Employing a Java Expert System Shell for Intelligent Support in Exploratory Activities

Charles Hunn

Biz Logic Solutions**
charleshunn@yahoo.ca

Abstract. This paper presents issues around the integration of a framework for the development of interactive exploratory activities (DANTE) and a rule engine and scripting environment (JESS). The paper initially presents these two systems and their use. We then discuss design decisions and implementation considerations providing an insight for researchers and developers who are considering the integration of exploratory activities with expert systems such as JESS. Finally, the paper presents further lines of research that could potentially provide cost-effective development tools for intelligent exploratory environments.

Key words: authoring, exploratory activities, expert systems

1 Introduction

Exploratory and interactive activities are usually integrated within instructional material to provide learners with the opportunity to develop through exploration usually a qualitative or sometimes a deep procedural understanding of a concept. The development of such exploratory environments or activities is quite time consuming. In addition, the extent to which they achieve the expected learning objectives relies on the effectiveness of the students' exploratory behaviour [3] and many studies (e.g., [4, 5] among others) have shown that there is a need for additional support other than just the interaction with the environment itself. It follows naturally then, that if such environments are to be at all effective their design must incorporate considerable expert knowledge of pedagogy and possibilities for intelligent support.

A drawback however, is that the more sophisticated the pedagogical model in a system the greater the challenge of knowledge representation and engineering. This problem is further compounded by the need to represent the actual subject domain knowledge in the system as well. A consequence of the need for a high level of embedded knowledge is that the development of an Intelligent Tutoring System (ITS) is generally a very time consuming process. Analysis suggests that it takes approximately 100-1000 hours of development time to produce 1 hour of

** Some of the work presented here is part of the author's MSc [1] and his employment at a Small Project Grant for the WaLLiS project [2], both at the University of Edinburgh.

instruction from an ITS [6] . Moreover, developers often require specialist skills or combinations of skills such as AI programming, expert knowledge of the subject domain, cognitive task analysis, etc. Whilst a few ITS's are developed by domain experts who have AI programming skills, many systems are developed by teams of AI programmers, subject domain experts and cognitive scientists. In such a situation time is inevitably expended in the communication of requirements, specifications and constraints. Thus, given the increasing demand for full commercial-scale ITS and the large development time for such systems, there is clearly a need for the ITS development process to be made faster, simpler and more accessible to non-programmers (e.g. teachers who are interested in developing their own content)

Although there have been several attempts to reduce the time to develop exploratory and interactive activities, and despite the fact that several platforms and frameworks (e.g., Flash, other frameworks that enable the authoring of Java Applets etc.) can be used to develop exploratory activities, work on the ways to support students while working with these activities has been quite limited until recently. Some researchers have begun to address this issue but despite the fact that there has been substantial progress in tools that support the authoring of interactive exercises for procedural domains [7], there is little work on finding ways to minimize the effort of the designer to develop tools or components of systems that can reduce the complexity and difficulty of developing exploratory let alone intelligent environments.

Motivated by the need to make the ITS development process faster, easier and more accessible to non-programmers, [1] investigated the integration an existing framework for developing exploratory activities (DANTE [4]) with the Java Expert System Shell (JESS) [8]. The overall aim behind the research reported here was to establish a better representation of the pedagogical knowledge that underlied the DANTE framework. The rationale for this approach was that a more suitable knowledge representation might result in greater speed and ease of system development. To achieve this, JESS was used to replace the pre-existing feedback module in DANTE, originally coded in a procedural language.

This paper, after briefly presenting DANTE and JESS, discusses some of the design decisions and considerations when integrating the two systems, providing some insight for researchers and developers who are considering the integration of Java applications with the computational power of JESS in particular and expert systems in general.

2 Background

2.1 DANTE and its integration in a web-based ILE

The School of Mathematics of the University of Edinburgh has developed a web-based Interactive Learning Environment called WaLLiS [2]. Its role is to provide instruction in mathematics to undergraduate science and engineering students. The rationale for the development of the system is to address a lack of basic

maths skills seen in school-leavers entering higher education. Some important features of WaLLIS are that it is easily-accessible, interactive, intelligent and responsive to individual students needs [2].

