Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

INTRODUCTION TO THE VISUAL SIMULATION ENVIRONMENT

Osman Balci
Anders I. Bertelrud
Chuck M. Esterbrook
Richard E. Nance

Orca Computer, Inc.
Virginia Tech Corporate Research Center
1800 Kraft Drive, Suite 111
Blacksburg, Virginia 24060, U.S.A.

ABSTRACT

This paper introduces the Visual Simulation Environment (VSE). VSE has been developed under research funding, primarily from the U.S. Navy for over a decade. It enables discrete-event, general-purpose, object-oriented, picture-based, component-based, visual simulation model development and execution. This advanced environment can be used for solving complex problems in areas such as air traffic control and space systems, business processes and workflows, computer networks, education, health care, manufacturing, satellite communications, supply chain management, and transportation.

1. INTRODUCTION

The research in building a discrete-event Simulation Model Development Environment (SMDE) began in June 1983 at Virginia Tech under funding from the U.S. Navy (Balci and Nance 1987). Guided by the fundamental requirements identified by Balci (1986), incremental development, evolutionary prototyping, and rapid prototyping approaches have been used to develop the prototypes of SMDE tools. An overview of the SMDE architecture and prototype tools is given by Balci and Nance (1992). A Visual Simulation Support Environment (VSSE) research prototype was completed in April 1992 (Derrick and Balci 1995, 1997).

Based on the experience gained from the use of the SMDE and VSSE prototypes, development of the Visual Simulation Environment® (VSE) started in August 1992 under the object-oriented software engineering environment of the Unix-based NEXTSTEP operating system. A fully functional research prototype of VSE was developed at Virginia Tech in July 1995 (Balci et al. 1995). Technology transfer, enabled by the creation of Orca Computer, Inc. (Balci et al. 1997d), has produced the first commercial version of VSE in November 1996 for the NEXTSTEP operating system. Subsequently, VSE was released for OPENSTEP, Windows NT 4.0, and Windows 95. VSE will be released for the new Macintosh (Rhapsody) operating system in early 1998.

The purpose of this paper is to introduce the VSE software product developed as a result of experimental research for over a decade. Section 2 presents a brief overview of the object-oriented paradigm. Section 3 describes basic VSE modeling concepts. Sections 4 through 6 describe the VSE toolset. Concluding remarks are given in Section 7.

2. THE OBJECT-ORIENTED PARADIGM

The VSE is fully and truly object-oriented and requires understanding of the object-oriented paradigm (OOP). This section presents a brief overview of the OOP in terms of the VSE concepts.

An **object** is an element of interest in a system being modeled. Each object possesses some characteristics, performs some services, and exhibits some behavior. Objects are the building blocks of VSE models.

A **class** is a grouping or categorization of objects with the same characteristics, services, and behaviors. A class may be extended into other classes. Extending a class is called *subclassing* and an extended class is called a *subclass*. A subclass inherits all the characteristics, services, and behaviors of the parent class. It customizes what it inherits and/or provides more characteristics, services, or behaviors. The parent of a subclass is called the *superclass*. The class at the top level (having no superclass) is called the *root class*.

The standard built-in VSE class library is called *VSLibrary*. All VSE models are developed by subclassing from classes in the VSLibrary class hierarchy. A class hierarchy of a simulation model of a traffic intersection in Blacksburg, Virginia is partially shown in Figure 1. For example, the selected EnterIntersection class in Figure 1 is a user-defined class inheriting all characteristics and behavior of superclass Area which inherits from

