# A Goal-Oriented Development Tool to Automate the Incorporation of Intelligent Agents into Interactive Digital Media Applications

HAN YU, ZHIQI SHEN, AND CHUNYAN MIAO
Nanyang Technological University, Singapore

Recent developments in games and interactive storytelling applications have seen artificially intelligent computer controlled characters being included extensively. Non-human controlled characters are starting to play an increasingly significant role in enhancing the perceived intelligence of games. Although many of them employed certain cheating techniques (e.g. allocating more resources at the start to AI opponents to make them appear more aggressive), some limited learning did appear in several games (e.g. letting AI opponents remember where human users initiated attacked in previous game). In our *Virtual Singapura* research project, we incorporate software agents into our virtual world to provide more complex user interactions. With intelligent software agents being infused into interactive digital media applications, there is great potential in improving the overall user experience. However, during the process of our research, we discovered that the traditional way of adding a multi-agent system into a computer game requires a large amount of investment in time and resources and a high level of expertise in Agent Oriented Software Engineering (AOSE). Moreover, game AI is usually closely coupled with other parts of the game code which makes it hard to reuse or replace. This research proposes a multi-agent development and runtime framework which not only provides ease-of-use agent design and implementation tools but also can be easily plugged into various interactive digital media applications.

Categories and Subject Descriptors: J.6 [**Computer Applications**]: Computer-Aided Engineering – *Computer-aided design*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Intelligent agents*.

General Terms: Design

Additional Key Words and Phrases: Goal-oriented, intelligent agent, multi-agent system, design tool, distributed systems

**ACM Reference Format:**
Yu, H., Shen, Z., and Miao, C. 2008. A goal-oriented development tool to automate the incorporation of intelligent agents into interactive digital media applications. *ACM Comput. Entertain*. 6, 2, Article 24 (July 2008), 15 pages. DOI = 10.1145/1371216.1371227 http://doi.acm.org/10.1145/1371216.1371227

## 1. INTRODUCTION

There have been increased interests of employing autonomous agents (e.g. lifelike animated virtual characters) in games especially role playing games, storytelling and other interactive digital applications. Intelligent agents are useful in such applications as they are capable of observing and reacting to user avatars and events in the virtual world.

Autonomous agents are computational systems that inhabit some complex, dynamic environment, sense and act autonomously in this environment and by doing so realize a

This research was partially supported by the Ministry of Education (MOE), Singapore Research Grant.
Authors' addresses: School of Computer Engineering, Nanyang Technological University, Singapore 639798;
e-mails: yuhan@pmail.ntu.edu.sg, zqshen@ntu.edu.sg, ascymiao@ntu.edu.sg.

set of goals or tasks for which they are designed [Chunyan et al. 2001; 2002; 2003]. Various interactive digital applications such as massive multiplayer online games in virtual world are examples of "complex and dynamic" environment that allows individuals, through their self-created virtual characters or "avatars," to interact not only with the virtual world but with other players' avatars as well. For example, students enjoy being immersed in virtual world, seeking richer and more engaging learning experiences. Researchers are exploring innovative ways to use virtual life space to foster interaction, collaboration, and excitement for learning. With many people spent increased hours in life in virtual spaces, there have been a strong need to infuse autonomous agents to offering a richer playing experiences.

Nevertheless, the traditional way of adding a multi-agent system into a computer game requires a large amount of investment in time and resources and a high level of expertise in Agent Oriented Software Engineering (AOSE) [Peter et al., to appear]. Moreover, game AI is usually closely coupled with other parts of the game code which makes it hard to reuse or replace. Our research aims to develop a multi-agent development and runtime framework which not only provides ease-of-use agent design and implementation tools but also can be easily plugged into various interactive digital media applications.

Agents are goal oriented [Zhiqi and Robert 2003]. The goal oriented model most closely resembles how people, in their day to day activities, make decisions to direct their actions. It has great potential since it can be understood easily by people with little expertise in intelligent software agent technology. Nowadays, most of the available intelligent agent design tools are targeted at users who are experts or at least highly skilled in the field of intelligent agents, to design a multi-agent system or agent mental states. One has to be even more highly competent in programming to actually implement the design. There is still a large gap between converting a storyline or a game plot into a multi-agent system for gaming or interactive storytelling applications. Therefore, this research proposes a goal oriented agent modeling and design tool – Goal Net Designer - to bridge this gap.

The Goal Net model is a goal oriented agent development methodology that covers the entire life cycle of the agent augmented system development from requirement analysis, architectural design, and detailed design to implementation [Zhiqi et al. 2006]. It has a practical implementation - the Multi-Agent Development Environment (MADE), which consists of two major components: The Goal Net Designer and the MADE Runtime. Out of the four main phases of agent augmented systems development, Goal Net Designer covers the requirement analysis, architectural design and the detailed design. The fourth phase is supported by the Agent Creator, Knowledge Loader and Agent Space Manager (collectively known as the MADE Runtime) which depends on data collected by the Goal Net Designer [Zhiqi et al. 2006]. Since MADE Runtime handles the actual creation and execution of the agents, the users are not concerned with agent oriented programming and, instead, can focus more of their time and resources on the design of their agent augmented applications.