At the heart of the 'intelligence' of WaLLiS is an adaptation of a feedback mechanism originally developed in the Dynamic Authoring aNd Tutoring Environment (DANTE) [4, 9]; a prototype framework for developing interactive exploratory activities for dynamic geometry environments (see an example in Figure 1). The rest of the description focuses mostly on DANTE as it bears more relevance to the work described here in relation to exploratory environments. WALLIS has been described in detail in [2, 10]

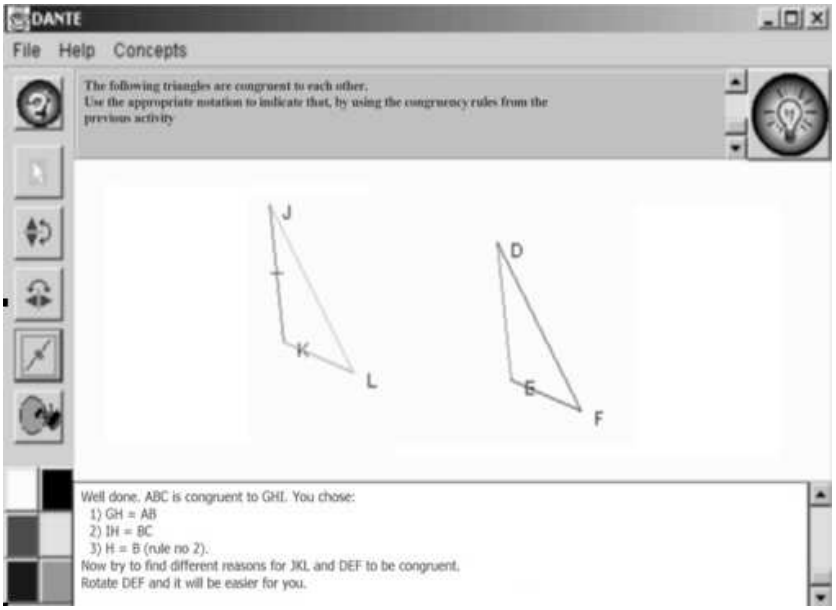

Fig. 1. An exploratory activity designed in DANTE. It asks from students to identify why two triangles are congruent and to indicate that using the tools of the system (reproduced with permission from [4]).

DANTE is divided into two components: the feedback mechanism and the components for the design of the interactive activities. The activities have to follow a certain programming interface which allows the feedback mechanism to monitor the learner's interaction.

The activity designer, usually a programmer, must specify (in Java code, using the API provided by the DANTE framework) the goals and sub-goals of the activity. For the exploratory activities, the goals are described in terms of what configurations the various available objects should be in. The activity-specific feedback text for the activity must also be added at this point and linked to

the appropriate goals. To simplify this process DANTE provides JavaBeans [11] components such as toolbars, inputs, function and integral plotters, geometric components and others. These state-aware components allow other parts of the application to register with them to be notified of any changes to the component state.. For example, the main canvas notifies any interested components that is being dragged, clicked and so on. Similarly, textboxes announce whether something is being typed in them, buttons whether they are clicked etc. In addition, following DANTE's programming interface, one can build other state-aware components or whole activities in other languages (eg. javascript, Flash) as long as they provide a 'wrapper' to communicate with the feedback mechanism [9].

The core part of DANTE is its feedback mechanism which responds according to rules that fire when the conditions, which the activity designer has set, are met. The initial mechanism as implemented in [4] and [2] expects, for each activity, a Java class that provides the specic rules, goals, actions and misconceptions (buggy-rules) for this activity. The goals of the activity form a tree structure (see 2) where goals are comprised of a number of subgoals each of which has a number of conditions and misconceptions associated with it. The feedback mechanism monitors the list of conditions for each subgoal, and if they are not satisfied checks for buggy rules and setsthe corresponding actions in motion.

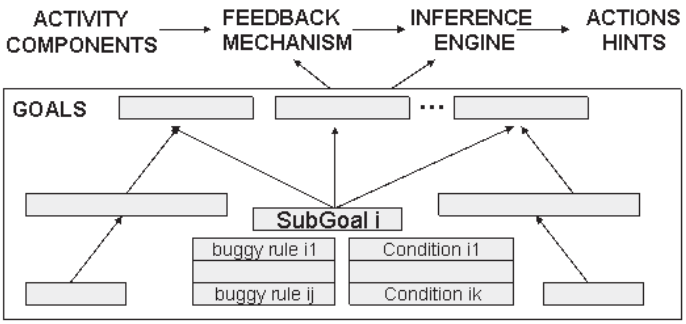

Fig. 2. Each subgoal has a number of misconceptions and completion conditions associated with it, which the feedback mechanism applies to the inference engine (reproduced with permission from [9]).

Despite the aforementioned support from the framework, authoring is still quite a time consuming process and requires not only an understanding of the DANTE framework but also a substantial knowledge of Java.

