Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

A GENERIC ARCHITECTURE FOR INTELLIGENT INSTRUCTION FOR SIMULATION MODELLING SOFTWARE PACKAGES

Tajudeen A. Atolagbe
Vlatka Hlupic

Department of Computer Science and Information Systems
Brunel University, Uxbridge,
Middlesex UB8 3PH, UNITED KINGDOM

ABSTRACT

This paper describes an architecture for an intelligent interactive instructional simulation modelling environment. It revolves around the production of tutorial and courseware authoring for different simulation software packages with a generic user interface shell. The generic shell is a "*front end*" which provides a uniform graphical user interface to diverse simulation modelling software tutorials. An object oriented perspective is combined with tutorial activities based on the task classification structure to form the interactions between objects. The development environment brings together objects that are functionally coherent and allows them to share common resources within the shell.

1 INTRODUCTION

Organisations adopt different strategies for assessing training needs and ways of implementing them. Because of the increasing use of simulation modelling as an analysis tool and the complexity of developing simulation models, computer based training is a viable option for implementing simulation software training. The provision, maintenance, and continuous updating of courseware, both in academic and industrial establishments, will continue to require huge investments. If an organisation is to sustain its competitive position, full integration of training with business activities, and a framework within which the tutorials needs and strategies for their development are vital.

The Intelligent Simulation Training System (1991), provides a generic architecture for building an intelligent simulation software training system. It uses a generic modelling of simulation scenarios for drilling the learner. The system is domain dependent, and its portability is limited. Its domain knowledge is specifically for simulation modelling. The TOTS system (1987) provides a domain-independent intelligent tutoring shell for task oriented domains and is based on "*procedural network*". The shell is domain specific and does not allow the user to interact directly with the target software. The EXCALIBUR (1987) produced a generic knowledge base system for learning. The system allows the learner to interact with the tutorial program(s).

Using simulation modelling software involves learning the model development skills, knowing what procedures exist for modelling exercises, and how to accomplish various conceptual model formation. It also involves recalling the procedures whenever a similar situation arises. It can be inferred that using a simulation software package requires analytical and cognitive skills.

The aim of this research is to develop an instructional simulation software environment and provide for the learner to interact directly with various simulation software packages. This method permits the learner to acquire procedural knowledge, which forms the foundation for cognitive skill acquisition (1992). The curricular consists of practical activities and illustrations, which help the learner to develop basic skills and knowledge for using the package.

An object oriented approach that provides a generic, direct manipulation graphical user interface for an interactive instructional simulation modelling development environment is presented. The software provides for cross platform integration and development of interactive tutorials for simulation software packages. The object oriented approach allows functionality of the simulation software package to be mapped directly into a tutorial structure.

The tutorial operations and learners mental model can be used to obtain the systems requirements which forms the basis for initial interface design. The architecture embraces systems for producing, authoring and delivery of tutorials for different simulation software packages with a shared graphical user interface. The shell is capable of seamless integration of audio, text and graphical images, which are used at various levels during instruction. The shell is a graphical user interface suitable for accessing heterogeneous instructional software and it can be utilised for verse delivery of tutorials. It embraces systems for the production and authoring for different simulation software packages with common front end across the packages. The "shell" also incorporates different learning aids, and learner interaction is based on direct manipulation dialogue.

Learning hierarchies proposed by Gagne (1965) can be employed to yield task classification structure. This method allows tutorial tasks to be represented in small modules thereby allowing portability and ease of maintenance. Alderman (1978) suggested that hierarchical structure of subject materials is suitable for a *"rule-by-practice"* learning method. The tutorial curriculum can be classified into hierarchical structure based on its functionality of the simulation software and its operational techniques.

The tool possesses three unique features: unique front end, text inputs which provides for a cross-platform courseware development, and an application software environment. These features provide facilities for courseware authoring and maintenance, and for application software to run independent of the tutorial. It can provide the learner with a content rich tutorial environment with an interactive guide.