With a friendly user interface based mainly on drag-and-drop operations, users can modify the design in an iterative manner with considerably less effort. This is especially important because changes to part of the goal net may affect other parts which make it error prone and difficult to manually keep track of them. The Goal Net Designer is implemented as a distributed system according to client-server architecture. This is to ensure that teams store their goal net designs on a centralized server which also provides generic and custom-made functions and web services for all developers to use. The goal

net designs from different teams can also be easily reused by others. With the proposed agent modeling and design tools, in our *Virtual Singapura* research project, we have successfully incorporate software agents into virtual world to provide more complex user interactions and new learning experiences.

In the following sections, we will give an overview of the Goal Net Methodology, introduce the architecture of MADE, discuss the design and important features of the Goal Net Designer, provide a demonstration of how to develop an agent using our tool under the Virtual Singapura Project test bed and conduct a usability study of the MADE Framework based on a real-world application "*Virtual Singapura*".

## 2. RELATED WORK

Currently, there are a number of approaches for designing multi-agent systems and agent mental states. One of the most popular tools for intelligent agent development is the Java Agent DEvelopment Framework (JADE). It provides an Application Programming Interface (API) to support the implementation of a multi-agent system in JAVA programming language [Fabio et al.]. However, JADE places its emphasis mostly on coding. In order to create an agent in JADE, one has to be familiar with both the Java programming language as well as the agent oriented programming paradigm. This is more often than not too much to expect from a designer of a game plot or storyline of an agent augmented interactive digital media application. Furthermore, JADE does not support agent goal modeling. Therefore, even for seasoned developers, using JADE to implement a software agent is by no means an easy task.

Another approach for designing multi-agent systems and agent mental states is to sketch the design on paper and feed it to a computer with software capable of making sense of it. In most cases, the software is platform and language dependent. Although the SketchiXML project offers a solution to this problem [Adrien et al.], it has nevertheless some inherent drawbacks by using this approach. Firstly, users must draw out the design following closely to the shapes of the recognized entities involved. Secondly, as the design is being modified, the user must also follow predefined patterns to delete unwanted items or simply start all over again. In the end, this approach does not enable the users to specify internal properties of each design item during the sketching phase. The skill level required to make use of this design approach is high and the implementation is still a long way to go from the design.

The Prometheus Design Tool (PDT) proposed by RMIT University, Australia, provides a wide range of support for the various stages of multi-agent system design using the Prometheus methodology. It has been integrated with Eclipse to provide a more closely coupled implementation process [Padgham et al. 2005]. However, the data acquired at design time serves only as a reference to the implementation process (which still has to be accomplished manually by writing program code). The correlation between the software agent implementation and agent data collected from PDT is not set up automatically.

The Tropos methodology is one of the most similar existing methodologies to the Goal Net Methodology. It is goal oriented [Jaelson et al. 2000]. A wide range of tools have been developed based on it and they cover a quite complete spectrum of the agent development process. However, many of these tools focus on a specific design area and there is a lack of tool to unify the entire design process. Although support of automatically generated multi-agent system based on Tropos has also been reported [Penserini et al. 2006]. Unfortunately, since it relied on the Jadex project, the code generated is limited to the Java language and must work within the JADE middleware.

Currently, the majority of games which require complex AI are not written in Java, using their approach may cause performance degradation when the agent running under JADE tries to interoperate with the game engine (written in C/C++ for example) via Java Native Interface (JNI).

## 3. THE GOAL NET METHODOLOGY

Goal Net is a composite goal hierarchy which consists of states (goals) and transitions [Zhiqi et al. 2006]. The states are used to represent the intermediate steps that an agent needs to go through in order to achieve its final goal. The transitions connect one state to another specifying the relationship between the states it joins. They can be abstracted to a mathematical function for the agent to transit from one state to another. Each transition must have at least one input state and one output state. A transition is associated with a task list which defines the possible tasks that agent may need to perform in order to alter the agent's state. An agent can be expressed by a single or multiple goal nets. A complex system can be recursively decomposed into sub-goals and sub-goal-nets (e.g. an entire system can have its various states modeled by the goal net methodology so that it appears as 'super-agent' which contains smaller agents each performing specific tasks to accomplish the overall goal). In such a manner, the system can be easily modeled and simplified. Figure 1 shows a conceptual drawing of a goal net.

The following list defines important concepts involved in a goal net design in the context of the goal net designer [Zhiqi et al. 2006].

- A *goal net* is a hierarchical net. It contains a set of states, transitions and arcs organized into a network to achieve an overall goal.
- A *state* is a tuple containing a set of variables that define the profile of the state, a set of application variables and a set of internal functions.
- A *composite state* is a state which contains additional information on the initial and target states of the sub goal-net it represents.
- A *transition* is a tuple containing a set of variables that define the profile of the transition, a set of application specific variables and a task list which consists of a finite set of tasks each, in turn, consisting of a finite set of functions.
- An *arc* is a tuple containing a set of variables which define the profile of the arc, a link to the input state/transition and a link to the output transition/state.

