In: J. Greer (ed.) Proceedings of AI-ED'95, 7th World Conference on Artificial Intelligence in Education, Washington, DC, 16-19 August 1995, AACE, pp. 1-8.

Intelligent learning environments for programming: The case for integration and adaptation

Peter Brusilovsky

International Center for Scientific and Technical Information, Moscow, Russia
(Currently at the Department of Psychology, University of Trier, D-54286 Trier, Germany)
E-mail: plb@plb.icsti.su

Abstract. The area of intelligent tutoring systems for programming is one of the most well-investigated topics in the AI-Ed literature. A number of powerful techniques to help novice learners have been suggested, developed, and described in hundreds of papers. However the "real world" situation with classroom applications of these systems is no better than it was 20 years ago. We claim that more attention should be paid to the problem of integrating existing techniques and tools into more versatile learning environments. In this paper we discuss the problems of integrated intelligent learning environments (ILE) for programming and review existing research that forms the background for future research in this area. In particular, we will review our work on a student model-centered architecture for ILE.

Introduction

From the very early days of intelligent tutoring systems (ITS), teaching programming was one of the favorite application areas for ITS designers. A number of powerful techniques to help novice learners and dozens of real ITS and ILE have been developed and described in hundreds of papers. Teaching programming can be considered as an example of successful application of AI in education. An attempt to build a taxonomy of existing ITS and ILE for programming reveals that existing systems cover almost all possible dimensions of classification. Even if we restrict ourselves with the most recent and best known systems (i.e. systems reported in several papers from 1992 to 1995) we can find that these systems support:

- learning a wide range of programming languages including procedural languages (Fix & Wiedenbeck, 1992; Ueno, 1994), functional languages (Corbett & Anderson, 1993) object-oriented languages (Chee, Tan & Chan, 1993), logic programming language(Gegg-Harrison, 1992), mini-languages (Ramadhan & du Boulay, 1993; Lelouche & Dion, 1994), graphical languages (Möbus, Thole & Schröder, 1993), parallel programming languages (Herzog, 1992),and some specialized languages like COBOL (McKendree, Radlinski & Atwood, 1992),
- acquisition of knowledge and skills required to perform all main steps of real programming activity: problem analysis (Rozinajova & Navrat, 1993), specifications (Möbus et al., 1993; Verdejo, Fernandez & Urretavizcaya, 1993), planning (Fix & Wiedenbeck, 1992; Möbus et al., 1993); coding (Corbett & Anderson, 1993; Singley, Carrol & Alpert, 1993); testing and debugging (Brna, Hernandes & Pain, 1993);
- several activities normally performed by a human tutor in the teaching process: adaptively guiding the student through the curriculum (Brusilovsky, 1992b), presenting the student with relevant

examples (Weber, 1995) and problems (Brusilovsky, 1992b; Gegg-Harrison, 1992) supporting the student in the process of problem solving (Corbett & Anderson, 1993), intelligent debugging (Brette, 1994; Ueno, 1994) of student's solutions, and presenting the student with a correct solution (Chee et al., 1993);
- various learning and problem-solving activities of the student: browsing various sources of knowledge (Fischer et al., 1992; Linn, 1992), playing with program examples (Weber, 1995), program design and coding (Hohmann, Guzdial & Soloway, 1992; Price, McCalla & Greer, 1994), visual executing and debugging (Eisenstadt, Price & Domingue, 1993; Merrill et al., 1992).

A novice researcher who analyses the existing literature on ITS for programming looking for a place for a really new contribution might decide that there is nothing new to do in this area. All possible directions of work have already been investigated by several researchers. For any part of the educational process where a teacher or a student may need computer support, one can find several papers reporting relevant support techniques and tools to implement them. Unfortunately, the situation looks completely different for a teacher who is interested not in reading about all these intelligent tools but rather in using them with students. After 20 years of research there are very few ITS or ILE used regularly for teaching programming in a real classroom, and most of them can not be used outside the university where they were created. On the other hand, the only computer-based educational tools used in universities and schools to teach computer programming are non-intelligent CAI, novice programming environments, and hypermedia systems. Is it a success or failure of intelligent technology?