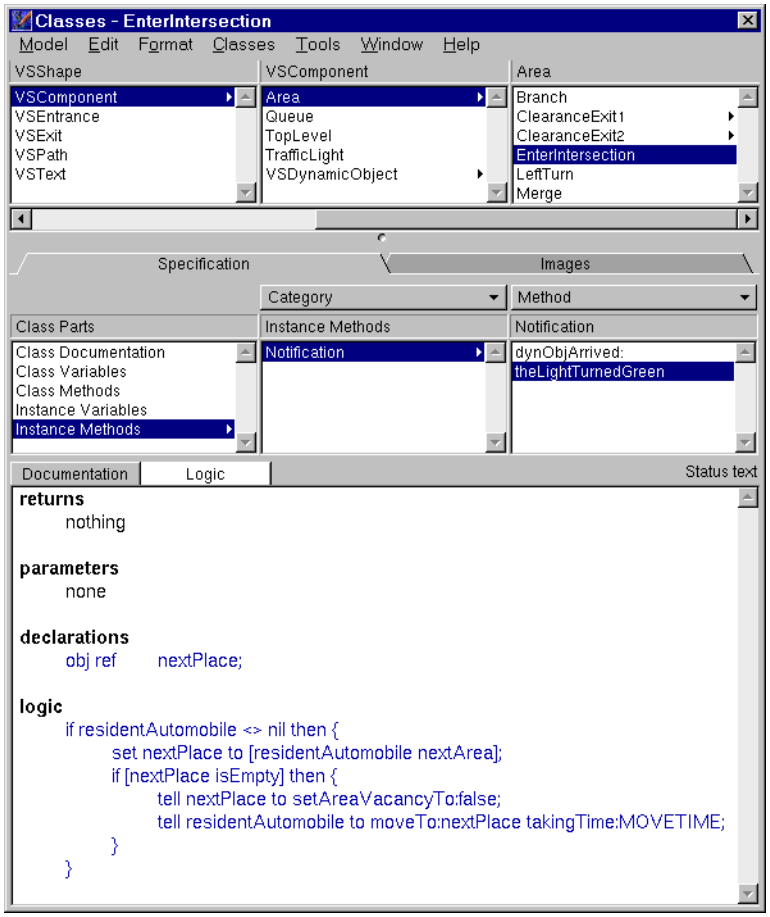

Figure 1: Class Hierarchy of a Traffic Intersection Model

VSComponent which inherits from VSShape which inherits from VSObject — the root class. A VSE class is composed of the following parts: class documentation, class variables, class methods, instance variables, instance methods and class images as shown in Figure 1.

Creation of an object belonging to a class is called **instantiation**. The new object inherits all characteristics (instance variables) and behaviors (instance methods) specified in the class from which it is instantiated. Instance variables of a class are created for each object instantiated as a member of that class. Instance methods are inherited, but no method code is replicated.

Three kinds of **variables** exist under VSE: class variables, instance variables, and local variables. *Class variables* are attributes of a class and are declared in the class for use by the methods of that class and its subclasses and by the objects instantiated from that class and its subclasses. *Instance variables*, declared in a class, are used by the instance methods of that class and are created for each object instantiated as belonging to that class or any of its subclasses, i.e., for each instance of a class, and hence the designator "instance." *Local variables* are declared within a method for use only during the execution of that method. On completion of a method, all local variable values are lost.

VSE variables can have the following data types: boolean, character, class reference, object reference, integer number, real number, and enumeration.

Services provided and behaviors exhibited by an object are specified in **methods**. Two types of methods exist: class methods and instance methods. *Class methods* are used to provide services specific to a class. For example, each VSLibrary class, by inheriting from VSObject root class, provides the method "new" which creates an instance of that class. *Instance methods*, given in a class, are used to specify the services provided and behavior exhibited for each object instantiated from that class. Each instance created as belonging to a class provides the services and behavior specified in the instance methods of that class.

The method code is specified only once in the class and is not replicated on each instantiation of an object from that class. Therefore, model maintainability is significantly facilitated since a potential change is localized to only one method when hundreds of objects may exhibit the method behavior.

A VSE method is composed of the following parts as shown in Figure 1. *Returns:* specifies the value type returned by the method, if any. *Parameters:* contains the declarations of the method's input parameters. *Declarations:* contains the declarations of all local variables. *Logic:* contains the logic of the method in the VSE object-oriented scripting language.