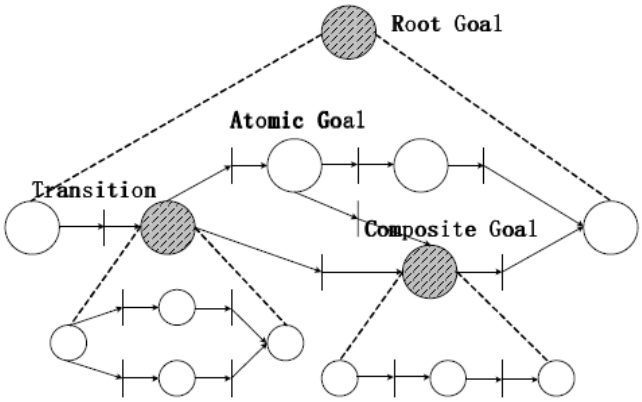


Fig. 1. Goal Net Design.

The Goal Net Methodology has defined a set of semantics to describe the meaning of the design data integrity requirement for any goal net. A goal net design has to satisfy three properties, namely: safety, liveness and modularity, in order to be correct [Zhiqi et al. 2006].

A refinement of a composite state in a goal net is the process of replacing it with the states, transitions and arcs that form its sub-goal net. A refinement of a goal net is safe if all its sub-goals are reachable from its initial state. For a goal net to be safe, its refinement must be safe [Zhiqi et al. 2006].

Liveness refers to the reachability of transitions. A transition is live if there is a transition firing sequence that includes it. A refinement of a goal net is live only if all its transitions are live. A goal satisfies the liveness property only if its refinement is live [Zhiqi et al. 2006].

Modularity is concerned with the preservation of the hierarchical structure of a goal net. In order to do so, the sub-goals of two different composite states must not be connected by a transition [Zhiqi et al. 2006].

## 4. THE SYSTEM ARCHITECTURE OF MADE

As shown in Figure 2, the Multi-Agent Development Environment (MADE) provides a framework in which multi-agent systems can be designed, implemented and deployed. Of these three stages, users are mainly involved in the design process. The implementation of agents using MADE still requires some program code to support functions required by a specific application. This is needed when the agents are set to run under a customized environment of the application (e.g. an online virtual 3D community). But the coding involves no Agent-Oriented Software Engineering (AOSE) concepts. The users can make use of procedural or object oriented programming techniques to implement their logic. Other than this part which requires the participation of technically inclined programmers, the agent implementation and execution of the multi-agent system according to the design have been automated by MADE. In our previous research, the kernel MADE has been used in agent based service composition and integration in e-learning, bio-informatics and manufacturing applications. [Peter et al., Zhiqi et al. 2005]

In this research, we focused on enhancing the usability of MADE with easy-to-use interfaces i.e. Goal Net Designer to enable the novice users to create agents without learning agent programming. The designers/users (usually people with limited technical

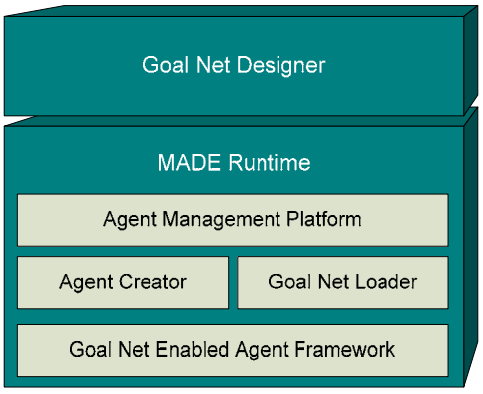


Fig. 2. MADE architecture.

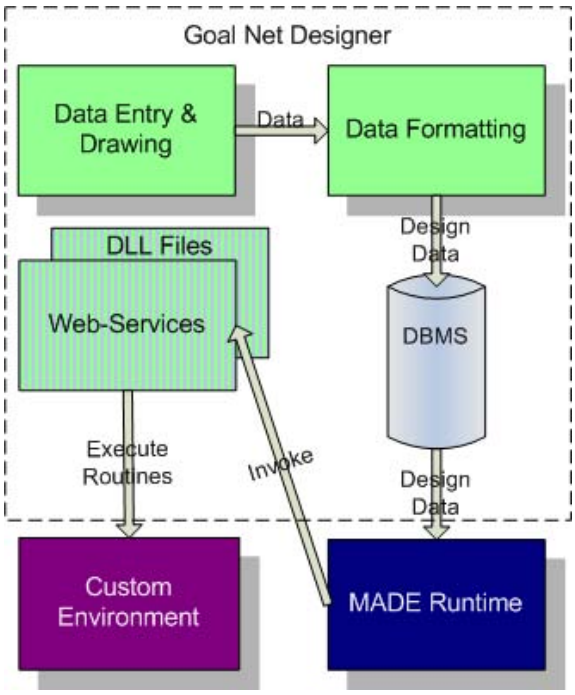


Fig. 3. Goal Net Designer architecture.

knowledge), can make use of a GUI tool – the Goal Net Designer- to draw a goal net based on the dependencies among the goals and transitions. This process is straight forward as it models how human logic works, so only minimal training is required to get familiar with the features of the tool in order to complete the design. After the Goal Net design, the designed drawing is automatically converted to design data and saved in a database. Goal Net Designer is an important component of MADE. Figure 3 shows the architecture of the Goal Net Designer.

