

# Converting Conceptualizations into Executables: Commentary on Web-based Adaptive Education and Collaborative Problem Solving

JOSEPH M. SCANDURA<sup>1\*</sup>

*MERGE Research Institute*

High-level descriptions for web-based adaptive education and collaborative problem solving proposed by Kennedy et al and by Eccles and Groth (this issue) provide useful starting points. The challenge is to build on and to convert such analyses into executable systems. This commentary shows how the proposed frameworks may be automated by representing knowledge in terms of Abstract Syntax Trees (ASTs) and associated Structural Learning Theory (SLT). A wide range of questions can be asked both at conceptually high levels of abstraction as well as regarding implementation.

ASTs provide an intuitive basis for accomplishing this goal by making it possible to represent knowledge at multiple levels of abstraction. The relatively complex inferential nature of associated tutorial logic also is shown to have important implications for how web-based adaptive education is actually implemented. Supporting collaborating learners will require representing the knowledge associated with multiple learners, and specifying inferences between multiple (or shared) learner models as learning progresses. In cooperative problem solving, problem solvers receive the same or complementary problems. In adversarial situations, problem solvers, human or automated, effectively have different problems (e.g., different goals), as well as different rule sets (i.e., knowledge).

*Keywords: Abstract Syntax Trees, collaborative problem solving, web-based tutoring, adaptive tutoring.*

---

\*Corresponding author: [scandura@scandura.com](mailto:scandura@scandura.com)

<sup>1</sup>As senior advisor I have overall editorial responsibility but all co-editors are regularly encouraged to comment in print where their expertise may shed light on an important issue. Furthermore, any board member or reader is free to submit informed commentary for publication. Authors of the articles in question also have the right to publish reactions. Only one rule applies: Each commentary, or reaction to a commentary, etc. may not exceed half the length of the immediately previous commentary or reaction.

In addition to publishing articles that advance the field, a major and unique goal of TICL is to promote cross-disciplinary awareness and dialog on fundamental issues, something that is frequently absent in our field. The aim is to improve understanding not by committee or consensus but through the power of publicly stated defensible ideas.

My comments on two articles in this issue, edited by Mike Spector in collaboration with Co-Editor Norbert Seel, are directed toward this end. These articles provide different but excellent vehicles for illustrating a deeply held belief (based many years of research) that the single most important problem in instructional technology is the disconnect between high level conceptualizing and the realization of those concepts as working systems.

As summarized in my introduction to TICL's recent 2005 Highlight issue, both top-down and bottom-up approaches face major hurdles in converting theory to working technology. **“Top-down efforts must do a better job of converting high-level conceptualizations into working systems. Bottom-up methods need the reverse – insuring that working systems accurately reflect coherent abstractions.”**

In this regard, Ian Kennedy et al (this issue) and David Eccles and Paul Groth (this issue) have done outstanding jobs in calling attention to two critical issues: web-based education and collaborative problem solving respectively. My goal in this commentary is to outline one way in which the frameworks they propose might be automated. I focus on a single well-integrated perspective, albeit one that I have been working on for some time. Other well reasoned commentary (on either the above articles or others), or reaction to this commentary, is encouraged and welcome in TICL.

**Kennedy et al** (this issue) call for the separation of content from delivery and an emphasis on adaptive instruction in web-based instruction focused on individuals as opposed to group characteristics.<sup>2</sup> They also recognize the need for converting concepts into executables. They rightly note that the definitions used by psychologists and pedagogy experts tend to be vague while IT developers work with concrete terms like variables (i.e., data structures) and processes.

Kennedy et al propose a model not unlike many others but with more emphasis on separation of knowledge to be acquired, logic governing how that knowledge is to be conveyed to the student, the student interface via which tutors and learners interact and what the system (tutor) knows about the

---

<sup>2</sup>The latter incidentally provides a far more effective way to address social concerns than harping on political solutions.

learners current state of knowledge. Based on their analysis, they outline and describe a simple system based on the following modules: a) a module representing the knowledge to be acquired, b) a tutoring module for delivering instruction on the content, c) a student interface via which the learner and tutoring model interact and d) a student model representing what the system currently knows about what the learner knows.