When an object is (a) declared as member of and (b) assigned to a class, it inherits the characteristics (instance variables) and behaviors (instance methods) of that class as well as the characteristics and behaviors of that class's superclass, and any ancestral classes, tracing back to the root class of the VSLibrary which is VSObject. **Inheritance** significantly facilitates reusability of earlier developed classes and decreases model development time.

All instantiated objects communicate with each other via **message passing**. Sending a message to an object implies the invocation of one of the receiving object's methods. Generally, it is said that "send message M to object A" as opposed to "send a message to object A to invoke its method M." VSE objects are identified by their unique addresses internally maintained by VSE. They are called the *object references*. An object reference is used to specify the object which receives the message.

Figure 1 shows that a traffic intersection area belonging to EnterIntersection class can be sent *theLightTurnedGreen* message upon which the following method

logic is executed. If a vehicle is waiting on the area that receives *theLightTurnedGreen* message, then *nextArea* message is sent to the resident vehicle and the returned object reference is stored into the nextPlace local variable. Then, *isEmpty* message is sent to the area pointed to by the nextPlace object reference. If the returned value is true, then *setAreaVacancyTo:* message is sent to nextPlace with parameter false and *moveTo:takingTime:* message is sent to the resident vehicle with parameters nextPlace object reference and MOVETIME constant.

The methods of an object, specified in the object's class, describe the services provided and behaviors exhibited by that object. Any object belonging to a subclass of the object's class can access the attributes of the object directly. All other objects must request the object's service or trigger its behavior by sending a message. How an object provides a service or exhibits a particular behavior is completely hidden from the rest of the world. Only through message passing, the services of an object can be requested. This feature of the OOP is called **Encapsulation** which means that the object hides its implementation from the caller objects requesting its services via message passing.

Polymorphism is the ability of objects of different classes to exhibit different behavior in response to the same message. For example, sending the *computeArea* message to an object of the Circle class would invoke a different algorithm from one invoked when sending *computeArea* to an object of the Square class. This means that an algorithm that operates on a heterogeneous set of objects does not need to consider the classes to which the objects belong — it simply sends a message to perform the desired action, and each object handles it as appropriate depending on its class.

Association is a property that ties an object in the system to one in the model of that system. The Principle of Association advocates that each system object of importance, based on the objectives of the modeling study, should have a direct correspondence to an object in the model.

3. BASIC CONCEPTS OF VSE MODELING

The VSE model architecture consists of static and dynamic parts. The *model static architecture* is composed of hierarchically decomposed *components* (Balci et al. 1997a). The *model dynamic architecture* is made up of dynamic objects. A *dynamic object* is an entity of interest which physically or logically moves from one point to another in a model. For example, bus, ship, aircraft, passenger, train, and computer job might be represented as dynamic objects. A dynamic object may be decomposed into a hierarchy of components similar to the model static architecture decomposition (Balci et al. 1997b).

A *component* is a part of the model static architecture or a part of the structure of a dynamic object. For example, the USA map can be the top-level component of a model which is decomposed into 50 other components, each representing a state. A component representing a state, e.g., California, can be further decomposed into components representing cities. Each city component can be decomposed into a component showing at least one airport. Each airport component can be decomposed into other components such as the passenger terminal, runways, and control tower. (http://www.OrcaComputer.com/VSE/Examples/Examples.html)

VSE provides two kinds of components: shallow and deep. A *shallow component* is one that has no layout (or no decomposition). A *deep component* is one that has a layout (or a decomposition). Shallow components represent the leaf nodes (the nodes which are not further decomposed) of the hierarchical decomposition. Deep components represent those nodes of the hierarchy which are decomposed further.

A deep component has a graphical layout which has a size and color, and may be composed of objects that may display images in EPS or TIFF formats. Images are dragged and dropped into the component layout in the VSE Editor, after which they can be resized and positioned. For example, an aerial photograph of the Prices Fork Road and Toms Creek Road traffic intersection in Blacksburg, Virginia is obtained from the Town of Blacksburg. Its scanned and cleaned TIFF image is dragged and dropped into the VSE Editor as the graphical layout of the top level model component as shown in Figure 3. Any portion of the TIFF image can be designated as a deep component (decomposition) or a shallow component.