Why ILE for programming are not used in the classroom

Let us try to look at the current situation from a viewpoint of a classroom teacher. At first, to be useful, systems have to support some reasonable part of teacher's or/and student's work. At the same time, various teacher and student activities and various parts of the course are very well-supported by existing ITS and ILE, but each single system supports a very small part of the course and very few activities. In most cases the benefit from using one more system is not worth the burden of installing and learning it. Moreover, very few existing ILE have adequate programming environment components (i.e. they need at least a novice-oriented editor and an interpreter) and usually nothing is done to support the work of "environmentless" systems with external programming environments. The same is true for internal and external hypermedia components (at least an on-line language reference is important). Second, there are many possible ways of teaching programming, and, to be useful, a system have to support a way of teaching adopted in a particular classroom. At the same time, most existing systems are strictly oriented to usage in their home universities. Such systems can not be used "as is" to support different ways of teaching or different course structures.

To explain further, let us look at the situation from the designer's viewpoint. Modern intelligent and non-intelligent systems for teaching programming are quite complex. Each component requires several person-years of research, and, as we have seen, to be really useful a system needs to have several components. What can be done in this situation by a single young researcher (perhaps a PhD student) who wishes to make both visible and practically useful contributions into the field? One way is to concentrate efforts and develop one original ITS component which demonstrates some new idea, but in the current situation may not be used alone. Another way is to split the effort among several components, trying to produce a useful system and, in the end, producing only a toy example. Only if the researcher is lucky enough to be a part of a larger research team is there a chance to design something interesting and make it work within the context of a bigger ITS. Teams, however, have the same problem because team resources are usually limited. Teams usually try to concentrate efforts on the intelligent components of ILE, and do not like to waste effort building components outside the team's research interests, eveen if these components might be important for the user.

Integration can solve the problems

The problems of ILE users and designers can be solved if we can find a way to integrate the functionality of different ITS, ILE, programming environments and hypermedia systems. A minimum requirement here is the possibility to integrate two different systems, for example an ITS and a programming environment. What would be most desirable is the possibility of extracting a particular component from a system and using it together with another system made by different developers. In that case, a teacher who is looking for a computer-based support for a particular programming course could integrate several existing intelligent and non-intelligent components which could together support the desired parts of the course and desired range of activities. At the same time, single developers and teams could concentrate their efforts on the components which are within their scope of expertise. They do not need to waste time developing other components just to make the system complete and useful, because the missing component can be borrowed from other systems.

Unfortunately, existing intelligent and non-intelligent systems are not designed to support such an integration. Each system is usually designed as a single whole and can not be used flexibly. One can not naturally integrate an external component (for example, a commercial programming environment or an intelligent debugger made locally) into another system. A researcher can not readily remove or replace an inappropriate component which does not fit with a particular way of teaching. If the system has a component that is really liked and needed, it is often hard to extract and re-use this component in another system. One can only use the system "as is".

As a result, many developers spend years of work to create incomplete systems which can not be used by classroom teachers. Many of these incomplete systems have original components implementing interesting ideas, but since these components can not be re-used, these ideas never come to practice and usually die together with the system when the developer's interests change. To stop this process we need to develop a new integration-oriented architecture for ILE. This architecture should encourage design of an ILE as an integration of re-usable components. In that case, each newly designed ILE will be a real contribution to the domain. The ideas and efforts of the developers will never more be lost, because all original parts of the system can be used by many other people. Good integration-oriented architecture is very important and can be a key to success. Good examples here are Unix commands and World-Wide Web nodes. In both cases, to achieve a particular goal the developer needs only to produce essential new commands or nodes integrating them (by Unix shell or hyperlinks) with the work of predecessors. In turn, these new commands or nodes can be re-used by future developers.

The rest of the paper concerns the problems of integrated ILE, i.e. ILE which are designed as an integration of re-usable components. We try to formulate some requirements to integrated ILE and design a prototype of an integration-oriented architecture for ILE. Then we list directions for research on integration-oriented architecture and review some recent work which forms a background for future research on integration.

Integration-oriented architecture for ILE

The main requirement for an integrated ILE is that an integrated system should not be just an aggregation but a real integration of its components. In our previous work (Brusilovsky, Pesin & Zyryanov, 1993) we considered two aspects of real integration. First, one component should be able to use the capabilities of another component, as well as exchange or share data with it. For example, a program checker needs to use a language interpreter to run the program being checked on the set of tests. Second, the results of students' work with any of the components during the session should be taken into account by other components to adapt their performance to the knowledge level and personal features of the particular

student. For example, the student may learn something new while working with a hypermedia component. The tutoring component should take this into account to avoid re-teaching of already-learned concepts. We refer to the first aspect as technical integration and to the second as conceptual integration.