Kennedy et al are silent on many details. AuthorIT (Scandura, 2005), an operational authoring and delivery system with which I am familiar, provides a convenient basis for comparison. Whereas Kennedy et al represent Student Knowledge as an overlay on the Knowledge Base, for example, they say little or nothing about essential next steps. These next steps include (but are not limited to):

- a) specifying how knowledge is to be represented and
- b) how to define the Student Knowledge overlay in terms of observable (test) behavior.

Both issues have played a major role in the Structural Learning Theory (SLT) for some time (e.g., Scandura, 1971). Only recently (Scandura, 2001), however, have Abstract Syntax Trees (ASTs) played a central role in how knowledge is represented in SLT, or in instructional systems generally. As outlined in Scandura (2006), ASTs make it possible to represent any structure or process at multiple levels of refinement, from the highest levels of abstraction to the lowest useful level of detail (see Scandura, 2003 for formal definitions). ASTs have also played a central role in AuthorIT's construction.

With a few notable differences, the ideas expressed in the Kennedy et al article find concrete realization in AuthorIT (Scandura, 2005). It is instructive in this context to consider relationships between Kennedy et al's Figure 1 (this volume) and Figure 1 in Scandura (2005), which is repeated below for easy reference.

Kennedy et al's *Domain Model* clearly corresponds to **Content/Knowledge Representation** in AuthorIT (Scandura, 2005). Similarly, Kennedy et al's *Student Interface* Module corresponds to AuthorIT's **Blackboard Interface** in Figure 1 above and Kennedy's *Picture of the Learner* to the **Learner**. What are divided by Kennedy et al into *Tutoring* and *Student Models*, however, are both handled by **TutorIT**. Why? The reason is that assessment (what is actually going on in Kennedy et al's Student model) and instruction are intimately related in tutoring. In order to know what to teach at any given point in time the Tutor must know what the learner knows. Conversely, after teaching something the tutor can not know with any certainty whether the student has

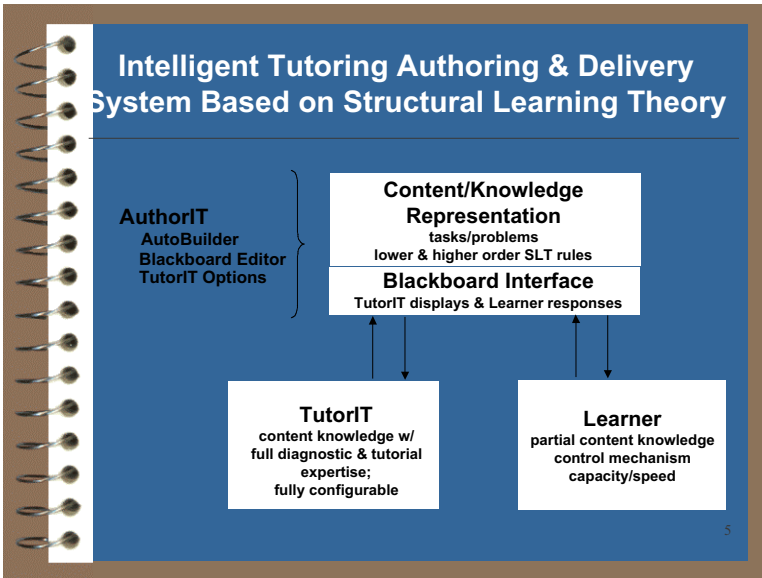


FIGURE 1  
Composition & role of AuthorIT & TutorIT in authoring & delivering intelligent tutoring via the Blackboard Interface.

actually learned unless he or she is independently tested. While all testing can be done independently, this is much less efficient from a learning point of view than when the two are tightly integrated. Communication between the two must be intimate and on-going. The close relationship between testing and tutoring, along with inferences the tutor can safely make about what the learner knows, is detailed in Scandura (2005) in the subsections on Learner Model (LM), Diagnosis and Instruction under Diagnostic and Instructional Logic (pages 210-213).

In a nutshell, the Learner Model (LM) is defined as an overlay on ASTs representing the knowledge to be acquired. Each node in an AST has one of three states: known (+), unknown (-) and to-be-determined (?). Because ASTs represent the same content at multiple levels of abstraction, the actual number required for diagnostic and instructional purposes is greatly reduced. Success on a sub-problem corresponding to a non-terminal node (e.g., borrowing), for example, necessarily implies success on all simpler sub-problems. "Knowing" (i.e., marking '+') a non-terminal node means by definition that all lower level nodes in the AST also must be +. Conversely,

all nodes higher in an AST than a node marked “-“ also must be marked “-“. Other than by chance (mutually compensating errors), it is impossible (using the same rule) to get a harder problem correct unless one can solve all of the simpler sub-problems of which it is composed. These and related ideas dramatically improve tutor efficiency (patent pending).