A component can be created as part of the model static architecture in the VSE Editor. It can also be created, manipulated, and destroyed during model execution by specifying the component as a template. At run-time, a component can be instantiated from a template. For example, in a Global Air Traffic simulation model (http://www.OrcaComputer.com/VSE/Examples/AirTraffic.html), 200 deep components representing cities are instantiated at run-time (by reading the city data from an input text file) and placed on the world map top-level component. Hence, parts of the static model architecture can be created at run-time as well as in the VSE Editor at design time.

A distinctive capability of VSE is the flexibility given to modeling with dynamic objects. For example, a dynamic object representing an aircraft can be decomposed into its inside view component which can be further decomposed into components representing cockpit, service area, passenger area, and cargo area. The

passenger area component can be decomposed into seats. Pilots, attendants, and passengers can be represented as dynamic objects. A passenger dynamic object can enter into an aircraft dynamic object, move within its component hierarchy, and occupy a seat component. All dynamic objects which enter into an aircraft dynamic object then move together with the aircraft to an airport component of the model. In essence, their collective behavior is represented by the enclosing dynamic object (aircraft) but each passenger object retains its individual representation. Balci et al. (1997b) presents more detail about dynamic object decomposition in VSE.

A dynamic object can be created as part of the model dynamic architecture in the VSE Editor to represent the initial conditions at time zero. Typically, a dynamic object is created, manipulated, and destroyed during model execution by specifying it as a template and instantiating it from that template.

Specification of dynamic object components is carried out in exactly the same manner as the specification of model static architecture components.

4. VSE EDITOR

The VSE Editor is used to create a model specification and to automatically translate it into executable code. It enables the specification of a model by using the ten structural model parts shown in Figure 2.

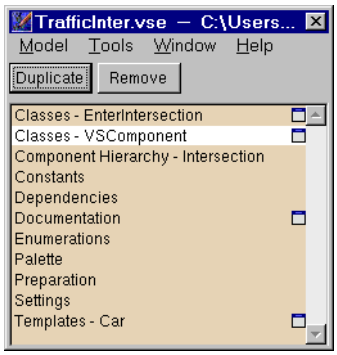

Figure 2: VSE Editor Model Main Window

Classes: Figure 2 shows two class windows active. The classes window is shown in Figure 1. The top part shows the class hierarchy browser. The classes with names starting with VS prefix are built-in standard VSE classes. They belong to the VSLibrary and are unmodifiable by the user. The VSLibrary is the library on which all VSE models depend. Users create their own classes by subclassing from the VSLibrary or from any other library a model is made to depend on by using the Dependencies model part shown in Figure 2.

The lower part of the classes window is used for documentation, declaration of class variables and instance variables, logic specification of class methods and instance methods, and specification of images used in the class and its subclasses. Class methods and instance methods are categorized for organizational purposes. The category column lists the categories of methods (Notification) for the corresponding class (theLightTurnedGreen). The categorization has no impact on the model logic and is created and named by the user.

A method is specified in four sections: `returns`, `parameters`, `declarations`, and `logic`. The Returns section specifies the type of the value returned by the method, if any. If no value is returned by the method, `nothing` is specified. The Parameters section is used to declare the method's input parameters. The method `dynObjArrived:` in Figure 1 contains a colon implying that it has one input parameter. This parameter is an object reference pointing to the dynamic object that arrived in the component receiving this message. The declarations section is used to declare local variables of the method. The logic section is used to specify the method logic by using the VSE Object-Oriented Scripting Language.

Images used in the class and its subclasses are specified under the Images tab as shown in Figure 1.