Agents are created and run by the MADE Runtime which is essentially a virtual machine that interprets the goal net design data. Whenever a reference to a routine defined in a transition, which is specific to the customized environment of the application, is encountered, the MADE runtime would redirect the call to the DLL files or web-services provided by the developers. The function defined in the DLL files/web-services would be executed in the customized environment. To optimize the performance of MADE runtime in order to keep up with the real time requirement of the interactive digital media application, the MADE runtime is mainly written in C/C++ with critical parts written in Intel 80x86 assembly language.

## 5. THE GOAL NET DESIGNER

### 5.1 Architectural Design Patterns

To enhance the reusability of the business logic and decouple it from the user interface as well as the data access parts of the Goal Net Designer, the *Model-View-Controller* design pattern as follows.

All the custom defined classes which facilitate the effective representation of the goal net and the rendering of the graphical visualization of it are separated out to be contained

in the *Model* layer. However, there are a number of properties which track the status of the goal net and require monitoring in order for the *View* layer (Graphical User Interface (GUI)) to respond to them accordingly. As the GUI is transparent to the *Model* layer, the *Model* layer cannot determine how the *View* layer shall respond to the changes in the goal net status. To avoid the use of polling, which wastes system resources, the *Delegation* pattern is used to create an event for each status variable in the *Model* layer and set it to be triggered upon the change of value of the variable. The advantage of the *View* layer implementing responses to these delegated events is that if a different technology is used to implement the *View* layer (e.g. ASP.NET instead of WinForms), no changes to the *Model* layer will be required. In addition, the event is raised only when the value of the monitored variable changes. It is essentially a software interrupt mechanism.

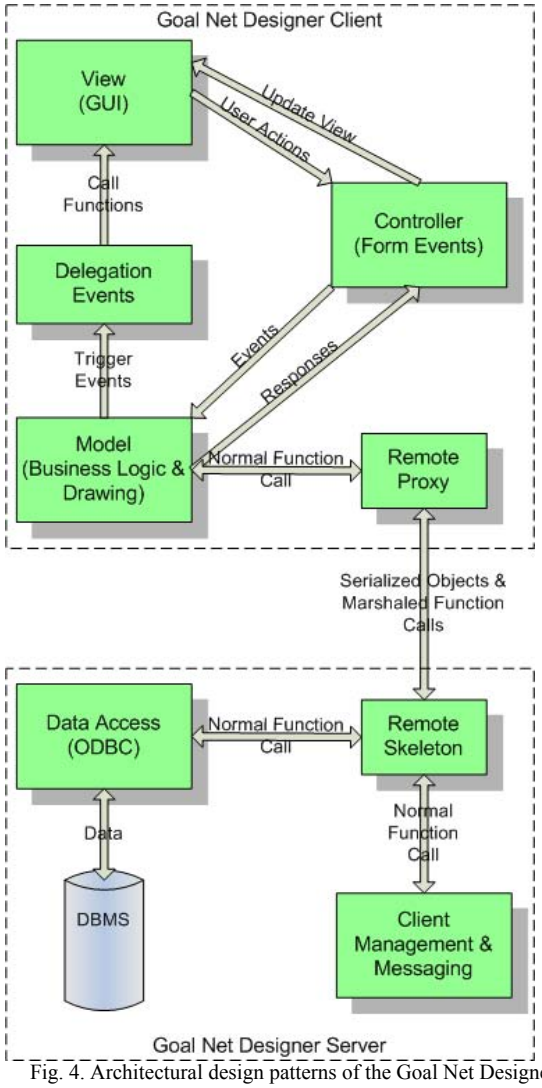


Fig. 4. Architectural design patterns of the Goal Net Designer.

The Data Access layer is also designed to be a separate layer from the rest. This is mainly due to the concern that during later stages of development, different database management systems (DBMS) may be employed or the data storage may even be residing on a remote server.

Since the user interface of the Goal Net Designer is based on Windows Forms, the *View* and *Controller* layers are closely related to each other. The *View* layer not only displays the graphical representation of the goal net, but also generates events in response to user actions so that the *Controller* layer can process them by delegating them to the business logic module in the *Model* layer.

In order to support distributed deployment of the goal net designer for a large group of developers to work in collaboration, the *Client-Server* architectural pattern is selected. This is mainly due to the fact that central data storage is required for collaborative development and certain entities (like functions and web services) have to be managed at a central location. To minimize network access and hence the network delay, the task of rendering the graphical representation of the goal net on the user interface is assigned to the client. In order to achieve this, an entire object of the custom created class *GoalNet* which contains relevant data structures as well as business logics is to be sent to the client after its creation on the server (Marshalling by Object). Here after, the client does all the processing locally. Upon user request, the design is to be encapsulated into the *GoalNet* object and sent to the server for storage into the database. This effectively makes the client a *Fat Client*. As the *Data Access* layer resides on the server, it publishes a list of functions for remote access from the clients and itself is marshaled to each client as a reference. This is accomplished by creating a *Remote Proxy* for the *Data Access* layer and directing all database operation requests from a client to the server residing on a different machine.

### 5.2 Important Features of the Goal Net Designer Server

The Goal Net Designer client and server applications are developed based on the Microsoft .NET framework. Figure 5 depicts the user interface provided by the Goal Net Designer server. The server communicates with each client via HTTP sockets. Both client and server prototypes are distributed as automatic installation programs so that the users can be freed from the process of setting up environmental variables and version control. Current development supports five main functional areas (as highlighted in Figure 5) for the system administrator.