2 TEACHING SIMULATION MODELLING SOFTWARE

There is a large gap between the functional behaviour of a simulation software package and the learner's perception of software operations. This misapprehension can be attributed to *"cognitive strain"* which users adopt as strategies designed to minimise cognitive load during problem solving and concept-attainment tasks (Bruner et al., 1956). The difficulty of usage can be exacerbated by the process of models representation and analysis. The functionality of the simulation software can be classified into a hierarchical task structure. The behaviour of simulation software can be depicted by constructing a conceptual model. Teaching operations of the software requires cognitive association of all instructional activities, structured into hierarchical array of sub-activities. We identified the following strategies for effective teaching of simulation modelling software: (1) tutorial tasks organised in a pedagogy structure, (2) minimising the cognitive load on the learner, (3) providing meaningful error messages during development, (4) provision of interactive tutorials, (5) use of animation and gaming-simulation as an alternative teaching strategy and (6) providing context sensitive help. The architecture provides for the development of procedural knowledge, direct mapping of structured curriculum tasks to an application model based on appropriate pedagogical structure.

The functionality of a simulation package and the tutorial operations can be classified into a pedagogy structure. The shell allow for direct mapping of simulation problem into predefined structure of the tutorial. The tutorial operations are represented by a root object, on which a hierarchical decomposed task object depends. The tutorial tasks of the software are arranged in pedagogy order, reflecting stages of building a simulation model and post simulation analysis. The sub-task level consist of diverse models of the software behaviour represented in varying levels of description.

An interactive tutorial that permits the learner to interact with simulated scenarios at certain levels during instruction can be provided within the shell. The architecture also uses animation to display the conceptual model of a scenario, with icons representing different activities. This method allows long-term transfer of skills and reinforces the proficiencies of using the software.

The user interface for a simulation software consists of data and functions objects, and their attributes. The learner activities involve manipulation of these objects. The object-oriented model consists of the following semantic elements: all objects are classified into types defined by their behaviour and inter-relation with other objects and their attributes. The behaviour of the objects can be defined by their operation, and their inherent characteristics.

The system provides the learner with an opportunity to exploit most features the simulation package such as ProModel PC, SIMFACTORY 11.5, WITNESS etc. A structured decomposition of the domain knowledge elements and functions are progressively presented as the learner advances through the curriculum. This technique enhances learner participation in building a simulation model of the problem. The shell also provides for unobtrusive "walk-through" stages of model development. Simulation model of a discrete event systems, passes through different activities and the system clock advances asynchronously. Constructing a simulation model involves depicting the activities and entities of the problems through time. This can be represented and presented during instruction by using a structured decomposition of the operations and taking the learner through stages of developing a simulation model.

3 SYSTEM ARCHITECTURE

An *"object oriented"* technique effectively captures and displays systems requirements during the development phases. The generic *"shell"* is represented by objects depicting various operations. The shell is based on the four layers architecture. These consist of the following: the presentation system, the application and tutorial module layer, knowledge representation and domain expert layer, and the main kernel.

1. The presentation layer consists of the graphical user interface (GUI), which consists of a dialogue system that communicates with all objects in the system. It also provides for direct manipulation of all objects on the GUI and provides feedback to the learner. It provides an effective and transparent interface across the shell.
2. The application module consists of a simulation modelling software package and online documentation provided by the vendor. It can be linked directly to the tutorial system and the courseware authoring environment, as shown in figure 1. The courseware authoring environment provides a platform for tutorial development and for courseware maintenance. It provides for components that can be used for integrating text, graphics, animation and for creating or importing dynamic multimedia objects. This is a hypermedia representation and other media objects can be integrated. A tutorial unit may consists of a combination of graphics, digital video images and animation.
3. The knowledge base contain structured domain knowledge, and simulation modelling problem, plus simulation modelling examples. The structured knowledge is organised into hierarchical task units. The control system is the domain dependent problem solving strategy that determines how the tutorial can be manipulated into the learner's requirements. The knowledge base is linked to a domain *"blackboard"* which contains all the instructional strategy and control of the tutorial. It executes rules which depend on the learners' behaviour during instruction.
4. The kernel consists of inheritance characteristics of all objects and their classes. It executes the codes and provides means for sharing objects and manages the memory. The kernel provide pre-emptive threads, and multiple threads operations. It allows for each tutorial unit to provides a pre-emptive *"threads"* to execute other units.