2.2 Overview of JESS

The Java Expert System Shell (JESS) is an expert system shell and scripting language written in Java. JESS was originally a clone of the popular CLIPS

(C Language Integrated Production System) expert system which was developed by a team at NASA in 1987. However, JESS has since taken on distinct language features that enhance its Java related capabilities. A consequence however is that code cannot always be ported between JESS and CLIPS. For more details about JESS the reader is referred to the JESS manual[1] and [8].

Perhaps the greatest advantage of JESS is that it can be easily integrated with applications written in Java through an application programming interface that is accessible from Java. This makes it particularly suitable for any application of a web-based expert system due to the platform independence of the Java language. As described in [8] because of its "flexibility, JESS can be used in command-line applications, GUI applications, servlets, and applets", therefore enhancing them with reasoning capabilities.

3 Integration of JESS and DANTE

As mentioned in Section 1, the main objective of this research was to re-implement the existing Java feedback mechanism as an expert system using JESS and to interface JESS with the existing architecture of DANTE. The details of this integration are available in [1]. Here we would like to present some of the most important decisions made towards this objective which are applicable when integrating JESS (or to some extent other expert-systems) with educational applications and particularly exploratory activities.

3.1 Choosing an Application Architecture

A number of design options for the application architecture are possible each employing a different balance of the JESS and Java languages [8]. According to the developers of JESS this is the most important design decision when developing an application with the shell. At one extreme it is possible to write a Java program entirely in the JESS language since JESS is capable of directly accessing Java libraries. At the other extreme it is possible to develop entirely in Java, creating the rules and other constructs required for an expert system in JESS through the API. A typical application structure is something in the middle of these two extremes outlined above with perhaps a Java application that interacts with an expert system that was written in JESS script.

In the context of the present study the choice of application architecture was constrained to just two possibilities, because the application is already largely developed in Java. Thus the possibilities are: (1) mostly Java code which loads JESS language scripts at runtime or (2) all Java code which manipulates JESS entirely through its Java API. In deciding on this matter the following considerations had to be taken into account:

- What is there to be gained from keeping the development entirely in one language? (i.e. Java).

[1] The manual is available online: `http://www.jessrules.com/`

- Would it be better from the perspective of rule- authoring for the editing to be done outside of the Java environment given that potential rule-authors may be non-programmers?
- At the syntactical level which of the two programming approaches would be easier to implement?

There do not seem to be any significant benefits of keeping the development entirely in Java. Perhaps one argument is that the Integrated Development Environments (IDE) which are available for the Java language are very sophisticated compared to those for JESS. There are however editor modes for JESS that fulfil many basic editor requirements.

From the perspective of rule-authoring, it could be more appropriate to have the pedagogical knowledge encoded in rules in a JESS script. Apart from making the code more modular and manageable, there is another conceivable benefit of this decision; rule-authors will have a simpler development environment to work in, assuming a little knowledge of the JESS syntax and the framework provided by the work described here. The final factor considered in deciding the application architecture was the differences, at the syntactical level, between creating a JESS expert system in Java alone or Java with JESS. Apart from ease of maintenance, simplicity and readability are also important considerations when authoring, debugging and changing code. It was thought that the Java implementation of the rule declaration increases the complexity of the rules from both a syntactic and 'elegance' point of view as it not possible to show the nesting of the rules and it requires extra supporting Java code (see [8] for examples).

A further and significant advantage of developing rules in a separate JESS script is that no compilation is required. This means potential activity-designers would not need to re-compile anything, simplifying the authoring process and the potential access they have to the rest of the software.

Thus, based on the evaluation of the three design considerations for system architecture outlined above, we chose to implement the system with Java code which loads JESS language scripts at runtime.

3.2 Enabling JESS to Reason with the state of Java components

One of the key advantages of JESS is that it is capable of reasoning with both pure logical facts and also with facts based on Java objects. The DANTE components were particularly suited for that as they follow the JavaBeans API, which enables the authoring of rules in JESS that include conditions referring to the state of Java objects.

An implementation decision that had to be taken was between 'static' and 'dynamic' updating of JESS with the actual values of the Java object. Static updating only refreshes the JESS knowledge base whenever a reset command is sent to JESS. With dynamic updating, any changes in the state of Java objects are reflected immediately in the JESS knowledge base.

Although this may seem only a technical decision, it has significant pedagogic implications, particularly for exploratory activities. These often involve experimentation with some graphical or other representational component in the interface. In addition, the 'correct' state of a component may be within a range of states rather than a specific one. In order to provide help therefore, any intelligent tutor must be able to monitor continuously the changing values of the interface component. The dynamic updating of the facts allows exactly that and enables the authoring of rules (and hence the provision of appropriate feedback) that capture critical states of the interface components. The alternative (static updating) would require an explicit request from Java to JESS (e.g., through a timer mechanism, or only when a student asks for help) which would constrain the type of help that may be provided.

