relevant pedagogical considerations; a more detailed treatment regarding the pedagogy of system-delivered feedback can be found elsewhere [4], [27]. We propose a knowledge engineering process that unfolds into *inner cycles* in which system configurations are developed in a three-step process and *outer cycles* in which the resultant system configurations are tested to inform improvements for the next design iteration. Authors will typically not execute the steps of the inner cycle in strict sequence, but rather move opportunistically back and forth between steps to fine-tune and match the definitions of patterns, messages and strategies.

Step 1: Pattern Definition. The first step is to decide *what* patterns are relevant and should be reacted to. A basic decision is which processes to support: problem solving, collaboration, or both. The definition of specific patterns may be based on theoretical considerations / previous research (e.g., typical problems in student-student interactions), concrete problems observed in preceding sessions, or a combination of both approaches (for instance, by checking whether problems reported in the literature can actually be observed in the current setting). An approach on how to operationalize the identified patterns within the CASE framework must be chosen. If patterns are well defined or can be heuristically approximat-

ed (e.g., diagram constellations that violate syntactic constraints), Jess rules (*Jess-Patterns*) or counter conditions (*Count-Patterns*) can be used. If patterns cannot be easily described in a declarative format, possibly because the conceptual idea does not translate easily into a concrete executable definition or the pattern is overly complex (e.g., patterns in natural language expressions), a feasible approach might be to automatically induce patterns using machine learning. The development of machine-learned classifiers is a separate knowledge engineering process involving collecting, coding and preprocessing of data; experimentation; and performance validation [19]. A number of general-purpose machine-learning toolkits are available to support that process, e.g., WEKA [28]. The resulting classifiers can be integrated into CASE as external services (*External-Patterns*) using a predefined API.

Step 2: Message Definition. The second step is to decide *how* to respond to patterns. Typically, non-authoritative message formulations are preferable since patterns are often heuristic in nature and do not necessarily, and without fail, identify errors on the part of the student [26]. Other important decisions include feedback / advice specificity (e.g., conceptual versus procedural hints), message length (e.g., students might be more inclined to read shorter messages), and message tone (polite versus impersonal). If the message relates to some concrete structures in the diagram, highlighting of these structures is often helpful to students in locating relevant diagram contents quickly.

Step 3: Strategy Definition. The third step is to decide *if and when* to respond to specific patterns. On the one hand, feedback-on-request may not be used frequently, even if students would benefit from it [11], [26]. On the other hand, unsolicited feedback may interrupt the problem solving / collaboration process and be perceived as annoying by the students. Mixed strategies might be considered, e.g., immediate, unrequested feedback to remediate collaboration problems (e.g., one student does not contribute to the solution at all), and feedback-on-request to give students the option to request hints on how to proceed with the task when they are stuck. If the number of patterns that can occur at a time is high, strategies should involve the informed prioritization and selection of messages. Authors may develop an idealized problem-solving model that subdivides the process into discrete, consecutive phases, each associated with patterns and messages that are particularly important in that phase. This model might be built based on theory and expert judgment or by inspecting prior learning sessions. Based on such models, messages can be annotated with phase-specific priority values, which are used by the system to make an informed selection between messages. Pattern definitions might also be adjusted to ensure that patterns are not activated before they become relevant according to the phase model ("relevance conditions").

8. Conclusion and Future Work

In this paper, we presented the CASE framework, a highly configurable software component to analyze and support educational argument diagramming activities. The CASE architecture has been devised with important software design concerns in mind. Maintainability and extensibility have been achieved through a modular design and predefined extensions points, which enable new functionality to be easily added. In order to make CASE highly configurable and thus usable across a wide range of scenarios and domains we have created a comprehensive configuration sub-system, parameterizable through XML and a dedicated API, allowing configuration changes at any time. The built-in mechanisms for parameterizing tutorial behavior enable researchers and practitioners to create tutorial support across a wide range of application. To illustrate this we presented and discussed four applications that demonstrate the diversity of CASE including application to different domains, student tasks and types of tutorial support.

Despite the many aspects of CASE that have been developed, there is potential for pushing the envelope further. An important contribution, distinguishing CASE from other pattern-matching approaches to support argumentation (e.g., [26], [11]), is its capability to identify and respond to patterns in student-student interactions. Yet, student collaboration includes chat discussions, in addition to actions taken in argument diagrams, so students can coordinate diagramming activities and discuss diagram contents. To assess the quality of collaboration in a more precise and comprehensive way, we plan to extent CASE so that diagram and chat activities can be jointly analyzed (*cross-modality analysis*).