The LM for a given rule is constructed by assigning “+”, “-“ or “?” to each node in the corresponding (e.g., procedural) AST — insuring in the process that nodes in each sub-tree defined by each non-terminal “+” node necessarily are also marked “+” and that all nodes higher than each node marked “-“ in the AST necessarily are also marked “-“. All diagnostic and instructional decisions depend directly on the current state of the ASTs defining the learner’s knowledge.

Given a problem, TutorIT selects nodes in the corresponding AST in accordance with options set by the author. Given a sub-problem corresponding to the selected node in the procedural AST, the purpose of diagnosis is to determine the current state of the learner’s knowledge. Diagnosis concentrates on nodes, marked “?”, whose correct state is yet to be determined. When ASTs are refined to the atomic level, empirical research demonstrates that a single test item for each operation node tested is sufficient (e.g., Scandura, 1971, 1973, 1977).

Ideally, one only wants to provide instruction when and where it is needed. Providing instruction when it is not needed both wastes time and can reduce learner interest. Providing information (e.g., hints) before a learner has mastered necessary prerequisites, on the other hand, can lead to learner frustration and giving up (e.g., Scandura et al, 1969). By default, instruction is provided only when the tutor knows that the learner does NOT know the corresponding material. Unless overridden by the author, TutorIT provides instruction only on nodes designated by “-“. Once instruction corresponding to a “-“ node has been presented to the learner, that node is marked “?” because the tutor cannot know for sure whether or not the learner has mastered the material. This can only be assured after subsequent testing.

The relatively complex inferential nature of the above tutorial logic has important implications for how web-based adaptive education is actually implemented. Many purists believe that learners should be able to obtain instruction directly from the web – with everything running on the server, with no local execution or downloads. In fact, it does not take much analysis to “blow holes” in purist thinking. Any web-based tutorial worth its salt requires media support of one sort or another: Flash Reader and Microsoft

Sound Recorder and Media Player are among the most pervasive. These reside on the local computer not the web.

Media aside, the reason for preferring web-based delivery is to avoid the complication of having to download and/or install software locally. In this context, Kennedy et al rightly suggest the desirability of separating tutoring logic from content. The question of how to accomplish this in a generic way remains. Most solutions to this problem (going back to Gagne's conditions of Learning in the 1960s) are based on a philosophy of divide and conquer. Tutorial shells are developed to support various kinds of content – facts, concepts, procedural, declarative, etc. As required by pragmatics (speed, flexibility and the like), one can imagine installing such shells locally and either downloading on demand and/or accessing content across the web as needed. Content involving multiple kinds of learning (most real learning), however, would require custom treatment.

TutorIT changes this dynamic. TutorIT is designed as a general-purpose tutor that currently supports all well-defined, even simple inferential kinds of content (i.e., forward chaining used in many expert systems). It can be downloaded and installed locally once and used in the same way as and in combination with Flash Reader and other media support files. Authors also can configure TutorIT to support alternative pedagogies. Learners can select from among available delivery modes without further downloading. Content can either be installed at the beginning of a lesson (as typically done in web-based delivery) or in advance. In either case, TutorIT's relatively sophisticated logic is handled locally. We had considered putting this logic on a server initially, but it rapidly became clear that doing so would not scale well in large applications.

While this is not the place for detailed analysis of what is inherently a complex problem, it should be noted as that technical as well as conceptual issues must be considered. The ideal is for the conceptual framework to inform the technical and vice versa, the technical to reflect the conceptual.

Although less immediately obvious, the concrete executable and traceable nature of the AuthorIT system (which includes TutorIT) helps address this need. Not only can AuthorIT be described at a high level of abstraction but each level of abstraction in its implementation is documented in hierarchical ASTs. While much is unpublished, each step in the construction of AuthorIT (and TutorIT) can be refined in the same manner as to be acquired knowledge (documented in Scandura, 2005). In general, any idea or process can be made as precise as desired in Softbuilder's AST-based Flexforms (see [www.scandura.com](http://www.scandura.com)). Scandura's

Softbuilder system (see [www.scandura.com](http://www.scandura.com)) supports indefinite refinement in this context. AutoBuilder currently automates many of the steps involved in simpler applications — those that do not involve dynamic input or output structures (i.e., higher order knowledge).