Area 1 is the web services panel. In order to support more distributed development and to be less programming language specific, web services are being added into the design process to supplement the DLL modules written in C/C++. An experimental Universal Description Discovery and Integration (UDDI) server has been set up for the MADE team to implement and test web service support as well as to work out a process for the users of the Goal Net Designer to supply their own web services to be used in the final multi-agent systems they build. Once the web services are added into the UDDI server, the administrator can press the *Update Web Services* button for an immediate update. Alternatively, the Goal Net Designer server also periodically polls the UDDI server to check for any new update. Once the update is complete, newly added web services will be made available to the functions lists of all active clients via callback and they will be able to include them in their design.

Area 2 is the client management facility. It provides the server administrator with information on which clients are currently connected to the server and enables the administrator to send and receive instant text messages to and from the clients. When the

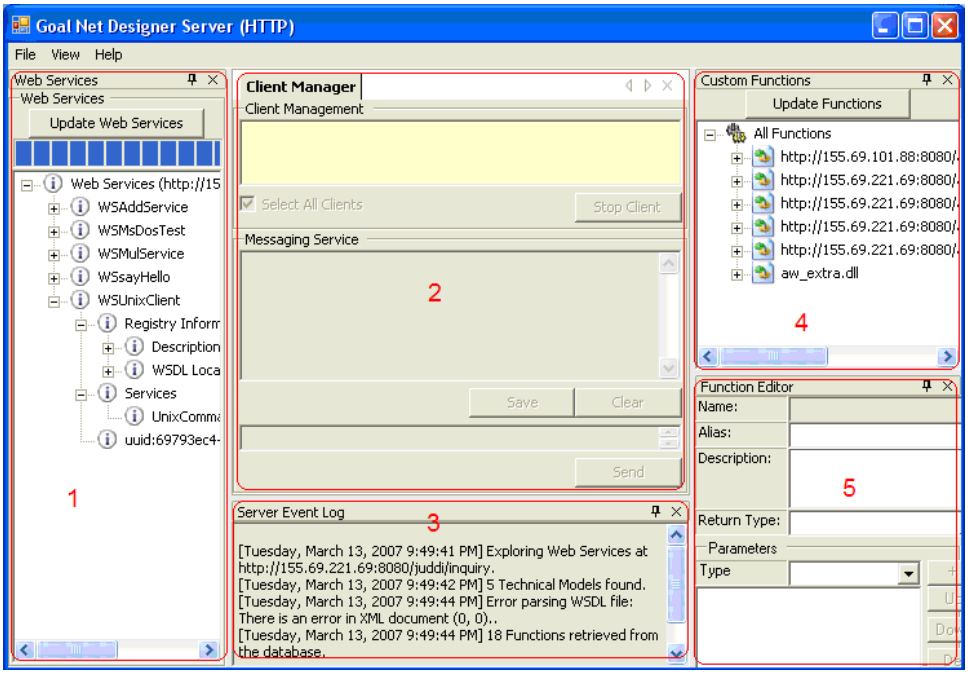


Fig. 5. Goal Net Designer server.

server needs to be shut down for maintenance or upgrade, the administrator can use the *Stop Client* button to safely force the active clients to close. Therefore, the clients' work can be saved before the server goes offline.

Area 3 is the server event log. It records down all activities initiated both by the server and the clients as well as any system generated error messages in human readable format with the date and time of occurrence. The log can be saved to a text file for future reference. Area 2 and 3 are primarily for system monitoring and job accounting purposes and do not have much to do with the multi-agent system development process.

Area 4 manages the custom written functions provided by the users. A dedicated folder is setup for the users to upload their DLL files containing C/C++ functions to the server. The server administrator can initiate a function update by pressing the *Update Functions* button. The server is also polling the designated folder periodically to automatically collect newly modified functions and update the list of published functions.

Area 5 provides an interface to display more details concerning each function and to let the server administrator put in additional information (like descriptions etc.) about each function. These pieces of information are read-only at client side.

The layout of the Goal Net Designer server can be customized by the server administrator in other ways not limited to the one shown in Figure 5.

### 5.3 Important Features of the Goal Net Designer Client

Figure 6 shows the user interface for the Goal Net Designer client. It provides the users with a unified tool to manage all the design data of multiple goal nets as well as to collaborate with other designers. The client uses a HTTP connection to communicate with the server. In the current prototype, messages passed amongst clients have to be re-

routed via the server first. Design data is stored at a centralized database at server side. When an existing goal net design is requested, the client pulls the necessary data from the server and temporarily stores it in local memory. The user actually modifies only their local copy of data and when the design is complete, the local copy is pushed back to the server. To simplify the problem of conflict resolution, the current prototype uses an exclusive locking technique to allow only one user to edit a goal net at any one time. From the observations of the usage pattern by the trial designers (i.e. undergraduate students from Nanyang Technological University and teachers from the National Institute of Education, Singapore), once a goal net is designed, the majority of subsequent accesses are read-only. In addition, each read/write access requires one entire goal net design to be loaded into the client's local machine. Therefore, the lock granularity is decided to be one goal net design file. This coarse lock granularity satisfies the usages pattern of the current prototype and reduces the overhead of transferring and storing the locks. Five main functional areas (as highlighted in Figure 6) are supported at the moment.