Component Hierarchy: The model component hierarchy window is shown in Figure 3. This is a split window. The top part shows the browser which is used to browse through the hierarchical graphical model architecture. Intersection is the name of the top level model component which is decomposed into deep and shallow components. Only "Queue 8", "Queue 9", and "Traffic Light" deep components are visible in Figure 3. A deep component's layout can be displayed by clicking its name in the browser or by double clicking its graphical representation in the layout.

The bottom part of the split window is called the *layout* and is used to specify a graphical representation for a deep component. The layout box is scrollable in x-y directions and its type (deep or shallow), size and color can be changed by using the Layout Inspector. The minimum layout size is 10 by 10 pixel elements, maximum layout size is 4,000 by 4,000 pixel elements, and the default is 1,000 by 1,000 pixel elements.

Constants: A *constant* is a value which never changes during simulation model execution. All constants in a VSE model are specified in the Constants window.

Dependencies: Earlier developed and tested model components can be reused in building a new VSE model. A repository of reusable model components is called a *library.* A model is specified to depend on a library by using the Dependencies model main window item. Each library may be assigned a color so that its classes can be visually distinguished in the classes window. When a library is assigned a color, the names of its classes, class parts, categories, and methods are all displayed in that color. Balci et al. (1997c) presents more detail about developing a library of reusable model components by using VSE.