To achieve conceptual integration we need to have some central student model which should collect and integrate the information about the student from all system components. The central model can be used by intelligent and non-intelligent components to adapt their work to the current state of the student. These considerations led us draft the architecture of ILE as a set of independent reusable modules connected to the central student model (fig. 1a). We can distinguish two kinds of modules: knowledge-based modules or agents (for example, a curriculum sequencing module) which contain various kinds of knowledge and provide the system with its intelligence, and interface modules, or tools which directly interact with the user. (We can consider the modules to have both a knowledge base and an interface, but separating a knowledge-based module from interface features makes it easier to port them to another computer platform and increase reusability). Both kinds of modules should be able to interact with other modules, to supply the information to the central student model, and to use this model to adapt performance.

To elaborate the draft architecture, we have to consider the requirements of flexibility and reusability.:

- the possibility to easyly re-use a particular component in another system
- the possibility to easily integrate a new component into a system (i.e. easyly connect it with the student model and other components)
- the possibility to easily replace a particular component (including the student model!) with a similar one with minimal changes to the components interacting with it.

Here by making something easy we mean tuning a component or a connection to a new context, which should be available to an advanced user and should not include any kind of re-programming.

These considerations set additional requirements for the interface between system components and between a component and a student model, as well as for these components themselves. To make a component or a student model replaceable, we need to design some standard conceptual-level protocols of interaction between various components. To make a component highly re-usable, we need to make it as context-independent, as possible. For an interface component it means designing highly adaptable components with various possibilities to adapt the interface to the application context. For a knowledge-based component it means designing a kind of generic component (or a shell) to perform a particular goal, for example, a generic problem sequencing component. To use a generic component in a very different context (for example, with a different part of the course) a designer can fill it with context-oriented knowledge. To use it in a similar context (for example, with a different way of teaching the same course content) a user can tune this knowledge.

If we can find an architectural solution satisfying all the above requirements and if it is simple and elegant enough to be used by various developers, we have a chance to establish standard integration-oriented architecture for ILE and solve the current set of problems with ILE. Further research efforts are required to achieve this goal. Such research should strive to answer the following questions:

Interaction
: How does one make a flexible interface between components which lets the components exchange information and control each other? How can a component interact with "dumb" components, for example, with a professional programming environment that does not support any standard architecture?

Student modeling
: How does one make an interface between a component and a student model that lets a component

supply information to the model and to use it for adaptation? What kind of student model is required?

Shells

What kinds of knowledge-based generic modules can be useful and how should they be designed?

Adaptive interface

How does one make standard interface components of an ILE for programming (for example, a structure editor) adaptive, using the student model? How does one make it adaptable to the application context?

Existing work provides a background for further research

A good background for the research on integration-oriented architectures has already been formed by a number of existing projects. The aspects of inter-component communication were addressed by the work on blackboard architectures for ITS (McCalla, Greer & Scent, 1990) and more recently by the original work of members of the French-speaking research community on multi-agent architectures for ITS and ILE (Futtersack & Labat, 1992; Girard, Gauthier & Levesque, 1992). A multi-agent ILE is a set of independent components, such as a problem-solver, environment, or instructional planer, which may have local knowledge and communicate by exchanging messages. The research on multi-agent architectures also investigate the problems of generic agents and centralized student modeling (Néhémie, 1992).

Very recent research (Ritter & Koedinger, 1995; van Rosmalen, 1994) considers more technical aspects of inter-component integration. These papers suggest an integration-oriented architecture where the components can be different separate applications, including "dumb" commercial packages. An important contribution of this work is the centralized architecture for flexible communication between components based on some inter-application protocol such as Apple Events or DDE. Two aspects of this architecture are an application-independent semantic-level communication language and a control center which collects and delivers all inter-component messages.

The problems of generic knowledge-based components are addressed by the works on various ITS shells. Good examples here are various components for the control of learning based on pedagogical knowledge (Major & Reichgelt, 1992; Murray & Woolf, 1992; Van Marcke, 1992; Vassileva, 1990; Vivet, 1988). A less investigated area with only one good example (Anderson & Pelletier, 1991) is problem solving support shells based on domain expertise.