3.3 Fuzzy reasoning

Aside from the basic employment of JESS as a means of improved knowledge representation we also represented some aspects of the feedback mechanism in fuzzy logic. The FuzzyJESS toolkit API of JESS allows developers to easily create fuzzy variables, fuzzy sets, fuzzy values and fuzzy rules. Options are provided for specifying which fuzzy inference algorithm will be used to execute the rule. In order to investigate the full potential of fuzzy logic FuzzyJESS was employed as it adds to JESS a number of fuzzy logic functions, enabling a rich combination of fuzzy and rule-based reasoning all in one system. This endows JESS with the ability to have fuzzy facts in the knowledge base and in the antecedents and conclusions of rules. This proved moderately suitable for representing several of the concepts in the feedback mechanism, and increases the potential of authoring rules based on knowledge elicitation from experts. The knowledge representation therefore was an improvement on the JESS-only implementation as a variety of linguistic terms (e.g., 'slightly', 'medium' or 'large') are available to describe states of the variables. This gives potential activity-authors considerable more power in their specification of values. However, as outlined in [12], there is a range of difficulties associated with fuzzy reasoning in terms of the amount of optimisation work required to make the inferencing process generate the required behaviour. Moreover, as pointed out in [2], optimisation of an ITS is in any case quite a significant task. Thus, the feasibility of introducing fuzzy logic into an ITS must be determined by balancing out the benefits of better knowledge representation against the possible disadvantage of increased time spent optimising the system.

4 Discussion

The main aim of the research described here was to investigate how the implementation of a specific feedback mechanism of the pre-existing DANTE framework in JESS, might extrapolate to the ease of general activity-authoring. Taking into consideration the aforementioned issues, the JESS implementation appears

to satisfy our overall objectives although we recognize the need for thorough evaluation of the chosen approaches. It remains to be seen in practice if employing JESS has the potential to reduce the time and complexity of authoring of exploratory activities.

The last few years there are few educational environments that are making their appearance and utilize JESS as the intelligent component of the system (e.g. [7, 13–15]). Developing a framework the integration of which not only provides the system's intelligence, but also simplifies the authoring of activities is quite challenging. A number of factors indicate that JESS is suitable for such a task. Whilst it is still a 'programming language' it is quite far removed from the complex syntactical representations typical in many procedural programming languages. From the observations made in a preliminary evaluation (see [1]) it seems clear that JESS with the declarative way of representing the rules, along with useful debugging features such as the 'watch rules' function to monitor rule firing, would indeed be feasible for our goal. This standpoint is further supported by the fact that the JESS rules scripts could even be dynamically reloaded into a running authoring application.

The investigation carried out into the use of FuzzyJESS also has some implications for activity-authoring. The representation of vague concepts using linguistic variables has an intuitive appeal, especially when the potential rule-authors may be non-programmers. However, consideration needs to be made of the fact that there is a wealth of options for customising the fuzzy reasoning process in FuzzyJESS. Moreover, to make the system behave in the desired way may require some considerable time spent on fine-tuning the fuzzy inference process. This factor could make the introduction of FuzzyJESS contradictory to the goal of speeding up and simplifying the ITS development process.

Additionally, it is clear that JESS offers some obvious advantages over Java in terms of knowledge representation, the present study along with others (e.g. [13]) indicate that JESS does not impose any limitations on web-based ITS. In addition to being easily integrated with other Java web-based applications, the core engine of JESS is optimised for fast performance, which is a required feature of an ITS since timely feedback is necessary.

Since effective authoring and debugging requires powerful tools, a line of research that is worth investigating is the continuation of the work presented in [16] which employed the Cognitive Tutor Authoring Tools (CTAT) [7] to author exploratory activities for DANTE and WaLLiS. CTAT is a framework for rapid development of Cognitive Tutors that supports all stages of the design and development of a cognitive tutor and is written to work alongside with JESS. Its main aim is the development of cognitive tutors that target procedural skills within well-defined problem spaces. This introduces some challenges in relation to using CTAT for developing exploratory activities, which are related to the discussion in Section 3.2. In particular, as discussed in more detail in [17], the tools CTAT provides are designed to deal with procedural steps that correspond to discrete actions in the interface. This is antithetical to the continuously changing values of the interface components in exploratory activities. As a consequence, this re-

stricts not only the possible types of feedback that can be designed with CTAT but also the power of some of the tools in CTAT. However, recent developments and additional tools, such as the "Example-Tracing" tool that provides simple tutors by demonstration [7, 18], or the "Simulated Student" tool [19] and other tools that support rule-authoring, controlled experiments and data analysis [18] make CTAT worth revisiting and extending further to cope with exploratory activities.