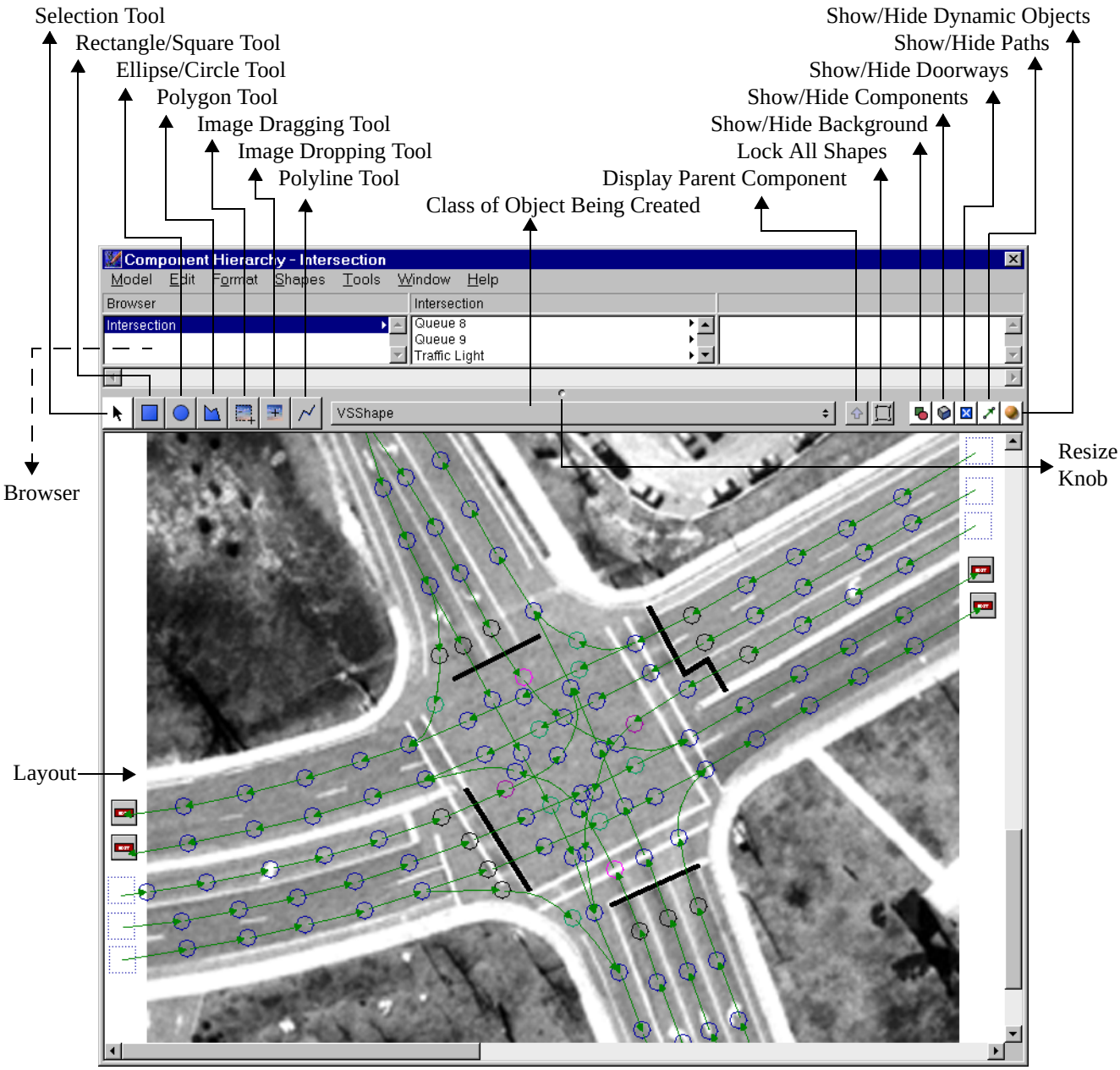

Figure 3: VSE Editor Component Hierarchy Window

Documentation: A high-level model documentation is specified using the Documentation window. The classes window in Figure 1 also provides the capability to document classes and methods. In-line and block documentations are also provided in methods.

Enumerations: are unique types that have a set of constant values. Similar to `integer`, `real` or `boolean` variable types, a variable can be created to be of an enumeration type named by the user with its set of constant values. All enumeration types and their associated set of constant values are specified in the Enumerations window. Appropriate use of enumerations increases the readability and understandability of a model.

Palette: provides reusable model components (objects). Whenever an object is created in the Templates window, that object automatically appears on the reusable objects palette. Any object shown in the palette can be dragged and dropped into a layout or a template. The Palette is provided for the sole purpose of reusability.

Preparation: provides four services for a model: simulate, prepare only, simulate only, and clean.

Settings: is used to specify model settings.

Templates: This window is used to create dynamic object and component templates from which objects can be instantiated at design time or run time. This window is similar to the component hierarchy window.

5. VSE SIMULATOR

The VSE Simulator provides an environment for execution, animation and experimentation. It is a stand-alone application created to run the simulation models developed using the VSE Editor. The user can modify the values of instance variables using the VSE Simulator's Inspector and perform experiments under different experimental conditions. Therefore, a model can be developed and sold/distributed to other users requiring only the VSE Simulator to perform experiments and obtain results by changing the values of model parameters (instance variables). The VSE Simulator is sold as a stand-alone product for this purpose.

The VSE Simulator model main window is shown in Figure 5 (the small window with title "TrafficInter.vse") and contains five window types as described below.

Simulation Control: window is shown in Figure 4. An experiment is designed using this window. It enables the specification of a variety of strategies for data collection from a model including the method of replications, method of batch means, and the regenerative method.

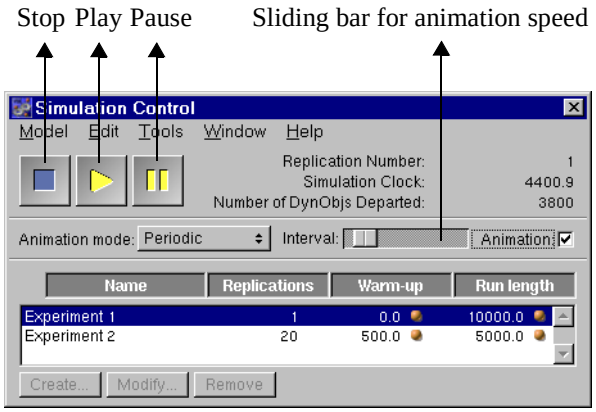

Figure 4: Simulation Control During Animation

Runtime Error: When a runtime error occurs, the Runtime error window is displayed and the model execution halts. A description of the problem is given. The name of the method where the error occurred is listed together with its class name and line number. The name of the method that called the method where the error occurred is also given together with its class name and line number of the call. The object that produced the error is highlighted with a red box.