The topic of centralized student modeling is addressed by research on student modeling and user modeling shells (Kay, 1995; Kobsa & Pohl, 1995; Paiva & Self, 1994). All these shells are designed to collect information about the user from different sources and to serve queries about the user from different applications. It should be noted that most of this work, as well as work on adaptive interfaces (Schneider-Hufschmidt, Kühme & Malinowski, 1993) has been done outside the traditional ITS domain.

Our group at the Moscow State University contributed to all four directions of research. Originally, we investigated technical aspects of integration and the problem of generic components. We worked to design different components of ILE as separate UNIX processes communicating by "pipes". In particular, we designed two "generic" components for problem sequencing (Brusilovsky, 1992a) and student modeling and "re-used" them in two different systems. Later we had to change the platform from UNIX to single-process DOS what forced us to concentrate more on conceptual aspects of integration. In the next sections we report our current work on the student-model centered architecture and on the design of various adaptive components for several ILE for programming.

Towards comprehensive adaptation and conceptual integration

Our research at the Moscow State University is centered around two problems of creating integrated ILE: the problem of comprehensive adaptation and the problem of conceptual integration. These propblems are closely interrelated. An ILE is *comprehensively adaptive* if all its components can adapt dynamically to the particular student. Most ITS and tutoring components of ILE can adapt their work (tutoring) to a given student, however very few programming environments and hypermedia components can do that. It was one of our goals to build adaptive programming environments and hypermedia components of ILE (Brusilovsky, 1993). What we made adaptive in our programming environments are a visual interpreter and a language-oriented editor. The visual interpreter of ITEM/IP (Brusilovsky, 1992b) uses the student's current knowledge level to provide adaptive error handling and adaptive visualization. In a newer system ITEM/IP-II we added adaptive explanatory visualization (Brusilovsky, 1994b) and a simple adaptive structure editor. Our most recent work is devoted to the adaptive hypermedia component for ILE. We have tried an adaptive presentation technique in ITEM/IP and examined the problems of adaptive navigation support in ISIS-Tutor (Brusilovsky, 1994a; Brusilovsky et al., 1993).

Conceptual integration implies that the results of students' work with any of the components during a session should be taken into account by other components to adapt their performance to the changed knowledge level of the particular student. Our approach to conceptual integration is to have a single representation of the student's knowledge in the central student model. This model should collect all the information about the student and should be accessible by all components of the ILE. After trying some solutions based on a traditional overlay student model we have designed an advanced student model-centered architecture for ILE (Brusilovsky, 1994c) which can be considered as an advancement of the integration-oriented architecture in terms of the aspects of communication between components and the central student model (fig 1b).

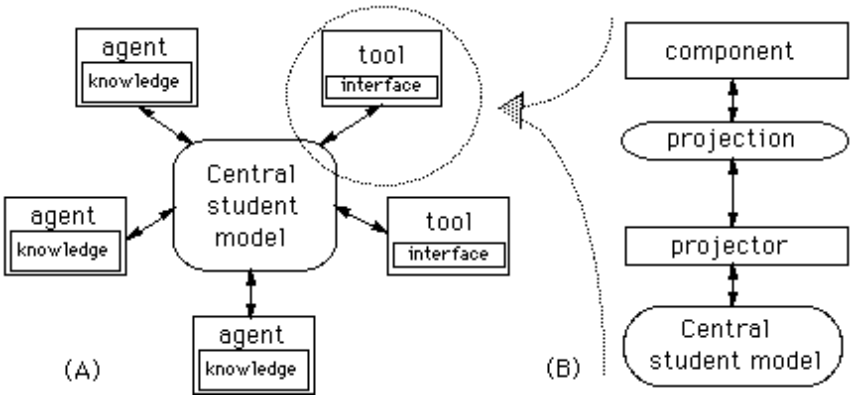

Fig. 1. (a) An architecture for integrated ILE and

(b) possible communications between a component and the central student model

The student model centered architecture separates the student model into two parts, the main student model and the projections. The main student model stands in the center of the environment and collects information about the given student from different sources. Student interaction with any of the system components is reported to the student model in the form of *standard conceptual-level events* . Example: "at time T the student visits the hypernode for the concept C for S seconds". These standard events are time stamped and stored in the model. No further processing is performed in order to avoid the loss of

important information. The main student model is a central repository of all the information about the student that can be used for the purpose of adaptation.