Another CASE asset, whose potential we intend to extend, is the integration of external analysis modules, as exemplified by ARGUNAUT's Deep Loop [19]. Elsewhere, we describe how individual Deep Loop classifications, which only refer to a relatively small portion of an argument diagram (e.g., a single contribution or a pair of contributions), can be aggregated to summarize a discussion in a more holistic way [29]. CASE provides a framework that can accomplish this. With a few technical enhancements CASE could aggregate and combine patterns originating from external analysis modules in various ways. Another interesting option is to include analysis results that are not based on argument diagramming activities at all. For instance, in Metafora, LASAD is only one of several learning tools. Students also use microword simulations, some of which include tool-specific analysis modules. Integrating these analysis modules with the CASE framework allows *cross-tool analyses*, which correlate students' structural discussions in LASAD with their activities in other tools. From a pedagogical perspective, existing diagnostic capabilities could be improved – or new ones enabled – with the additional accessible information. From a technical perspective, well-established and tested analysis modules can be reused.

Finally, some lecturers use argument diagramming as an integral part of their classes, that is, students use a diagramming tool regularly, throughout an entire semester (or even beyond) [30]. The current version of CASE has been designed with a focus on a single session. Hence, a

possible extension is a *cross-session student model*, which continuously assesses students (argumentation) skills and misconceptions, based on the quality of created diagrams. On the basis of such a model, feedback could be better tailored to individual students based on his or her learning history. Moreover, in the sense of a classical ITS outer loop [31], problems could be selected appropriate to the student's current skill level. For instance, a LARGO transcript could be chosen to match the transcript difficulty with the student's level of expertise.

Acknowledgements. We would like to thank Kevin Ashley, Toby Dragon, Collin Lynch, and Niels Pinkwart for feedback regarding the descriptions of CASE applications. This work is supported by the German Research Foundation (DFG) under the grant "Learning to Argue: Generalized Support Across Domains" (LASAD).

REFERENCES

[1] R. Driver, P. Newton, and J. Osborne, "Establishing the norms of scientific argumentation in classrooms," *Science Education*, vol. 84, no. 3, pp. 287–312, 2000.

[2] T. van Gelder, "Argument mapping with Reason!Able," *The American Philosophical Association Newsletter on Philosophy and Computers*, vol. 2, no. 1, pp. 85–90, 2002.

[3] O. Scheuer, F. Loll, N. Pinkwart, and B.M. McLaren, "Computer-Supported Argumentation: A Review of the State of the Art," *Intl. J. Computer-Supported Collaborative Learning*, vol. 5, no. 1, pp. 43–102, 2010.

[4] O. Scheuer, B.M. McLaren, F. Loll, and N. Pinkwart, "Automated Analysis and Feedback Techniques to Support and Teach Argumentation: A Survey," *Educational Technologies for Teaching Argumentation Skills*, N. Pinkwart, and B.M. McLaren, eds., pp. 71-124, Bentham Science, 2012.

[5] F. Loll, N. Pinkwart, O. Scheuer, and B.M. McLaren, "How Tough Should It Be? Simplifying the Development of Argumentation Systems using a Configurable Platform," *Educational Technologies for Teaching Argumentation Skills*, N. Pinkwart and B.M. McLaren, eds., pp. 169–197, Bentham Science Publishers, 2012.

[6] A. Soller, A.M. Monés, P. Jermann, and M. Mühlenbrock, "From Mirroring to Guiding: A Review of State of the Art Technology for Supporting Collaborative Learning," *Intl. J. Artificial Intelligence in Education*, vol. 15, pp. 261–290, 2005.

[7] I. Magnisalis, S. Demetriadis, and A. Karakostas, "Adaptive and Intelligent Systems for Collaborative Learning Support: A Review of the Field," *IEEE Trans. Learning Technologies*, vol. 4, no. 1, pp. 5–20, 2011.

[8] P. Bell, "Using Argument Representations to Make Thinking Visible for Individuals and Groups," *Proc. 2nd Intl. Conf. Computer-Supported Collaborative Learning (CSCL-97)*, R. Hall, N. Miyake, and N. Enyedy, eds., pp. 10–19, Toronto: University of Toronto Press, 1997.

[9] O. Scheuer, B.M. McLaren, A. Weinberger, and S. Niebuhr, "Promoting critical, elaborative discussions through a collaboration script and argument diagrams," *Instructional Science*, 2013. doi:10.1007/s11251-013-9274-5

[10] F. Loll and N. Pinkwart, "LASAD: Flexible representations for computer-based collaborative argumentation," *Intl. J. Human-Computer Studies*, vol. 71, no. 1, pp. 91–109, 2013.

[11] N. Pinkwart, K.D. Ashley, C. Lynch, and V. Aleven, "Evaluating an Intelligent Tutoring System for Making Legal Arguments with Hypotheticals," *Intl. J. Artificial Intelligence in Education*, vol. 19, no. 4, pp. 401–424, 2009.

[12] A. Soller, "Supporting social interaction in an intelligent collaborative learning system," *Intl. J. Artificial Intelligence in Education*, vol. 12, pp. 40–62, 2001.

[13] K. Ashley, *Modeling Legal Argument: Reasoning with Cases and Hypotheticals*, Cambridge MA: MIT Press/Bradford Books, 1990.