**Eccles and Groth** (this issue) address the equally important and very difficult job of shedding light on collaborative problem solving involving both human and technological agents. They draw on an impressive array of evidence pertaining to human and animal collaboration highlighting differences in the way technological agents are currently implemented. They identify a number of key characteristics separating cooperation in animal and human species that are not shared by technological agents: a) an effective system of communication and b) an ability to update a shared mental model. They conclude that a problem solving system involving humans and technological agents will be effective to the extent that the technological agents are able to share a common mental model. Such a model among other things must include the current status of the problem, which solution strategies are needed at each point in time and which agent will undertake each. The importance of each is highlighted in a series of interesting applications ranging from science, combat systems in the military and medicine.

My proposals regarding Eccles and Groth are more futuristic. As described in Scandura (2005, p. 223), representing knowledge associated with multiple learners (as needed in collaborative learning) presents significant yet seemingly achievable challenges. As described above, what the tutor knows about the learner (model) is currently represented in TutorIT as an overlay on nodes in each AST. In addition to the AST structure itself, TutorIT uses knowledge states associated with AST nodes to make inferences as to knowledge associated with other knowledge states.

To date, the Structural Learning Theory (SLT) has only been applied to individuals. The basic theory, however, appears equally applicable to collaborative learning. Extending TutorIT to support collaborating learners will require representing the knowledge associated with multiple learners, and specifying inferences between the various (or shared) learner models as learning progresses. The same or similar complex domains may have to be analyzed from multiple perspectives, yielding multiple knowledge representations (AST-based rule sets in the case of SLT).

Implementing such knowledge in AuthorIT might be accomplished, for example, by introducing arrays or linked lists to represent the knowledge associated with multiple learners. Implementing inferences will require

introducing logic specifying how the performance of individual learners effects what the tutor knows or can infer about the others. For example, one learner solving a problem (that may stump others) serves as instruction effecting inferences TutorIT is willing to make about other learners. For example, one person in a group solving a problem does not imply that others have actually learned as a result. Accordingly, the corresponding state for other learners might reasonably be set to undetermined (i.e., '?'). Conceptual issues, of course, are only one part of the equation. Technical issues pertaining to distribution across the web will also need to be considered where learners are scattered at different locations.

Other more general issues go beyond to the need for executable detail. The above scenario assumes one automated tutor and multiple human learners. Other possibilities include multiple automated problem solvers under the control of a single human, multiple automated agents and multiple humans, etc. Eccles and Groth focus on cooperating problem solvers. In this regard, problem solvers may work either in cooperation on the same basic problem with the same overall goal, and/or as adversaries with their own competing goals. In cooperative situations, a minimum of one person must solve the given problem, and all must agree on the problem Goal. Otherwise, the situation becomes more complicated with successful problem solvers (in the absence of strict hierarchical control) having to convince others. In adversarial situations, each problem solver must in addition be able to guess or infer what the adversary knows (i.e., what their rules set is). TutorIT's current diagnostic capabilities enable it to diagnose/infer what an independent agent (e.g., learner) does and does not know. Similar diagnostic capabilities may be extendable to multiple agents and could play an essential role in the process.

In cooperative problem solving, problem solvers receive the same or complementary problems. Their respective knowledge (e.g., AST rule sets) may represent differing knowledge however. Indeed, the number of possibilities is large. Each human and/or agent may have shared goals, and the same or similar rules sets (e.g., as in a team of similarly trained scientists working on a common problem). Alternatively, a single individual may have a team of automated agents to be called upon as needed. Where the automated agents need to interact, implementation will require each to serve as an "event handler" which receives input events from other agents (or the human) or sends output events to others. Event handlers are commonplace in interacting systems. The major questions in each case are not so much how to implement them but rather the logic governing internal operations as well as their

interactions. Eccles reference to changing or specialized demands in cooperating systems, for example, could be handled by coordinating event handlers. (Within single organisms, allocating resources would be handled by higher order selections – selection rules — among alternative sub-goals or rules in any given situation, e.g., see Scandura, 2001, p. 283.) This is the kind of information that needs to be made explicit before one can seriously talk about the benefits and limitations of such systems.

