A Generic Architecture for Intelligent Simulation Training Systems

Murat Draman

Department of Industrial Engineering
University of Central Florida
P.O. Box 25000, Orlando, FL 32816

Abstract

A generic object-oriented model for building Intelligent Simulation Training Systems with adaptive, interactive, pseudo-real-time simulations is presented. The model provides for the generic modeling of simulation scenarios that train the student to detect and remove undesirable situations, and to create desirable situations. These situations are organized, created (as a tutorial activity), detected, traced, supported and explained by a lattice of generically defined curriculum objects. Simulation objects are modified by the student whose actions may remove or delete situations. The feedback of the system and the evaluation procedures are based on the timing of the student actions, their side effects and their comparison with expert action suggestions. Activated segments of a distinct rule-based domain-expert system generate suggestions and evaluations.

1. Introduction

An Intelligent Simulation Training System (ISTS) may be defined as an Intelligent Tutoring System (ITS) that uses simulation scenarios for training the student. An ISTS monitors the trainee's performance in a realistically simulated and user-friendly environment. Expert feedback that warns about important situations in the scenario and provides suggested actions are used as a basis for instruction. Dynamic feedback or concept explanations are provided by the ISTS upon a trainee request or upon the detection of a performance flaw.

Like ITSs, ISTSs have models for representing and activating three types of knowledge: knowledge to be communicated (the Domain Expert), knowledge about didactic strategies for conveying the domain knowledge (the Tutor), and the knowledge of what is considered learned by the student (the Student Model). Other task modules execute necessary background activities that support tutorial actions. The Translator parses, filters and executes the interactions between the student and the environment. The Diagnose/Interpreter implements diagnostic strategies employed to interpret these interactions. The Evaluator integrates success and failure criteria. Figure 1 shows the basic elements of an ISTS.

The Expert Domain Knowledge of an ISTS relates to the detection/explanation of situations that require trainee intervention, and to the generation/explanation of efficient expert suggestions for handling these situations. The Expert Domain Instructor Knowledge of an ISTS provides ways to create these situations and criteria for the evaluation of trainee interventions.

The task components of an ISTS use these domain-dependent knowledge bases for different purposes. The Translator uses the required communication syntax and scenario-related constraints for filtering the trainee instructions. The Interpreter uses domain knowledge to detect what the student is doing (creating/eliminating desired/undesired situations) and trace the status (red-alert/critical/normal) of current situations. The Evaluator compares trainee instructions to expert suggestions and determines the desired/undesired side effects of these actions. These are recorded as student performance data. The Tutor uses domain-dependent event-creation knowledge to initialize and modify simulation scenarios. Dynamic feedback is also extracted from these knowledge bases.

This paper outlines a generic architecture for building an ISTS as described above. The need for such a system and the principles behind the claim for its generic nature are discussed first. The description of an object-oriented architecture is followed by the outline of its tutorial capabilities.
2. Need for Generic ISTS

It can be seen from the available ITS literature that most of all the applications are prototypes whose aim was to explore some aspects of teaching and/or modeling [3]. Their knowledge models are highly dependent on the tutored domain. As little or no code is reusable, any change to most of them or their application to another domain would require a significant amount of work [6]. Like the components of an ITS, those of an ISTS are still open to research. ISTSs are prone to frequent and significant changes by their designers and by their users.

The organization of an ITS in a series of distinct task components does not automatically bring modularity or domain independence. This often results in multiple and overlapping representations of the domain knowledge [1], and in data and code duplication. This argument also holds for an ISTS, as its task modules access and use this knowledge differently. Changes to any major component or additions to the domain knowledge requires an update of all of the system. The dependence of the components on the application domain renders the adaptation to a new domain impossible.

ISTS architectures must be generic and highly modular, therefore amenable to changes in all of their major components upon requests by their suppliers or users. These features are necessary if progress is to be made without loss of the work invested in the design process and if the development times are to be minimized for new applications.