[14] C. Lynch and K.D. Ashley, "Modeling Student Arguments in Research Reports," *Proc. 4th AHFE Conf.*, V.G. Duffy, ed., pp. 191-201, CRC Press, 2012.

[15] T. Dragon, B.M. McLaren, M. Mavrikis, A. Harrer, C. Kynigos, R. Wegerif, and Y. Yang, "Metafora: A Web-based Platform for Learning to Learn Together in Science and Mathematics," *IEEE Trans. Learning Technologies*, in press.

[16] D.W. Johnson and R.T. Johnson, *Learning together and alone: Cooperation, competition, and individualization*, p. 53, Englewood Cliffs, NJ: Prentice Hall, 1991.

[17] N.L. Kerr, "Motivation losses in small groups. A social dilemma analysis," *J. Personality and Social Psychology*, vol. 45, pp. 819–828.

[18] M.W. Berkowitz and J.C. Gibbs, "Measuring the developmental features of moral discussion," *Merrill-Palmer Quarterly*, vol. 29, pp. 399–410, 1983.

[19] B.M. McLaren, O. Scheuer, and J. Mikšátko, "Supporting collaborative learning and e-Discussions using artificial intelligence techniques," *Intl. J. Artificial Intelligence in Education*, vol. 20, no. 1, pp. 1–46, 2010.

[20] B.B. Schwarz and A. Glassner, "The role of floor control and of ontology in argumentative activities with discussion-based tools," *Intl. J. Computer-Supported Collaborative Learning*, vol. 2, no. 4, pp. 449–478, 2007.

[21] H.U. Hoppe and K. Gaßner, "Integrating collaborative concept mapping tools with group memory and retrieval functions," *Proc. of the Conf. Computer Supported Collaborative Learning 2002*, G. Stahl, ed., pp. 716–725, 2002.

[22] M. Baker, J. Andriessen, K. Lund, M. van Amelsvoort, and M. Quignard, "Rainbow: A Framework for Analyzing Computer-Mediated Pedagogical Debates," *Intl. J. Computer-Supported Collaborative Learning*, vol. 2, no. 2–3, pp. 247–272, 2007.

[23] R. Wegerif, B.M. McLaren, M. Chamrada, O. Scheuer, N. Mansour, J. Mikšátko, and M. Williams, "Exploring creative thinking in graphically mediated synchronous dialogues," *Computers & Education*, vol. 54, no. 3, pp. 613–621, 2009.

[24] A. Harrer, S. Ziebarth, A. Giemza, and U. Hoppe, "A framework to support monitoring and moderation of e-discussions with heterogeneous discussion tools," *Proc. 8th IEEE Intl. Conf. Advanced Learning Technologies 2008*, pp. 41–45, 2008.

[25] E. Friedman-Hill, *Jess in Action: Java Rule-Based Systems*, p. 38, Greenwich CT: Manning Publications, 2003.

[26] D. Suthers, J. Connelly, A. Lesgold, M. Paolucci, E. Toth, J. Toth, and A. Weiner, "Representational and Advisory Guidance for Students Learning Scientific Inquiry," *Smart machines in education: The coming revolution in educational technology*, K.D. Forbus and P.J. Feltovich, eds., pp. 7–35, Menlo Park: AAAI/MIT Press, 2001.

[27] V.J. Shute, "Focus on Formative Feedback," *Review of Educational Research*, vol. 78, no. 1, pp. 153–189, 2008.

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[29] O. Scheuer and B.M. McLaren, "Helping teachers handle the flood of data in online student discussions," *Proc. 9th Intl. Conf. Intelligent Tutoring Systems*, B. Woolf, E. Aimeur, R. Nkambou, and S. Lajoie, eds., pp. 323–332, Berlin: Springer, 2008.

[30] C.R. Twardy, "Argument Maps Improve Critical Thinking," *Teaching Philosophy*, vol. 27, no. 2, pp. 95–116, 2004.

[31] K. VanLehn, "The behavior of tutoring systems," *Intl. J. Artificial Intelligence in Education*, vol. 16, no. 3, pp. 227–265, 2006.

Oliver Scheuer is a researcher at the Center for e-Learning Technology (CeLTech), at Saarland University, Germany. His research concerns adaptive technologies for computer-supported collaborative learning, in particular, argumentation learning. In this field, he has published more than 30 papers in peer-reviewed journals, conferences and workshops, focusing on topics such as the application of educational data mining and intelligent tutoring techniques, the design of software architectures, and empirical studies of learning.

Bruce M. McLaren is a Senior Systems Scientist in the Human-Computer Interaction Institute at Carnegie Mellon University in Pittsburgh, PA USA and an Adjunct Senior Researcher with the Center for e-Learning Technology (CeLTech), at Saarland University, Germany. Dr. McLaren has research interests in educational technology, collaborative learning, intelligent tutoring, and artificial intelligence. He has over 100 publications in peer-reviewed journals, conferences, and workshops.