The components of an ILE do not use the student model directly, but instead use local views on the student that we call *projections* . A projection represents that information about the student considered essential for the component to adapt its work to the student. A component has as rich and wide a projection as it needs for the purpose of adaptation. A projection is built and updated from the main student model by a special set of rules called a projector. Each component has its own projection and projector, which provides the interface between the component and the main student model. A subset of the projector rules is used to project the main student model into the local projection. These rules refer to the student model in their left hand sides and contain commands to update the projection in their right hand sides. Another subset of the projector rules is used to provide reverse projection: to project if required the results of the student's work with the component into the form of standard events used by the main student model. Examples of using projections can be found in (Brusilovsky, 1994c; Brusilovsky & Zyryanov, 1993).

From our point of view, the student model of a classic ITS is just one of the local projections: the one used by the tutoring component. Other components of the system (such as the hypermedia component) can use quite different projections. The main student model stores partly processed information about the student, because further processing can lose information important to one of the components. The student model is more than a traditional "history", but is less formalized than a classic overlay model. Rather, it is *a structured history* . Further processing and projecting to more traditional overlay form is made separately by the projectors according to the requirements of different components.

The use of projections and rules provides the open architecture with a good degree of flexibility. Since the performance of the component depends on the projection, we can tune the performance for a particular use by changing the projector rules (or even the projection itself) without influencing other components. A new component can be easily integrated into the environment by designing a set of rules which connect the central model with the given component and its local view on the student. If a new component requires new forms of interaction that cannot be projected into the existing set of standard events, this set can be extended. For example, an event "at time T the student *heard* the presentation of the concept C from a multimedia record" can be projected into event "at time T the student read the description of the concept C" or can be recorded as a new kind of event. If a particular module needs to take into account a new kind of event for better adaptation, its projector can be updated. Thus, the use of projections and rules provides exactly the level of flexibility required by the integration-oriented architecture.

Summary: Dreams or real future?

In this paper we have justified the rationale for future research on an integration-oriented architecture for ILE and reviewed existing work that forms the background for this direction of research. The analysis of existing research let us to imagine some features of future ILE, when and if the common integration-oriented architecture is ever designed by the joint efforts of the ILE community. Such an ILE will probably be a set of independent components integrated around a central student model. Some knowledge-based components, such as the curriculum planner or problem solving support component will be developed by instantiating the proper generic components. Some components of the ILE will be designed by the authors of the ILE, some will be re-used from other ILE projects and some will be just standard commercially available systems. All of these components, however, will be able to send information about the student to the model and to use the model for adaptation. Any components will be able to control other components and to exchange data with them. At the physical level, the interaction

between components will be implemented using inter-module, inter-application or even network communication(Price et al., 1994). In the latter case, some knowledge-rich components of ILE can run on powerful remote computers in another part of the world.

Perhaps soon all components of this imagined ILE and other available ILE could be designed according to a common architecture and will support all standard inter-component protocols. In that situation any teacher could be able to plug together an ILE for the part of the course requiring support for individualized teaching. Teams of developers will be able to concentrate their efforts on really new components and will be able to re-use existing components to make their systems complete. No research effort will be lost. In this scenario, even a single original component designed by a lone researcher could be appreciated and used in many ILE.

References

Anderson, J. R. & Pelletier, R. (1991). A development system for model-tracing tutors. In L. Birnbaum (Eds.), *Proceedings of International Conference on the Learning Sciences* (pp. 1-8). Charlottesville: AACE.

Brette, J.-F. (1994). Contextual guidance on type constraint errors in Pascal/V. In J.-L. Dessalles (Eds.), *Proceedings of CALISCE'94, International conference on Computer Aided Learning and Instruction in SCience and Engineering,* (pp. 337-344). Paris: Telecom.

Brna, P., Hernandes, E. R., & Pain, H. (1993). Learning prolog debugging skills. *Proceedings of Seventh International PEG Conference* (pp. 561-568). Edinburgh.

Brusilovsky, P. (1993). Student as user: Towards an adaptive interface for an intelligent learning environment. In P. Brna, S. Ohlsson, & H. Pain (Eds.), *Proceedings of AI-ED'93,* (pp. 386-393). Charlottesville: AACE.

Brusilovsky, P. (1994a). Adaptive hypermedia: the state of the art. In P. Brusilovsky (Eds.), *Proceedings of East-West International Conference on Multimedia, Hypermedia and Virtual Reality* (pp. 24-29). Moscow: ICSTI.

Brusilovsky, P. (1994b). Explanatory visualization in an educational programming environment: connecting examples with general knowledge. In B. Blumenthal, J. Gornostaev, & C. Unger (Eds.), *Human-Computer Interaction* (pp. 202-212). Berlin: Springer-Verlag.