A natural follow-up to this research is the implementation of more activities and more formal comparisons along the lines of the evaluations of other authoring tools [20]. These evaluations should consider the efficiency of the expert system in several educational settings and, equally important, the time spent in authoring activities.

Acknowledgements. The author would like to thank Manolis Mavrikis for his advice and help during the research reported here and the reviewers of previous versions of this paper for their insightful and critical comments.

References

1. Hunn, C.: Employing a Java expert system shell for a web-based ITS. Master's thesis, The University of Edinburgh, School of Informatics; AI (2001)
2. Mavrikis, M., Maciocia, A.: Wallis: a web-based ILE for science and engineering students studying mathematics. Workshop of Advanced Technologies for Mathematics Education in 11th International Conference on Artificial Intelligence in Education, Sydney Australia (2003)
3. Bunt, A., Conati, C.: Probabilistic student modelling to improve exploratory behaviour. User Modeling and User-Adapted Interaction **13** (August 2003) 269–309
4. Mavrikis, M.: Towards more intelligent and educational Dynamic Geometry Environments. Master's thesis, The University of Edinburgh, Division of Informatics; AI (2001)
5. Kirschner, P., Sweller, J., Clark, R.: Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential and inquiry-based teaching. Educational Psychologist **41**(2) (2006) 75–86
6. Murray, T.: Authoring intelligent tutoring systems: An analysis of the state of the art. International journal of artificial intelligence in education **10** (1999) 98–129
7. Koedinger, K.R., Aleven, V., Heffernan, N.T.: Toward a rapid development environment for cognitive tutors. In: 12th Annual Conference on Behavior Representation in Modeling and Simulation., Simulation Interoperability Standards Organization (2003)
8. Freidman-Hill, E., ed.: Jess in Action. Manning Publications Co. (1995)
9. Mavrikis, M.: Improving the effectiveness of interactive open learning environments. In: 3rd Hellenic Conference on Artificial Intelligence (SETN) - proceedings companion volume. (2004) 260–269
10. Mavrikis, M., Maciocia, A.: Wallis: a web-based ILE for science and engineering students studying mathematics. three years on. Electronic Proceedings of WebALT 2006. First WebALT Conference and Exhibition. (2006)
11. Hamilton, G.: Javabeans specification. Sun Microsystems (1997)

12. Jameson, A.: Numerical uncertainty management in user and student modelling: An overview of systems and issues. User Modelling and User-adapted Interaction **5** (1996) 193–251
13. Crowley, R., Jukic, D., Medvedeva, O.: Slidetutor a model-tracing intelligent tutoring system for teaching microscopic diagnosis. In: Proceedings of the 11th International Conference on Artificial Intelligence in Education. (2003)
14. Melis, E., Andrs, E., Goguadze, G., Libbrecht, P., Pollet, M., Ullrich, C.: Activemath: a generic and adaptive web-based learning environment. International Journal of Artificial Intelligence in Education **12** (2001) 2001
15. Choksey, S.: Developing an affordable authoring tool for intelligent tutoring systems. Master's thesis, Worcester Polytechnic Institute (2004)
16. Hunn, C., Mavrikis, M.: Improving knowledge representation, tutoring, and authoring in a component-based ile. In: Intelligent Tutoring Systems. (2004) 827–829
17. Mavrikis, M., Hunn, C.: Interoperability issues in authoring interactive activities. In: Artificial Intelligence in Education - Supporting Learning through Intelligent and Socially Informed Technology. (2005) 869–871
18. Aleven, V., Sewall, J., McLaren, B., Koedinger, K.: Rapid authoring of intelligent tutors for real-world and experimental use. In Kinshuk, R., Koper, P., Kommers, P., Kirschner, D., Sampson, G., Didderen, W., eds.: Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies (ICALT 2006). (2006) 847–851
19. Matsuda, N., Cohen, W., Sewall, J., Lacerda, G., Koedinger, K.: Evaluating a simulated student using real students data for training and testing. In Conati, C., McCoy, K., Paliouras, G., eds.: Proceedings of the international conference on User Modeling (LNAI 4511), Springer (2007) 107–116
20. Aleven, V., McLaren, B., Sewall, J., Koedinger, K.: The cognitive tutor authoring tools (ctat): Preliminary evaluation of efficiency gains. In Ikeda, M., Ashley, K., Chan, T., eds.: Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006), Springer Verlag (2006) 61–70