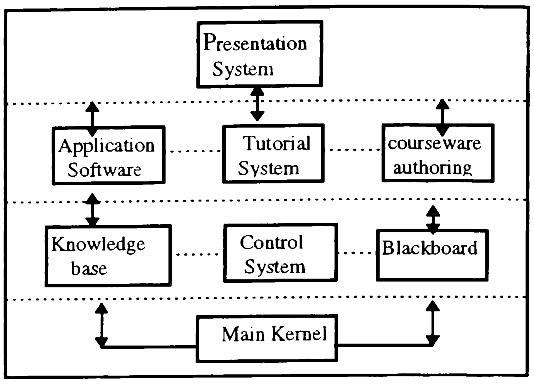

Figure 1: Schematic Representation of the Shell

Each object has defined characteristics which can be embedded into the software. The presentation system provides a uniform representation of the objects. It also provides a mechanism for assessing both the applications and the tutorials modules. The GUI allows the user to communicate directly with the system (1982), whilst the shell consists of a set of routines, which allows the learner to control the instruction. It reflects the general tasks classification structure and semantics for both the tutorials and the application software. The GUI consists of icons that represent objects which the user can manipulate while under instruction. By providing a unique front end, courseware designers can write diverse tutorials for different simulation software packages.

Research findings suggests that acquisition of procedural knowledge can be improved if learner can observe and interact with visual models of the abstract concepts (1982). Shneiderman, (1987) suggested that learners can learn 'quickly' by manipulating the interface object. Tutorial activities depend on the hierarchical structured tasks represented as objects.

The architecture exploits both hypermedia and artificial intelligent methods for courseware authoring. Angelides and Paul (1993), suggested that for an instructional tool to be effective, it should incorporate *"intelligent tutoring"* components. The artificial intelligence components can be embedded within the tutorial environment. The generic tool allows the user to suspend a tutorial unit and interact directly with the simulation software or to seek other method of learning i.e. to choose a demonstration option or observe the animation of the scenario. Some simulation software vendor provided tutorials, for example ProModel PC, permits the learner to interact with a subset of the real application software. These tutorials do not teach the compete functionality of the software. This method may result in a reduction of the learners' skills level development.

The instructional paradigm is based on different techniques of teaching a simulation software package. The tutorial units provides a *"step-by-step"* guide to constructing a simulation model and illustrating the use of all the components of the software. The tutorial objective is for the learner to develop a procedural knowledge for a specific simulation software package. This approach is similar to SOPHIE, I, II, III (1982) which is based on developing the users skills in troubleshooting electronic circuit. The systems are based on expert systems tutoring environment, but offer limited control to the user.

The control system provides for dynamic process communication between the "blackboard" and the event manager, as shown in figure 2. It allows the learner to use their own model and generates simulation models based on their tutorial activities. The event manager controls the operations of the software and the tutorial activities. The control system consists of inference mechanism which controls the structure of the program, and interpreter rules that are embedded in the system. It compares the learner input with the information in its knowledge base and decides which information satisfies the rules. This technique is used for accessing the tutorial network and it limits the search and retrieval time.

The *"kernel"* can execute only one tutorial at a time, but allows user interruption at any time. During interruption it "spawns" supplementary threads to perform other tasks. By separating the graphical user interface from the underlying instructional applications and application software, the shell can be easily transfer to other environments. Other advantages of this approach are versatility, code reusability and ease of updating, Booch (1990).

All the components of the shell send and receive events messages via a protocol. The controller performs this function and co-ordinates the scheduling of processing time between application system and the tutorial system. The protocol determines the order of the events based on priority values set in the program. User interruption triggers events which provides a multitasking environment and allocates the resources.

