Checking Conditions for Graph Based Collaborative Modeling Systems

Niels PINKWART
Kai HERRMANN
University Duisburg Essen

1. Cool Modes – a Graph Based Modeling Tool

Cooperative systems with a special focus on enabling sharing and commenting of resources and material are a prominent subject in the research area of computer technology in education. The representation mode of these shared resources plays an important role for the learning process [1]. One frequently used mode is a graph based notation in which a shared visual representation consists of objects and relations between them. With a focus on argumentation support, this representation mode has been used in a number of successful environments such as Belvedere [1] or gIBIS [2]. Belvedere also aimed to teach the students scientific argumentation – the system contains constraints that have to hold for the graph structure developed by the users. This approach of enriching cooperative graph based argumentation environments with rules and interpretation patterns can be integrated with pedagogic approaches like "cooperative discovery learning in science" that show the possible benefit of integrating computer based modeling methods and intelligent techniques with cooperative environments. As a number of modeling languages (like e.g. Petri Nets) also make use of a graph based notation, the technical integration step is possible very smoothly.

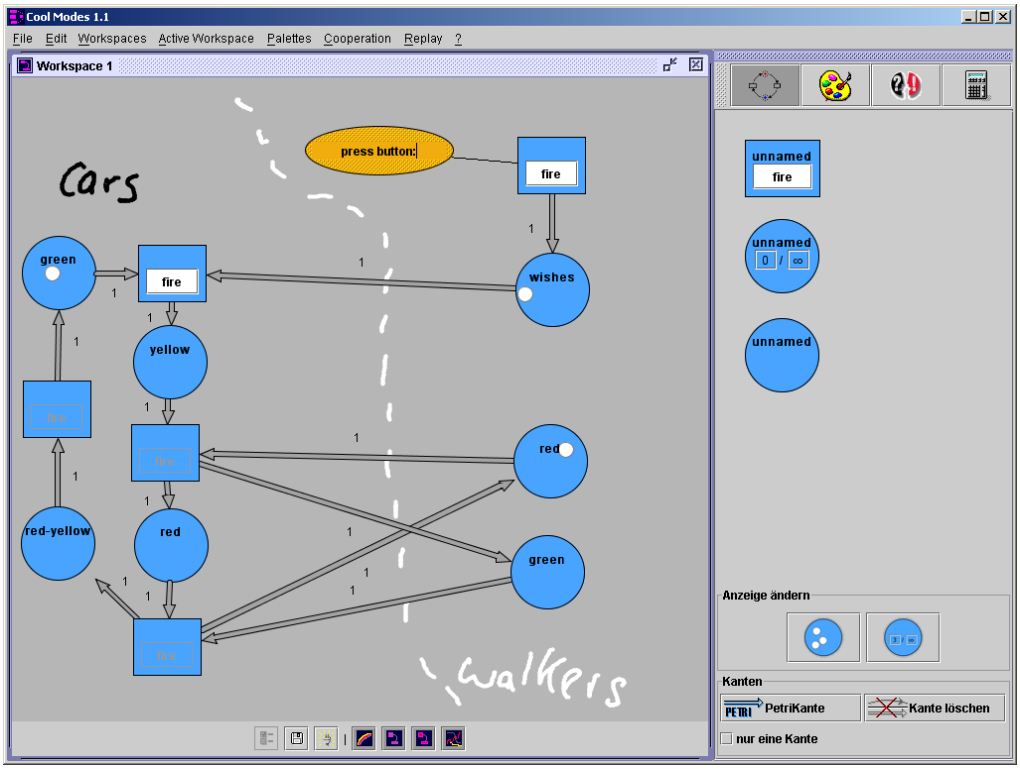

Figure 1. Cool Modes Screenshot

The event will start with a brief introduction to the environment Cool Modes (Collaborative Open Learning and MODEling System) that realizes this integration. We will demonstrate the argumentation support and modeling capabilities the system offers and outline the XML based plug-in mechanism [3,4]. Using a traffic light simulation with Petri Nets as an example (see Figure 1), we will give the participants the chance to explore and try out both the flexible co-operation features and the modeling tools that the system offers.