Area 1 is the function view in the client. It is read-only to the user and its contents are kept up to date by means of a server callback. Apart from viewing the information of all the functions (organized in a hierarchical structure that reflects which function belongs to which DLL file or web service), the users are able to associate the functions with goals and transitions in their goal net designs by dragging on the function names and dropping them on the desired target. These functions specify what an agent needs to accomplish in order to transit from one state to another or the logic to select the next transition path at a state.

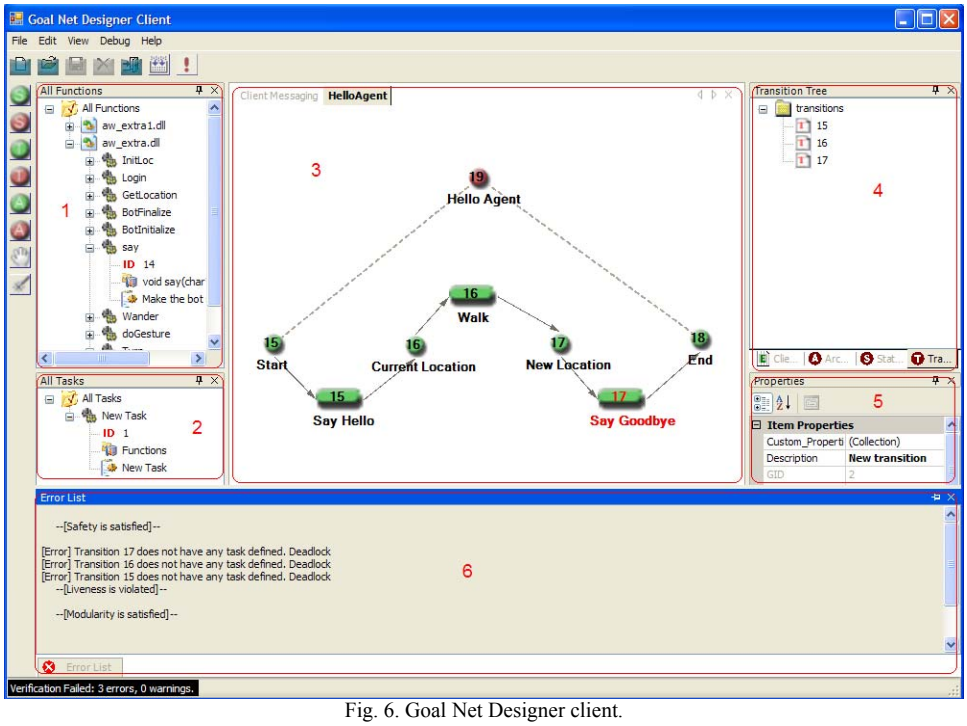

Fig. 6. Goal Net Designer client.

Area 2 is the task view which the users can modify. A task is basically a container which accommodates one or more functions organized into certain sequences by the user. When the MADE runtime encounters a task the functions within that task will be executed in the order that they were organized in. Users can create tasks in every transition. Once created, the task will be made available to all users for future reuse. A master copy of the task list is maintained at the server side. All active clients are updated via the callback mechanism.

Area 3 is the drawing board on which the users design the goal net for their agent or multi-agent system. Multiple drawing boards can be opened simultaneously and organized into tabs so that the users can work on more than one design at a time. The name and unique ID number of each goal and transition are overlaid on its graphical representation for easy reference. Drawing is carried out mainly through drag-and-drop operations. Basic building blocks of every goal net (i.e. simple state, composite state, transitions and arcs) are listed in the left toolbar. All the properties associated with each entity can be edited by right clicking on its image.

Area 4 shows the transitions, states and arcs of the goal net currently being edited. Their ID numbers are organized into three different tree views to highlight which entity is selected. Essentially, they offer a way for the users to lookup all the entities conveniently if the design gets overly complicated.

Area 5 hosts the property grid and client event log. The property grid lists all the editable and read-only properties related to each entity when an entity is selected. It is the place where detailed design of the goal net is carried out. The client event log is similar in functionalities to the server event log as mentioned above except it records events happening at the client side for future reference.

Area 6 contains the model checking module which provides verification for compliance to the safety, liveness and modularity requirements for a goal net design. To give the users a hassle-free experience in designing a goal net, the Goal Net Designer adopted an approach which allows the users to freely alter the positioning of the state and transitions on the design drawing to best suit the natural way of human interpretation of hierarchical structure. A goal net design is logically organized into a hybrid data structure consisting of multiple linked-lists and trees with states and transitions as nodes while arcs are pointers to the child/peer nodes. The users can simply initiate a verification process for a goal net by loading the design into the local machine and selecting the *Debug->Verify* menu item. The detailed diagnostic results will be shown in the verification result panel in Area 6.

As in the case of the Goal Net Designer server, the layout of the Goal Net Designer client can also be customized by the users in a flexible way.