The blackboard comprises of the student model and a model generator. It can provide for communicating between the two models. The student model provides the learner with different problems based on the user *"overlay"* model. The "overlay" model describes the learner in relation to its interactions with the system and competence level. It maps problem domain into tasks classification structure of the tutorial. It also presents increasingly complex exercises based on the current tutorial unit level.

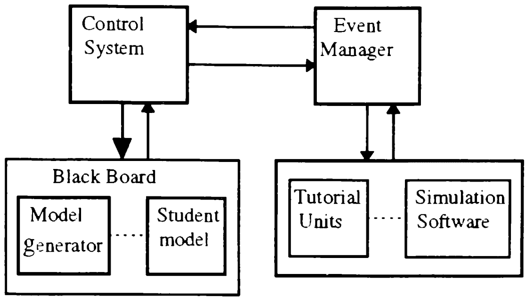

Figure 2: Inter-Process Dialogue

The model generator allows for a hierarchical structured tutorials domain knowledge representation. It provides a structured approach to building a simulation model and consists of combination of the following related phases: (1) identification of simulation concepts and software components, (2) formation of the relationship between concepts and software, (3) hierarchical approach to model representation and mapping problem domain into predefined structure of the tutorial, (4) the output analysis. It first identifies the tutorial needs of the learner and presents the tutorial as represented by rules. It employs the "overlay utilisation rule" to generate simulation model exercises and tutorials.

3.2 Simulation Model

The second layer of the architecture allows for interactive development of simulation models and the representation of the systems components. For a discrete-event simulation model, the state of the *"entities"* change at discrete times. The entity can be represented as objects, characterised by their attributes. An object transforms into different states of activities within an interval. All the objects relates to events, which have dynamic characteristics determined by the user. The *"entities"* advance through different states and behaviour asynchronously and depends on dynamic message routing.

An example of a simulation model of a scenario can be interrupted by the learner at any time. Learner interruption allows for manipulation of the attributes and allows the learner to suspend the current task or perform another operation. The scope and characteristic of the simulation model must be appropriate to the learner's tutorial level and should permit instructional control. Constructing a simulation model of a scenario involves the follows phases: hierarchical representation of the scenario, relating scenario to a classification structure depicting the functionality of the software, and components of the package. Post simulation analysis illustrates output collection, and statistical analysis.

4 CURRICULUM DEVELOPMENT

All simulation software packages possess different characteristics and functionality, they also differ in how entities are represented. Developing curriculum for a simulation modelling package should be a dynamic process set within a context, which unifies the simulation modelling theory and practices. The representation of the domain should consists of the tutorial units with specified internal structures that relate to the theory. It should allow the software to run independently of the tutorial. This representation is similar to that of a Lisp program, that captures the learners behaviour during instructions. It is essential to identify which paradigm will guide the way simulation theory is applied in the tutorial. The difficulties of writing tutorials for a simulation software package without relating to simulation theory is complicated. To minimise these obstructions, the following procedures were formulated:

1. The curriculum is decomposed so that all the contents and instructional strategy reflect the functionality of the software.
2. The curriculum relates to information on simulation modelling theory at appropriate level in the tutorial.
3. Developing models that can be supported by the simulation software
4. An hierarchical representation of building the model, demarcating conceptual representation and statistical analysis.
5. The functionality of the simulation software should be demonstrated in the tutorial.

This representation allows modularity and each composite part is independent of the other. The advantage of this architecture is its simplicity and object oriented nature, plus its usefulness as an integrated courseware authoring tool.

4. 1 Tutorial Units

Discrete event simulation software consist of three major components - library, control shell and report generator, Pidd (1992), and the tutorial units can structured to depict the functionality of the software and these components. The tutorial units can be represented as structured tasks and subtasks, elicited from the tutorial as a tree. The subtask unit portrays the characteristics of the software package and its functionality. Hierarchical task analysis can be applied to organise the operations involved in using the software into a task classification structure, Gagne (1965). The tutorial consists of problem solving operations, with various degrees of complexity within domain. Simulation modelling operations can be decomposed into a coherent sequence. The instructional paradigm is based on the combination of hierarchical task analysis, learner control technique and learner cognitive preference.