3. A Generic Model for Representing Simulation Contents

Representing and communicating the subject matter generically are possible by breaking the domain into its smallest, therefore generic, elements about which the system needs to discourse. Such an approach has been demonstrated in an object-oriented architecture for discovery environments [1]. These systems are organized around individual primitive topics to be covered, rather than around the major task components or knowledge bases. These topics are classified as definitions, black-box concepts, concepts to be explicitly demonstrated, and concepts that are discovered by the student through exploration.
This "miniaturization" approach can be extended to dynamic simulation scenarios. The simulations are perceived as a series of primitive events that the student must detect and handle. Events are abstractly defined as situations present or predicted to happen between one or more entities ("Simulation Objects") in the simulation. Events may represent any physical or causal, qualitative or quantitative relationships that can be detected from the implementation of the simulation objects (components, operators, vehicles etc.).

Examples of such events are: an elementary malfunction in a component, the presence of an unidentified object on the radar screen, a specific combination of characteristics in a series of objects, rising temperature or pressure in a series of components, variations of current, pressure or temperature in a series of components, deviation of an object from its assigned path, the predicted violation of a minimum safety distance between two planes (in the Air Traffic Control domain), violation of the safety zone around a robot or moving parts, any single action that the student must execute on a simulation object, any command that must be sent to a simulation object, elimination of an undesired object, allowing an operator to proceed with a requested activity, required step for communicating with an operator, etc. Well-defined variations of an event may differ conceptually or in the simulation object types to which they relate. There may be represented as similar but distinct events.

At any moment of a simulation scenario, a list of current events in the simulated environment (a "snapshot" of events as seen on figure 2) shows the interventions required of the student. An event can be created (caused) and eliminated (accomplished, corrected) by alternative single student instructions. It may also be created by the system as part of a training scenario. A student instruction to a simulation object may create and/or eliminate one or more events, or leave the snapshot of events unchanged. The events that conceptually require multiple actions need to be modeled as a series of successive single-action events.

This model allows for the interpretation and evaluation of the effects of individual trainee instructions in function of the type, number and status of the events that they create and eliminate. An undesirable event that is created by a trainee instruction (as seen in figure 3) has a negative effect on the evaluation of that instruction. Events are given a normal, critical or fatal status in relation to global simulation variables. In domains illustrated with pseudo-real-time simulations, the status of most events relates primarily to simulated time. An example of such an application is the domain of Air Traffic Control (ATC) training. (The examples in this text are chosen from this domain.) As seen in figure 2, the status of the event shows that the student may have ample time to eliminate an event, that there may be a critical need for the elimination, or that it may be too late. The presence of events is also handled flexibly for evaluation purposes: their creation/elimination may be ignored, treated positively or negatively. By default, an event is viewed as a situation that must be eliminated before it is "too late". Consequently, a large number of traced event types allow a finer evaluation of trainee instructions.

4. Proposed Generic Architecture

The architecture outlined here [2] proposes five basic types of generic system objects to organize the curriculum of an ISTS and to link it with a simulation and its associated knowledge bases: Simulator Modules (SMs), Context Modules (CMs), Expert Modules (EMs), and Text Modules (TM) and a Root Module. An object oriented approach is used for the definition of these "Curriculum Object" types, and for the control flow of the final ISTS. The encapsulation, polymorphism, and inheritance capabilities provided by object oriented development languages [5] enhance maintainability, upgradability, flexibility and code sharing for the final product. These basic object types are defined by inheriting the characteristics of a combination of class definitions. Instances of these curriculum objects are generated by an application designer (a "user"). The user integrates simulation-dependent data/code and personal tutoring options/strategies for the creation and monitoring of simulation scenarios in these instances. As seen in Figure 4, they appear as nodes on the curriculum structure that is built as a lattice.