## 6. BUILDING AN AGENT WITH THE GOAL NET DESIGNER

To demonstrate how the tool works, this section shows how a simple agent comes to life from the Goal Net design step-by-step. For the purpose of illustration, the agent should say “Hello”, walk to a new location and say “Goodbye” and then stop. The test bed used for this demonstration is the *Virtual Singapura* project which is based on the Active World online 3D environment so that the actual activities of the resulting agent can be directly observed.

The first step is the design of the goal net to model the mental states of the agent. As shown in Area 3 of Figure 6, the goal net consists of four distinct states (circles) each with a name that is self-explanatory. The arcs (arrows) indicate the flow of logic from one state to the next. To go from state *a* to state *b*, a transition (round-corner rectangle)

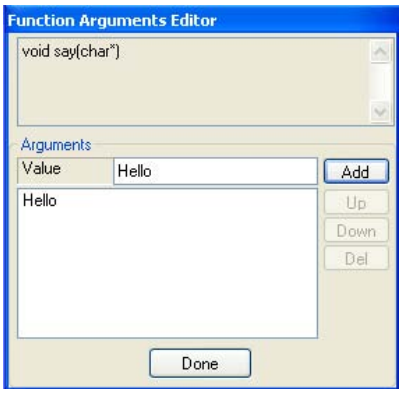


Fig. 7. Add a task to a transition.

must be passed through. It is in these transitions that the actions the agent must perform are associated with their actual implementation functions. The composite state (red circle 19) acts as a high level representation of the entire goal net. It can be reused by implanting itself into another goal net. The numbers overlaying on the graphical entities are their respective identification numbers. Each ID is unique within its own type domain but not across type boundaries (a state and a transition might have the same ID number).


Fig. 8. Agent controlled 3D avatars built with MADE.

Each state and transition has a set of properties (e.g. names, descriptions, etc.) which can be edited to facilitate a better understanding of the design by other users. Apart from these informational properties, other properties, for instance cost of going through a transition, can be specified by the users which will be modified during runtime to facilitate the execution of the default goal selection algorithm provided by the MADE runtime. Area 5 of Figure 6 illustrates a property pane with default values assigned to all fields. The designed goal net is saved into the database with a name after the designer finishes.

The last step is to create an agent and run it in the MADE runtime. The name of a goal net (which is checked for collision during design time and guaranteed to be unique) is passed to the agent as the argument. MADE runtime will create an agent, load the goal net from the database and interpret the goal net from the starting state. Whenever a defined function is encountered, it is invoked by pushing the function name and parameters to the system stack. Using this in our example, the agent controlling its 3D avatar in the *Virtual Singapura* project test bed will execute the actions as defined in the goal net. Figure 8 shows a screen shot of the agent controlled 3D characters within our virtual environment.

In virtual Singapura, we use player centered design. One of our goals is to let secondary school teachers as well as the students who do not have any agent programming knowledge to be able to create agents for various education and learning purposes.

## 7. QUANTITATIVE USABILITY STUDY OF THE MADE SYSTEM

The MADE system has been deployed to the *Virtual Singapura* project which aims to provide an agent-augmented 3D virtual online environment for secondary school students in Singapore to engage in immersive situated e-learning. The role of the MADE system is to "agentize" the existing virtual environment without introducing too much changes to its original design. One application scenario is to design an Interactive Storytelling (IS) system in the virtual environment for students to investigate the Dengue Fever plague in the 19$^{th}$ century Singapore. A heavy agent presence in the story is required to facilitate the unfolding of the storyline, provide hints to help the students to move forward and monitor the students' progress. During the design and development process, a team of final year computer engineering undergraduate and post-graduate students from Nanyang Technological University as well as secondary school teachers from the National Institute of Education, Singapore have been actively involved in using the MADE system. The role of the post-graduate students is mainly to guide the undergraduate students in using the MADE framework as well as programming in the 3D virtual online environment.

### 7.1 Objectives of the Usability Study

In order to offer a better perspective of the advantages of the MADE Framework, we conducted a usability study among the group of 7 post-graduate students and research staff who have had prior experience with other multi-agent system development frameworks (e.g. JADE). A questionnaire was given to them to investigate the users' attitudes toward the MADE Framework by considering the factors of perceived usefulness and perceived ease of use based on the Technology Acceptance Model (TAM) [Davis 1989].

### 7.2 Data Preparation

The survey consists of five-point Likert-type questions with an absolute value (Strongly Disagree = 1, Disagree = 2, Neutral = 3, Agree = 4, Strongly Agree = 5). A number of

Table I. Reliability Analysis for Variables with More than One Item

| Measurable Variables | Focused Aspects of the MADE Framework | No. of Items | Cronbach's α | |
|---|---|---|---|---|
| Perceived Usefulness | Saves development time | 2 | 0.727 | 0.92 |
| | Saves development effort | 2 | 0.727 | |
| | Improves the quality of work | 3 | 0.875 | |
| | Reduces the level of skills required | 2 | 0.8 | |
| Perceived Ease of Use | Intuitiveness of the GUI | 3 | 0.706 | 0.891 |
| | Effort required from users to interact with the GUI | 3 | 0.6 | |
| | Flexibility of the GUI | 2 | 0.6 | |
| | Help and guidance provided by the framework | 1 | N.A. | |

questions were used to measure perceived usefulness and perceived ease of use. Average values of responses to each question were obtained by summing the values by all the respondents and dividing the number of respondents. The No. of Items column in Table 1 represents the weight we assign to each aspect of the two measurable variables in order to compute the final perceived usefulness and perceived ease of use values. Table 1 shows that the Cronbach's α for this questionnaire exceeds the recommended 0.60, indicating good reliability of the survey items.