Decomposition of the tutorial tasks into structured units can minimise the cognitive complexity of the operations, Gagne (1968). A simulation modelling exercise can be decomposed into smaller subsystems and the curriculum task units can be represented as "tree" in an hierarchical structured network. This consists of several units, arranged in pedagogy array of tasks operations. A unit is represented as a node and it consists of several subtasks which can be depicted as leaves. The tree node can be linked a tutorial units, depicting a component of the software and the leaf nodes correspond to a command in the software. The contents of a node can easily be linked to other child nodes, as shown in figure 2, and addition instructions can be attached as leave at different levels.

The tasks are depicted as graphs with individual units, as nodes and prerequisite relations as links. Instructions are encapsulated on each node. The units are represented by a semantic network and each node represents a tutorial unit, which can be linked to other similar operation. The arcs represents the "*segment*" of tutorial unit.

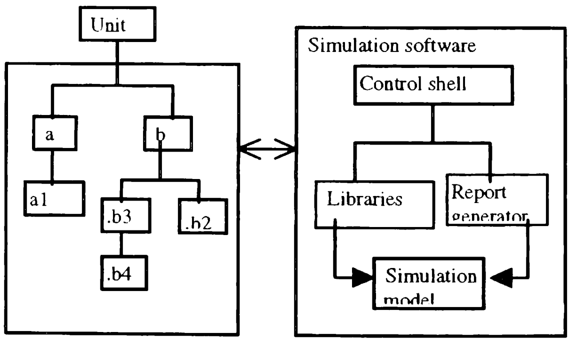

Figure 2: Structured Task Unit and Simulation Software Components

Each node in the structured tasks tree can represent a production rule, which controls the presentation frame. All the nodes possess attributes which indicate the names of up to two or more child nodes that are dependently linked. Navigation through the tutorial depends on the learners path, which can be directly mapped into the task structure.

Each unit is represented as object and every object has one or more activities. All the nodes consists of classes and subclasses. The tutorial can be represented into subtasks levels. These consists of a hierarchical model building process and other operations, which are represented as classes and subclasses in the tasks structure.

The sequence of instruction from a basic model representation to a complex simulation model and analysis employs the *"overlay utilisation rule"*. This rule considers the learners' *"current knowledge"* before presenting the tutorial and can provide control. The learner control provides the choice of content, direct manipulation of instruction, priority of actions and tutorial scope. It also provides control for sequence selection, tutorial option and review. It incorporates commands which can allow the learner to navigate through the tutorial freely.

Tutorial knowledge for a simulation software package can be generated from training and user manuals, task classification structure and structured interviews with experienced users, plus case studies of diverse simulation projects. Knowledge is elicited from the tutorials after careful analysis of the task classification structure which is mapped

5 USER INTERFACE OBJECTS

The user interface objects provides an interactive, *"direct manipulation"* of the curriculum objects and all other functional components of the system. The user interface can support unlimited set of semantic messages and provides direct mapping of tutorial knowledge to the functionality of the software, to a representation on the GUI. All the object have a semantic constituent represented as sets of rules that can manipulate the objects on the interface. Some of the rules describe units of lessons referenced by unique name, each unit consists of sets of classes and objects stored as sets. The tutorial task decomposition is utilised as object hierarchies which depicts the tutorial functions and dialogue specification. This can be used to form a specification of the graphical user interface and depict the object oriented hierarchical structure. The GUI consists of data and object functions which can be inherited. Each application comprises of framework objects. The encapsulated object is specified by its behaviour and state. It permits control of data between objects. Some classes were incorporated for the specific needs of the tutorial. They include the following:

- Action-Class is a lower level class, which maintains a list of actions that are displayed to the learner when appropriate. Each window has an action associated with it. This class is responsible for freeing the memory associated with each action.
- The feedback-Class is a component of the intelligent tutoring system. It provides a content sensitive feedback to the learner. This depends on the type of error and the level of the learner in the task classification structure.
- The Navigation-Class helps the learner to keep up with the current position in the presentation program. It also monitors all the windows titles in the presentation. It allows the learner to navigate throughout the program. The navigation system address context, location, and progress indicator.
- The Windows-Class controls the graphical user interface portion of the sub-frame windows in the presentation. It refreshes the user interface when required, and options chosen by the user. The option can also be used to display textual multiple choice questions. Texts that cannot be displayed are scrolled.
- Model-Class controls all simulation model examples presented to the learner. The learner can use this option to interact with the simulation model to change any of the variables or use the instructional gaming system

These classes present a highly interactive choice of principles and methodologies which can be used by the courseware designer.

Polymorphism of a single task to refer to different sub-task and inter-related tasks, that can be satisfied by rules, represented by the knowledge elicitation method. The form of object orientation is as proposed by Booch (1990). All tasks and operations are modelled as objects, closely linked to the structures and behaviour of the package, incorporating single and multiple inheritance. All objects that have common characteristics and behaviours can by grouped together.

5.1 Portability to other platforms

The major factors for portability are modularization of instruction, with code reusability, file transfer, adaptation to different environment, and cross platform integration of simulation software packages. A *"transfer"* command from the courseware authoring option allows the text file and the interface to be transfer to diverse environment. Courseware authors can also import data from text files into the tutorials. Choice of development software is crucial for the development of the tool, as the primary objective is to produce a tool that will enable the transfer of both the interface and the files to a target environments. The only feasible programming languages are

Visual C++ or Visual Smalltalk. Visual C++ allows partial transfer of its code to other platform, but it is complicated and it has a long development cycles. Visual Smalltalk allows easier transfer of codes to major wider area and is easier to learn. Although both languages support object orientation, Visual Smalltalk has a better memory management and supports multiplatform development. Object Orientation methodology is being used in the design of the software but the development environment will be Visual Smalltalk.

6 CONCLUSIONS

The paper describes an architecture for an intelligent instructional system for simulation modelling software packages. It describes an integrated environment for teaching simulation modelling, courseware production and delivery. The curriculum development environment provides a dynamic process set within a context which unites the simulation modelling theory and the functionality of the software. The tutorial development framework allows for flexible courseware authoring and maintenance. The development environment allows the transfer of both the front end and the text file to diverse environments.

The direct manipulation of user interface object allows the tool to be adaptable and can simulate learning by allowing the user to manipulate objects of the system and adapt a preferred learning styles. It allows the learner to interact with the simulation software package whilst under instructional control. The tutorial of a simulation software can be tailored extensively to suit this methodology. The architecture provides for multiplatform utilisation. This can be achieved by structuring the tutorial into tasks units and by separating the tutorials from the rest of the application. The shell can manage cross platform transfer across the various hardware platforms of Smalltalk and can be implemented within a range of curricular.

The object oriented architecture ensures a uniform approach for analysis and design of the system, and consistencies of the system for courseware authoring and maintenance. The framework objects can be encapsulated, and allow control of data between all objects. Its inhabitant characteristics allows changes to be made to the tutorials when the software functionality is improved. The interactive development environment allows for the courseware author to visualise the functionality of the software and its components. This reduces the time required for analysis and development of the courseware and improves the courseware quality. It also helps to standardise the curriculum and improves usability of the system . The architecture may also provide a platform for development and implementation of such systems.

REFERENCES

Alderman, D.L. 1978. Evaluation of the TICCIT *Journal of Computer Assisted Instructional System in the Community College*. Final Report Vol. 1 Educational Testing Services, Princeton, NY

Anderson, J.R, Breuker. D, and Sandberg, J (eds.) 1989. Psychology and Intelligent Tutoring. Artificial Intelligence and Education, *Proceedings of the 4th International Conference of AI and Education, IOS*, Amsterdam,

Anglids M.C., and Ray Paul. 1993. Towards a Framework for Integrating Tutoring Systems and Gaming-Simulation. *In Proceeding of the Winter Simulation Conference, ed.* G.W. Evans, M. Mollaghasemi, E.C. Rusell, and W.E. Biles, 1281-1289. Institute of Electrical Engineers, Los Angeles, California.