Accommodating such knowledge also would require enhancing AuthorIT's Blackboard Interface, enabling it to support inputs and outputs from multiple problem solvers, human and/or automated (i.e., technological). Currently, the Blackboard Interface can support only one automated system (i.e., TutorIT) and one human interacting with TutorIT. Responses from any one problem solver would necessarily have to be made available to the others, enabling them to build on and/or react to visible changes.

The situation is further complicated in adversarial situations. Problem solvers, human or automated would effectively have different problems (e.g., different goals), as well as different rule sets (i.e., knowledge). In adversarial problem solving, the Blackboard Interface would not only have to accommodate multiple problem solvers, with differing rule sets and identical or differing goals, but require separate interfaces for the adversaries. The goal of each adversary is to assess (i.e., understand) what the (other) adversary knows and to act accordingly.

## CONCLUSIONS

In this short commentary, I have only scratched the surface of what is a very complex set of problems. A wide range of questions can be asked – both at conceptually high levels of abstraction as well as regarding implementation. In this respect, the high level descriptions provided by Kennedy et al and by Eccles and Groth provide only starting points. They frame important issues and provide direction. The real challenge — in our field generally — is to build on and to convert such analyses into executable systems.

---

The Softbuilder system Scandura.com developed a few years back as part of an IBM-led Advanced technology Program project was designed specifically with complex interactions among event handlers in mind. As noted by the program manager at the time, Softbuilder provides for the first time a way to develop and test distributed systems from the top down (patent pending). See [www.scandura.com](http://www.scandura.com) for more information.

I urge TICL researchers to put more emphasis on filling the wide gap between high-level conceptualizations and working systems. Otherwise, it will remain difficult if not impossible to compare and evaluate alternatives. As Norbert Seel remarked at a TICL conference a few years back, although motivating theories may differ, it is often hard to distinguish the actual systems created (in particular applications). Intuitive judgments made during implementation typically have as much if not greater effect on resulting systems than the model or theory one starts with.

I clearly am proposing ASTs (and SLT more generally) as one way to address these issues. At a minimum, ASTs provide a rigorous and executable foundation on which to build. I further suggest that this gap be filled in a traceable, repeatable fashion. If our broad eclectic field is to mature, it must become more cumulative in nature — not just in the technologies employed but also the concepts guiding actual development. To accomplish this, core ideas need to be systematically refined, tested and corrected as necessary. Structural Analysis (e.g., Scandura, 2003, 2005) represents one systematic method for constructing knowledge representations (i.e., AST-based rules). It can be applied to any set of ideas and/or processes, making it possible to represent them in whatever degree of precision may be desired.

As a final cautionary note I call attention to (but do not discuss here) the opposite problem cited in the introduction (cf. Scandura, 2006) — the importance in of insuring that bottom-up technical solutions accurately reflect defensible and coherent abstractions. Research in this area, including well thought out alternatives and carefully constructed challenges, are all to be encouraged in TICL.

## REFERENCES

- Scandura, J.M. Structural Learning Theory: Current Status and Recent Developments. *Instructional Science*, 2001, 29, 4, 311-336. Also in Scandura, J.M. Structural Learning Theory in the Year 2000. *Journal of Structural Learning and Intelligent Systems* (special monograph), 2001, 14, 4, 271-306, including Appendix A: A Flow Chart overview of research in SLT from early 1960s to 2001 and Appendix B: Relationships to Formal Work in Computer Science.)
- Scandura, J.M. Domain Specific Structural Analysis for Intelligent Tutoring Systems: Automatable Representation of Declarative, Procedural and Model-Based Knowledge With Relationships to Software Engineering. *Technology, Instruction, Cognition & Learning (TICL)*, 2003, 1, 1, 7-57.
- Scandura, J. M. AuthorIT: Breakthrough in Authoring Adaptive and Configurable Tutoring Systems? *Technology, Instruction, Cognition & Learning (TICL)*, 2005, 2, 185-230.
- Scandura, J. M. AST Infrastructure in Problem Solving Research. *Technology, Instruction, Cognition & Learning (TICL)*, 2006, 1-13.