Brusilovsky, P. (1994c). Student model centered architecture for intelligent learning environment. *Proceedings of 4-th International Conference on User Modeling* (pp. 31-36). Hyannis: MITRE.

Brusilovsky, P., Pesin, L., & Zyryanov, M. (1993). Towards an adaptive hypermedia component for an intelligent learning environment. In L. J. Bass, J. Gornostaev, & C. Unger (Eds.), *Human-Computer Interaction* (pp. 348-358). Berlin: Springer-Verlag.

Brusilovsky, P. & Zyryanov, M. (1993). Intelligent tutor, environment and manual for physical geography. *Proceedings of Seventh International PEG Conference* (pp. 63-73). Edinburgh.

Brusilovsky, P. L. (1992a). A framework for intelligent knowledge sequencing and task sequencing. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 499-506). Berlin:

Springer-Verlag.

Brusilovsky, P. L. (1992b). Intelligent Tutor, Environment and Manual for Introductory Programming. *Educational and Training Technology International* , 29(1), 26-34.

Chee, Y. S., Tan, J. T., & Chan, T. (1993). Applying cognitive apprenticeship to the teaching of Smalltalk in a computer based learning environment. *Proceedings of 7-th International PEG Conference* (pp. 569-588).

Corbett, A. T. & Anderson, J. R. (1993). Student modeling in an intelligent programming tutor. In G. Dettori, B. du Boulay, & E. Lemut (Eds.), *Cognitive Models and Intelligent Environments for Learning Programming* (pp. 135-144). Berlin: Springer-Verlag.

Eisenstadt, M., Price, B. A., & Domingue, J. (1993). Redressing ITS fallacies via software visualization. In G. Dettori, B. du Boulay, & E. Lemut (Eds.), *Cognitive Models and Intelligent Environments for Learning Programming* (pp. 220-234). Berlin: Springer-Verlag.

Fischer, G., Girgensohn, A., Nakakoji, K., & Redmiles, D. (1992). Supporting software designers with integrated domain-oriented design environments. *IEEE Trans. Software Engeneering* , SE-18(6), 511-522.

Fix, V. & Wiedenbeck, S. (1992). Designing a tool for learning ADA using empirical studies. *Proceedings of 5-th Workshop of the "Psychology of programming interest group" (PPIG5)* (pp. 237-246). Paris: INRIA.

Futtersack, M. & Labat, J.-M. (1992). QUIZ, a distributed intelligent tutoring system. In I. Tomek (Eds.), *Proceedings of 4th International Conference, ICCAL'92* (pp. 225-237). Berlin: Springer-Verlag.

Gegg-Harrison, T. S. (1992). Adapting instruction to the student's capabilities. *Journal of Artificial Intelligence in Education* , 3(2), 169-181.

Girard, J., Gauthier, G., & Levesque, S. (1992). Une architecture multiagent. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 172-182). Berlin: Springer-Verlag.

Herzog, C. (1992). From elementary knowledge schemes towards heuristic expertise - designing an ITS in the field of parallel programming. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 183-190). Berlin: Springer-Verlag.

Hohmann, L., Guzdial, M., & Soloway, E. (1992). SODA: a computer-aided design environment for the doing and learning of software design. In I. Tomek (Eds.), *Proceedings of ICCAL'92* (pp. 307-318). Berlin: Springer-Verlag.

Kay, J. (1995). The um toolkit for reusable, long term user models. *User models and user adapted interaction* , 4 (to be published).

Kobsa, A. & Pohl, W. (1995). The BGP-MS user modeling system. *User models and user adapted interaction* , 4(2), 59-106.

Lelouche, R., & Dion, P. (1994). Using the model-tracing methodology in a learning environment with a non-directive tutoring strategy. In J.-L. Dessalles (Eds.), *Proceedings of CALISCE'94, International*

conference on Computer Aided Learning and Instruction in SCience and Engineering, (pp. 259-268). Paris: Telecom.

Linn, M. C. (1992). How can hypermedia tools help teach programming. *Learning and Instruction* , 2,119-139.

Major, N. & Reichgelt, H. (1992). COCA: A shell for intelligent tutoring systems. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 523-530). Berlin: Springer-Verlag.