### 7.3 Findings and Analysis

According to the data collected from the questionnaire, we have calculated the perceived usefulness value to be 4.389/5 and the perceived ease of use value to be 4.167/5. According to TAM, the perceived ease of use, the perceived usefulness and the users' attitude forms a chain of causality (i.e. ease of use → usefulness → usage). Therefore, the high level of perceived ease of use and perceived usefulness achieved by the MADE Framework is likely to lead to its adoption by more users.

From this study based on a real development cycle, the MADE framework has demonstrated its ability to lower the technical barrier required to design and implement agent-augmented interactive digital media application by providing a powerful design assistance tool and hiding most of the implementation work. As long the existing system has an open architecture (i.e. with an open API and accepts third party function calls), it can be "agentized" without interference with the original system design. The MADE framework has received generally positive feedbacks from users with wide ranging levels of technical competency.

## 8. CONCLUSIONS AND FUTURE WORK

The MADE framework separates agent mental state modeling from task design in order to incorporate intelligent and autonomous behaviors into agents. The current prototype has achieved its design goal of providing a friendly user interface to facilitate better design experience, hiding the implementation process to save development time and cost, and automating the execution process to support the incorporation of multi-agent systems into an existing application without interfering with the original system design.

Through the real-world case study in Virtual Singapura, Goal Net designer has been proven to be an excellent tool for enabling novice users to model, design and implement agents for various interactive digital applications. The easy-to-use Goal Net designer tools not only brings the exciting experiences in exploring virtual world with agents but also gives students a positive image of synthesize Artificial Intelligence, programming and interactive digital media.

In future development of the framework, we will include more sophisticated agent environment models to simulate more complex interactions between the agents and the virtual world settings. In addition, to expand the realm of application of the framework, a trust model will also be incorporated to cater for higher level of security requirements for deployment in large scale and open online communities.

## REFERENCES


ADRIEN, C., STEPHANE, F., MANUEL, K. AND QUENTIN, L. SketchiXML: towards a multi-agent design tool for sketching user interfaces based on USIXML, Jean Vanderdonckt Université Catholique de Louvain, School of Management (IAG).

CHUNYAN, M., ANGELA, G., AND ZHONG HUA, Y. 2002. Agent that models, reasons and makes decisions, *Knowledge-Based Systems (KBS), 15*, 3.

CHUNYAN, M., ANGELA, G., YUAN, M., AND ZHONG HUA, Y. 2001. A dynamic inference model for intelligent agent. *International Journal of Software Engineering & Knowledge Engineering (IJSEKE), 11,* 5, 509--528.

CHUNYAN, M., ZHIQIANG, L., YUAN, M., AND ANGELA, G. 2003. Dynamic causal multi-agent infrastructure in large decision support system, *International Journal of Fuzzy Systems (IJFS), 2003.*

DAVIS, F.D. 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly* (Sept.), 319–340.

FABIO, B., GIOVANNI, C., TIZIANA, T., AND GIOVANNI, R. *JADE Programmer's Guide.* University of Parma.

JAELSON, C., MANUEL, K., AND JOHN, M. 2000. A requirements-driven software development methodology.Tropos Working Paper, Dept. of Computer Science, Univ. of Toronto.

PADGHAM, L., JOHN, T., AND WINIKOFF, M. 2005. Tool support for agent development using the Prometheus methodology. In Proceedings of the *Fifth International Conference on Quality Software* (QSIC 2005, R. Melbourne Institute of Technology Vic., Australia).

PENSERINI, L., PERINI, A., SUSI, A., AND MYLOPOULOS, J. 2006. From capability specifications to code for multi-agent software. In *Automated Software Engineering, 2006. ASE '06. Proceedings of the 21st IEEE/ACM International Conference,* 253-256.

PETER, L., CHUNYAN, M., AND BU-SUNG, L. Survey of agent-oriented software engineering for aervice-oriented computing, *International Journal of Web Engineering and Technology,* to appear.

PETER, L., CHUNYAN, M., AND BU-SUNG, L. An agent based software engineering methodology for service-oriented computing. *Multi-Agent and Grid Systems, an International Journal,* to appear.

ZHIQI, S. AND ROBERT, G. 2003. Goal-based intelligent agents. *International Journal of Information Technology 9*, 1, 19-30.

ZHIQI, S., CHUNYAN, M., AND ROBERT, G. 2006. Goal-oriented methodology for agent-oriented software engineering. *IEICE Transactions on Information and Systems E89-D*, 4 (April), 1413-1420.

ZHIQI, S., ROBERT, G., CHUNYAN, M., TIN-WEE, T., CHONG-SEAN, K., AND HUI-MIEN, L. 2005. Process modeling and automated execution for bio-manufacturing. *International Journal of Information Technology 12*, 1, 37-48.


Received July 2007; accepted April 2008