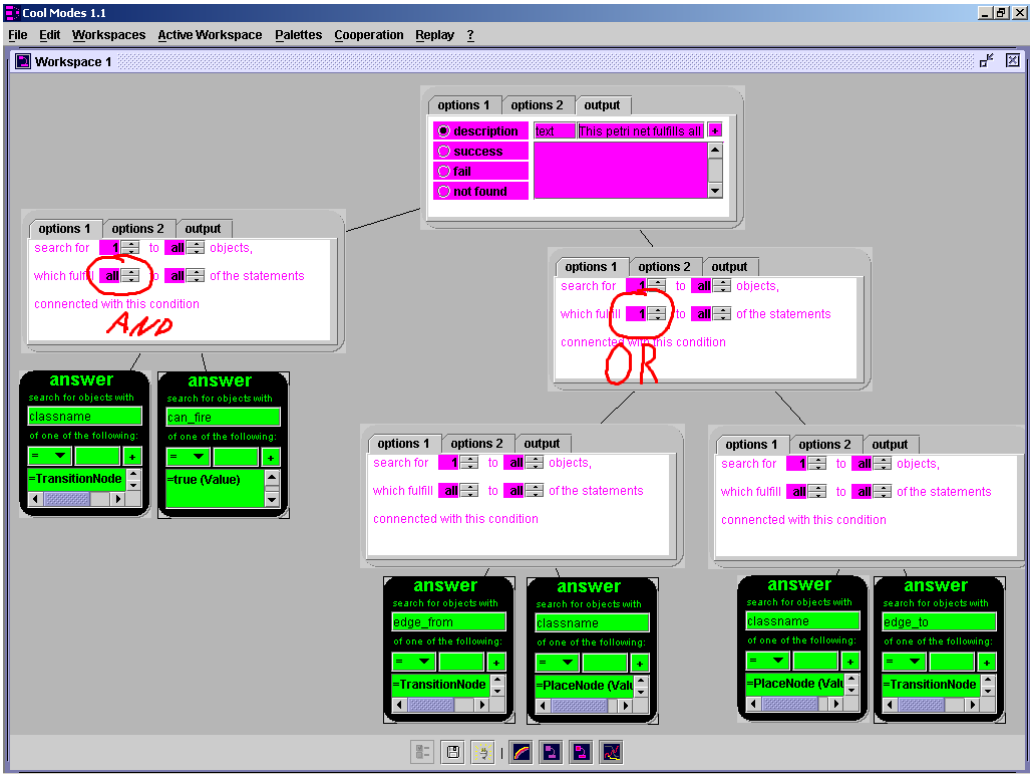

Figure 2. The checker plug-in.

2. The Checker Plug-In for Cool Modes

The second part of the event deals with the “Checker” plug-in for Cool Modes, a widely configurable analysis tool. It checks objects of visual languages with respect to different parameters. These parameters are user-defined and easily expandable; they can vary from the geometric orientation of components to semantic information like e.g. the number of marks in a Petri Net place. The Checker enables teachers or educational designers to precisely define conditions that a workspace content has to fulfill and to specify feedback for success or failure cases. A concrete example for a very simple condition that can be expressed with the checker, taken from the domain "Petri Nets":

IF	there is not at least one transition that can fire
THEN	give feedback “This net will never do anything. Do you really want this?”

The Checker for Cool Modes allows the construction of quite complex condition trees from such elementary conditions even for persons who not familiar the details of Boolean logic (see Figure 2). At the left side of the screen shot there is a graphical representation of the "at least one transition can fire" condition. At the lower right side of the screen there is a condition which postulates that every place should be connected to a transition. At the top of the screen there is another condition which combines the others.

The checker also offers direct feedback mechanisms: the user can test the produced conditions instantly. With these two features (no previous knowledge needed and simplicity

of use), the Checker is suitable especially for the use at school. Students can explore and test their solutions in a trial and error way with it, using condition trees provided e.g. by the teacher. By this, they are able to solve problems with a degree of complexity that would be out of reach otherwise. Like all other plug-ins for Cool Modes, the Checker appears in a graph-like representation, which opens up a much more intuitive access to the complex logic features than a textual representation like e.g. Prolog. Additionally, running together with Cool Modes, the Checker has got multi-user facilities. A condition built by member X can be used by member Y, e.g. to import it into another condition tree, or to discuss with X about it and to edit or refine it together with him (making use of the integrated hand writing annotation, chat and argumentation plug-ins of Cool Modes).

As mentioned, both Cool Modes and the Checker work with a great variety of domains. They can be used for very different things like, e.g., the simulation of System Dynamics models on the one hand or learning about Jewish rites on the other hand [3]. The second part of this interactive event will build upon the first part (collaborative construction of a traffic light simulation) and use the Petri Net plug-in. We will present how to express constraints within the Checker environment and then start a hands-on phase in which the participants can define and try out their own rules and advices using drag & drop, work together for an interactive collection and editing of rules, and test their rules with realistic examples. The following two tasks will be solved by the participants:

1. Build a condition structure that separates "good" from "bad" Petri Nets and detects common mistakes. (E.g., a “good” net should have at least one transition that can fire; otherwise it would not be very useful.). The resulting structures will typically be useful in introductory courses for Petri Nets.
2. Build a condition structure that decides whether a given Petri Net correctly simulates a traffic light. This is a task a teacher might have to face in preparing a school lesson in which the students then build the traffic light simulation. (cf. [5] for a more detailed description of a school scenario.)

The focus in both tasks, of course, is not getting new insights in Petri Nets, but using the interactive features of Cool Modes and the Checker within a concrete example.

3. Technical Requirements

For this event, we need networked computers (one computer for two participants) with the Java Runtime Environment 1.4.1 or higher installed. Wacom tablets or other facilities for pen-based input would be ideal as input devices. For the demonstration parts, we need a data projector (or, ideally, an electronic whiteboard).

References

[1] Suthers, D., Connelly, J., Lesgold, A., Paolucci, M., Toth, E., Toth, J., & Weiner, A. (2001) Representational and Advisory Guidance for Students Learning Scientific Inquiry. In Forbus, D. & Feltovich, P. (Eds.): Smart Machines in Education, Menlo Park: AAAI Press, pp 7-35.

[2] Conklin, J. & Begemann, M. L. (1987). gIBIS: A hypertext tool for team design deliberation. In *Proceedings of Hypertext'87* (pp. 247-251). Chapel Hill, NC (USA).

[3] Pinkwart, N., Hoppe, H.U., Bollen, L. & Fuhlrott, E. (2002). Group-oriented modeling tools with heterogeneous semantics. In *Proceedings of ITS 2002* (eds. Cerri, Gouardères & Paraguacu), pp. 21-30. Springer, Berlin.

[4] Pinkwart, N. (2003). A Plug-In Architecture for Graph Based Collaborative Modeling Systems. To appear in *Proceedings of the AIED 2003*.

[5] Herrmann, K., Hoppe, U., Pinkwart, N. (2003). A Checking Mechanism for Visual Language Environments. To appear in *Proceedings of the AIED 2003*.