All the information and decisions concerning a specific type of event are centralized in a single SM instance. Each one of these SMs is in charge of creating, detecting, and tracing a single type of event. An event is "injected" in a scenario by creating or modifying one or more simulation objects. Different subtypes of SMs are available. An instance of a particular subtype is chosen by the user according to the creation/detection requirements and characteristics of the event that it represents.
Context Modules organize the activation of the SMs hierarchically. Simulator Modules that are in charge of very similar events, or that need to be activated together are grouped together with links to a unique CM. A CM can therefore be said to represent a topic to be illustrated by the creation of events. Expert Modules assure the interface of the SMs with an expert system for providing expert feedback and domain-related evaluation. Text Modules are used for providing alternative static documentation to all other modules. A single Root Module runs the control algorithm through the modules in the curriculum.

Each instance of these curriculum objects executes specific tasks for a particular type of event or topic. They can be easily included to or excluded from an active scenario. This is done by setting or invalidating their links to the curriculum lattice. As a result, the user may modify the contents and the intensity of a scenario by adding new curriculum objects, and by modifying the control parameters of existing curriculum objects, or by removing them from the active curriculum lattice.

5. Links with the Simulation

An object-oriented implementation is imposed for the application simulation. The simulation scenario consists of a separate process where series of simulation objects receive, execute, and send messages to each other. The object-oriented approach allows the user's own simulation object classes to inherit a generic "basic-simulation-object" class that is provided. Consequently, each instanciated simulation object will inherit a series of basic components and capabilities. These components will supply the links necessary for recording the object, and storage for basic object data such as the object's name, input-history, and its state after the execution of a trainee command. This state may be "stable", "modified-transient" (data readout temporarily inaccurate or unrealistic), or "modified-stable". This allows the integration of realistic simulations. The protocol of this basic class enforces required default message handling capabilities. It also allows the simulation process to update the state of the simulation objects and to report it to interested SMs.
The generic links with a running simulation process are established by an internally generated Simulation-Coordinator object. The Simulation-Coordinator:

- accesses the user-provided code that creates default simulation objects in specified categories (i.e., code that will return an instance of fast-aircraft with default data),
- stores the active simulation objects separately based on their category,
- sets the communication between SMs and active simulation objects (SMs exclusively access the simulation objects of specified categories),
- communicates trainee commands to those simulation objects that are addressed (in the format parsed, if necessary, by a user-defined Translator that may check for syntax: a "klm123 climb and maintain 20000 feet" instruction is parsed into the message "climb 20000" and sent to the simulation object with the id "klm123"),
- communicates instruction refusal messages to the student (pilot refusals, or instructions that are logically wrong or physically impossible for the simulation object to execute), and
- stores executed commands in the input history of the simulation objects.
6. Specifications of Curriculum Objects

6.1 Context Modules

The CMs form a hierarchical network with the Root module at the top. The hierarchy shows the activation priorities of the CMs: it represents the relationships between topics and the preferences of the user in ordering the topics to be illustrated in a scenario. The CMs that are closest to the Root are activated first. CMs placed lower in the hierarchy represent topics that are variations or specializations of those of their parents', topics that are dependent on their parents', or more complicated topics. The topic of a CM is activated after the successful conclusion those of all of its parents. As an example, the user may decide to include the "emergencies" topic and the corresponding events only when regular communication topics are handled properly. The emergency CMs would then be placed as "children" of communication CMs.

Activating a CM allows its linked SMs to operate. Each SM reports its success or failure to its Context Module at the end of its event generation and tracing activity. A user-defined number of such SMs conclude the activity of the CM successfully (a failure is handled similarly). The topic of such a CM is considered to be mastered. The activation of its lower CMs is therefore allowed. A failure would disable all its lower CMs and the children of those.