Burton, R.R. and Brown, J.S. 1982. An Investigation of Computer Coaching for Informal Learning Activities. In Sleeman, D. and Brown, J.S. (eds.) *Intelligent tutoring systems*, Academic Press, London.

Shneiderman, B. 1987. *Designing the user interface: strategies for effective human computer interaction*, Addison-Wesley, USA

Bruner, J.S., Goodnow, J.J, Austin, G.A. 1956. *A study of thinking*. New York: Wiley.

Dede, C. 1986. A Review and Synthesis of recent Research in Intelligent Computer-Assisted Instruction. *International Journal of Man-Machine* Studies, 24 329-353.

Gagne, R.M. 1965. *Learning hierarchies.* New York: Holt, Rinehart, and Winston.

Booch, G. 1990. *Object oriented design with applications.* Benjamin/Cummings Publishing Company, Inc. USA.

Hartley, J.R and Tait, K. 1986. Learner Control and Educational Advice in Computer based Learning, The study-station concept. *Computers and Education*, 10, (2), 259-265.

Hartley, J.R. 1973. The Design and Evaluation of an Adaptive Teaching System. *International Journal of Man-Machine Studies*, 5, 421-436

Ohlsson, S. 1993. Impact of Cognitive theory on the practice of courseware authoring. *Journal of Computer Assisted Learning*. Vol. 9 No. 4, 194-221

Richard, T., Webb, G.I., and Craske, N. 1987. *Object oriented control for ICAL systems.* Technical Report, T., La Trobe University, Boundoora, Australia.

Richard, T., Webb, G.I., 1987. *Topic structuring in ECCLES: A simplified CAL authoring Methodology.* Technical Report 9, La University, Boundoora, Australia.

Patrick, J. 1992. *Training research and practice,* Academic Press, London.

Sleeman, D. and Brown, J.S. *Intelligent tutoring systems,* London Academic Press

Draman, M. 1991. A generic Architecture for Intelligent Simulation Training Systems, *In Proceeding of the Winter Simulation. Institute of Electrical Engineers.*

Mitchell, P.D. *EDSIM.* 1978. A Classroom in a Computer for Lesson planning Practise. In Meardy, J, (ed.). *Perspectives on Academic Gaming and Simulation.* pp. 191-204. Kogan Page.

Nuron Data. 1990. *Nextpert object user manual.*

Patrick, J. 1992. *Training research and practice,* Academic Press.

Pidd, M, 1992. *Computer simulation in management science,* Third Edition, John Wiley. London.

Poole, T and Szymankiewicz, J. 1977. *Using simulation to solve problems.* McGraw-Hill Book Company (UK) Ltd.

AUTHOR BIOGRAPHIES

TAJUDEEN ATOLAGBE works as Consultant with Microparadigm Research and Development Group. He is currently undergoing a Ph.D. in the interdisciplinary program in Simulation Modelling and Instructional Systems at the Brunel University. Holds an M.Sc. in Business Information Systems and initial background in Systems Training and Development. Current research interests include aspects of instructional systems, Simulation Modelling, HCI and Object Orientation methods.

VLATKA HLUPIC is a Lecturer in Simulation Modelling in the Department of Computer Science at the Brunel University. She holds a B.Sc.(Econ) and an M.Sc. in Information Systems from the University of Zagreb, and a Ph.D. in information Systems at the London Schools of Economics, England. She is researching into, and has published extensively, in simulation modelling software approaches to manufacturing problems. She has practical experience in the manufacturing and waste disposal industries, as well as having held a variety of teaching posts in England and Croatia. her current research interests are in manufacturing simulation, software evaluation and selection, and in simulators and simulation languages.