McCalla, G. I., Greer, J. E., & Scent Research Team. (1990). SCENT-3: An architecture for intelligent advising in problem-solving domains. In C. Frasson & G. Gauthier (Eds.), *Intelligent Tutoring Systems: At the crossroads of artificial intelligence and education* (pp. 140-161). Norwood: Ablex Publishing.

McKendree, J., Radlinski, B., & Atwood, M. E. (1992). The Grace Tutor: a qualified success. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 677-684). Berlin: Springer-Verlag.

Merrill, D. C., Reiser, B. J., Beekelaar, R., & Hamid, A. (1992). Making process visible: Scaffolding learning with reasoning-congruent representations. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 103-110). Berlin: Springer-Verlag.

Möbus, C., Thole, H.-J., & Schröder, O. (1993). Interactive support of planning in a functional, visual programming language. In P. Brna, S. Ohlsson, & H. Pain (Eds.), *Proceedings of AI-ED'93,* (pp. 262-269).

Murray, T. & Woolf, B. P. (1992). Tools for teacher participation in ITS design. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 593-600). Berlin: Springer-Verlag.

Néhémie, P. (1992). A systemic approach for student modelling in a multi-agent aided learning environment. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 475-482). Berlin: Springer-Verlag.

Paiva, A. & Self, J. (1994). TAGUS - a user and learner modeling system. *Proceedings of 4-th International Conference on User Modeling* (pp. 43-49). Hyannis: MITRE.

Price, B. R., McCalla, G. I., & Greer, J. E. (1994). Combining scaffolding and diagnosis in a distributed tutoring system. In P. Brusilovsky, S. Dikareva, J. Greer, & V. Petrushin (Eds.), *Proceedings of 3-rd East-West Conference on Computer Technologies in Education (EW-ED'94)* (pp. 189-194). Moscow: ICSTI.

Ramadhan, H. & du Boulay, B. (1993). Programming environments for novices. In G. Dettori, B. du Boulay, & E. Lemut (Eds.), *Cognitive Models and Intelligent Environments for Learning Programming* (pp. 125-134). Berlin: Springer-Verlag.

Ritter, S. & Koedinger, K. R. (1995). Towards lightweight tutoring agents. *Proc.of AI-ED'95,* See this volume

Rozinajova, V. & Navrat, P. (1993). Making programming knowledge explicit. *Computers and Education* , 21(4), 281-299.

Schneider-Hufschmidt, M., Kühme, T., & Malinowski, U. (Ed.). (1993). *Adaptive user interfaces: Principles and practice* . Amsterdam: North-Holland.

Singley, M. K., Carrol, J. M., & Alpert, S. R. (1993). Incidental reification of goals in an intelligent tutor for Smalltalk. In G. Dettori, B. du Boulay, & E. Lemut (Eds.), *Cognitive Models and Intelligent Environments for Learning Programming* (pp. 145-155). Berlin: Springer-Verlag.

Ueno, H. (1994). Integrated intelligent programming environment for learning programming. *IEICE Transactions on information and systems* , E77-D(1), 68-79.

Van Marcke, K. (1992). Instructional expertise. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent Tutoring Systems* (pp. 234-243). Berlin: Springer-Verlag.

van Rosmalen, P. (1994). SAM, Simulation and Multimedia. In T. de Jong & L. Sarti (Eds.), *Design and production of multimedia and simulation-based learning material* (pp. 167-187). Dordrecht: Kluwer.

Vassileva, J. (1990). An architecture and methodology for creating a domain-independent, plan-based intelligent tutoring system. *Educational and Training Technology International* , 27(4), 386-397.

Verdejo, M. F., Fernandez, I., & Urretavizcaya, M. T. (1993). Methodology and design issues in Capra, an environment for learning program construction. In G. Dettori, B. du Boulay, & E. Lemut (Eds.), *Cognitive Models and Intelligent Environments for Learning Programming* (pp. 156-171). Berlin: Springer-Verlag.

Vivet, M. (1988). Knowledge-based tutors. Towards the design of a shell. *International Journal of Educational Research* , 12(8), 839-850.

Weber, G. (1995). Providing Examples and Individual Remindings in an Intelligent Programming Environment. *Proceedings of AI-ED'95,* See this volume.

Acknowledgments

I would like to thank Ben du Boulay, Jim Greer, and Gerhard Weber who have beed influential in the development of the ideas presented in this paper. Part of this work was supported by the the Royal Society Fellowship and by Alexander von Humboldt-Stiftung Fellowship to the author.