6.2 Simulator Modules

A series of SMs are linked to a CM to generate variations of relevant events for illustrating its topic. A CM covering the sub-topic of lateral separation-violations between aircraft may be illustrated by the generation of two different type of events by two SMs, one for fast aircraft catching up with a slower aircraft, and the other for aircraft with opposite directions. The events are generated by creating new simulation objects and altering their default data (new aircraft scheduled to enter the sector with potential problems), or by modifying those that are already active. SMs extract data from simulation objects to detect the presence of their events. They analyze the objects when they determine that this is necessary and possible. The format and timing of the event-creation and event-detection activities are determined by the user's tutorial choices, and by the characteristics of the traced event. (It may be computationally costly to detect the future potential violations of an aircraft until it finishes a turning maneuver and stabilizes its direction: this could be considered to be a transient period until the end of which a particular SM would wait to access data.) Each SM keeps a record of its own current events in the simulation.
The SMs analyze the effects of trainee instructions on their own partial snapshot of the simulation. They "blame" trainee instructions for creating events, and "credit" them for eliminating events. The analysis of a modified simulation object by all interested SMs will produce the global effects of the trainee input.

For the event-creation process of an SM, the user specifies:
1. the categories and quantities of simulation objects to be created or modified,
2. the internal data components of the objects that need to be modified (random or fixed value for their coordinates, speed, flight-plans, etc.),
3. the message to be executed by the object after its creation ("show-self"),
4. additional modification done in function of the other active objects that need to be simultaneously checked (user defined code and its activation format for deviating an aircraft from its planned route-or-flight),
5. the maximum number of events to be generated by the SM,
6. the minimum and maximum number of events that must always be present in the simulation,
7. the minimum time interval between generations, and
8. the time of generation for the first event (some events may be present at startup).

For the event-detection process of an SM, the user specifies:
1. user-defined code that will detect the event (time-dependent path-checker code for detecting collisions and for returning specific data in specified format),
2. the categories of simulation objects that relate to the event,
3. the ways these objects are grouped for analysis (this specifies the format in which the objects of specified categories are accessed by the event-detecting code),
4. trainee instructions that these objects execute, that may cause or eliminate this particular event (an identification request will not interest an SM that is checking for separation violations),
5. the presence of transient periods (this specifies if the SM should wait for a "stabilized" message from the object to access its data),
6. credit/blame strategy for multiple inputs recorded during transient periods (precedence in time and/or importance of type of input),
7. the possibility of the appearance of the event without the execution of a message by the simulation object (this specifies a requirement for periodic, instead of modification-activated checking; detecting pilot behaviors, requirements for communications to the aircraft after they reach a typical location or the limit of the controlled area, etc.),
8. the priority of the event, and
9. the criteria/code for determining when the event becomes critical of fatal.

Each SM keeps its own student records:
1. number of events that were created or eliminated by the trainee and their status,
2. the evaluations of each new/gone event by linked Expert Modules,
3. the number of poorly handled events, and
4. the grade averaged for the event.

Based on these records, different options are given to the user for allowing an SM to initiate help feedback. These records are also used as a basis for the continuation or the successful/unsuccessful abandonment of the activity of the SM.

6.3 Expert Modules

The expertise related to topics of the events are accessed by the EMs. This expertise needs to be defined in terms of suggestions that eliminate the events with effective actions. Multiple Expert Modules may be linked to the an SM. Each one activates its individual knowledge base (set of rules) that provides a different type of action to eliminate each event detected by the SM: one EM that solves separation-violations may generate instructions of type "turn", another of type "climb", etc. The individual activation of EMs allows the knowledge bases that are not interested in the current changes to stay inactive.

An SM initially communicates the appearance of its events to its Expert Modules. The EMs pass that data to their rules in the specified format (a "fact" asserted in the "blackboard" of the knowledge base). An SM alerts its EMs of relevant messages received by its simulation objects. It also issues a warning if modifications are noticed in the simulation data retrieved during successive detections of an event. If the knowledge base associated with an EM generates its action for an event at the first detection of the event, this action may need to be updated when the modification warnings are received (earlier expert suggestions to "climb" may not be valid after a set of instructions that modified the available airspace). Alternatively, the
knowledge base may generate its action when requested: the SM requires it at the elimination of the event. In this case, relevant simulation data may need to be stored by the knowledge base when the warnings are received. This may be necessary as simulation data may be irreversibly changed: the expert suggestions generated for comparison with the trainee action must relate to the state of the simulation object and its environment before that action is executed. The first option may be easier for communication-type events, the second for time/space related events.

To evaluate the elimination of an event, the SM compares the suggested actions received from the EMs to the trainee's input. It tries to find a match. The action of a single EM may produce a match. An input that perfectly matches a suggested action should simply eliminate events. A partial match (matching action, nonmatching specifications or range: the trainee's executed climbing altitude is outside of the range suggested by the expert) or a non-match are considered as the reason for the creation of events. The results of the comparison and data showing the created/eliminated event instances are sent back to the knowledge base of the matching EM to be evaluated. The event instances accessed by the knowledge bases contain data that shows their importance and status. The evaluation grades are classified as good, acceptable and bad according to the user's specifications and stored in the student records of the SM. A trainee input that does not create/eliminate any event for any SM is considered "inefficient" if a partial match is found, or "marginal" if no match is found. For such inputs, the evaluation results are collected from all the SMs that are affected by the input.

6.4 Text Modules

The TMs provide static text storage facilities for supporting definitions, hints, explanations etc. They may be attached to any other module for explaining a topic, event, or rationale behind a typical expert action. The user may integrate any number of alternative explanations with consecutive levels of complexity. The TMs may be set to limit the number of times that they may be activated, the maximum level of complexity allowed to be explored, and the number of times the same text may be repeated.

7. Tutorial Activity

This model does not build a cognitive model of the trainee nor does it speculate on the reasons of trainee actions. The role of the trainee's understanding in his/her performance and the assessment of his/her performance are still active areas for research [4]. Here, the trainee is drilled: he/she is allowed to modify the simulation as desired, as long as his/her instructions are not refused by the objects in the simulation. The results of his/her actions or omissions are determined
and evaluated. The events that the trainee action eliminated are detected, and the trainee action is compared with suggested expert actions generated for those events. The side-effects of the trainee action are blamed to its differences with the suggested expert actions. When a trainee timely eliminates a series of simulation events without undesired side-effects, and receives good evaluations from the knowledge bases, the corresponding topics are considered mastered.

Any SM may activate its help/feedback capability with an explanatory warning when a user specified threshold is reached (a certain number of unsatisfactory event creations, a bad evaluation that follows a perfect action, etc.). Figure 5 shows the extent of the feedback that may be received when the trainee explores the curriculum lattice. The feedback given to the trainee includes:

- the current events of an SM,
- the reasons provided by the simulation/user code for the presence of these current events,
- alternative actions and their documentation as received from the knowledge bases,
- descriptions received from Text Modules (the contents of Text Modules may be ideally replaced by a localized discourse system),
- results of the actions in terms of the events that were created and eliminated, and
- the evaluation of each trainee action.

8. Conclusion

The presented model allows the creation and monitoring of non-trivial dynamic simulations. The scenario evolves in time, as the trainee and the tutor are allowed to modify it unpredictably. The tutor modifies the scenario as planned, and as a reaction to trainee actions. The trainee is expected to take proper and timely actions to eliminate undesirable situations and create desirable ones. Dynamic adaptive feedback is supported as the expert knowledge bases are interfaced to solve the problems generated by the trainee and by the tutor.

The object-oriented environment includes high-level definitions that are harder to master when compared to a typical artificial intelligence shell or language. Nevertheless, building an ISTS from scratch with an AI language or shell involves a much larger design effort. The modularity, flexibility and upgradability of the model, and the generic nature of its control mechanism and elements would save its users a considerable amount of time. Ideally, such a model would be considerably enhanced with an authoring interface that minimizes or eliminates the need for the integrating the user's code in specified formats. Such an interface would be sophisticated enough to integrate the functionality of such code.

A prototype of this architecture and its user interface, the Tutorial Tool for Expert-Supervised Simulations (TTESS) [2], has been developed in Flavors and Common LISP on a Symbolics LISP machine. TTESS interfaces with a rule based expert system written in ART.

9. References