Method Trace: Method tracing is provided for model verification and validation purposes. During debugging, it can be enabled to trace message passing. If method tracing is enabled and a runtime error occurs, the method trace window shows the last method invocations, one of which causes the problem. VSE Simulator provides the mapping of runtime errors back to the model specification. Double clicking a method name in the method trace window launches the VSE Editor, opens the model, and displays the method where the error was detected.

Transcript: window shows the output generated by print statements in the model. It is used for model verification, validation and testing. Specifically, during debugging, the model may be instrumented, i.e., a piece of code (trap code, probe) is inserted at different logical points in the model for the purpose of writing out intermediate results to the Transcript in order to trace the control flow and/or data flow of the model. Then the user analyzes the Transcript contents to identify the sources of execution problems.

Component Viewer: Figure 5 shows the animation of a traffic intersection in Blacksburg, Virginia. Only one component viewer is shown in this animation. By clicking the New Viewer command button in the model main window, more viewers can be created and displayed on the screen. An unlimited number of component viewers can technically be displayed enabling the user to see the animations of many different components of a model simultaneously. However, monitor size and main memory restrict the number of viewers. It is technically possible to connect 2 or more monitors to the same PC to open more viewers. Orca has an Intel-based PC with two 17" color monitors providing continuous screen space from one monitor to the other so that more viewers can be displayed during animation.

6. VSE OUTPUT ANALYZER

The VSE Output Analyzer is used to obtain general statistics and to construct confidence intervals for simulation output data. The Output Analyzer can open: VSE model files (with extension ".vse"), VSE output data files (with extension ".vseout"), RTF (Rich Text Format) files, or text files. A new data file can be created or an existing one can be opened. In creating a new data file, the data values must be separated by at least one space. Both integer and real values can be entered. Comments are entered starting with "//". Anything between "//" and the next line break is ignored in the processing of the data file. Different font families, styles, and sizes can be selected using the Font Panel dialog box. Anything in the data file can be colored using the Colors dialog box.

7. CONCLUDING REMARKS

The VSE technology enables discrete-event, domain-independent, object-oriented, picture-based, component-based, visual simulation model development and execution. This advanced environment can be used for solving complex problems in areas such as air traffic control and space systems, business processes and workflows,

Figure 5: Visual Simulation of Prices Fork Road and Toms Creek Road Traffic Intersection in Blacksburg, Virginia

computer networks, education, health care, manufacturing, satellite communications, supply chain management, and transportation. VSE can be used to represent different areas, such as supply chain management, manufacturing, and business processes, all in the same model due to its general-purpose nature.

The VSE technology enables the componentization of visual simulation model development with its fully and truly object-oriented capability inherent to its conceptual framework. Model factories can be established that manufacture model components that can be reused by others in the development of a VSE model. (Balci et al. 1997c)

The VSE technology enables the establishment of component-based simulation modeling marketplace so that the customers of the simulation industry can observe large economic benefits such as reduced costs, increased quality, and interoperation.

The VSE technology increases automation and productivity in simulation model development by enabling: (a) quality and reliability improvements in simulation models, (b) reduced time to develop and test simulation models, and (c) cost amortization through simulation model component reuse.

The VSE technology increases the productivity of simulation modelers by enabling increased quality through specialization and improved focus on problem solving instead of simulation programming.

The VSE technology broadens simulation markets for model producers by enabling: (a) creation of systematically reusable simulation model components, (b) increased simulation model and other software interoperation, and (c) easy adaptation for use in international markets. (Balci et al. 1997c)

ACKNOWLEDGMENTS

The research that has led to the creation of the Visual Simulation Environment technology since 1983 has been sponsored in part by the U.S. Navy under research grants totaling $1.2 million. Visual Simulation Environment is a registered trademark of Orca Computer, Inc. (http://www.OrcaComputer.com)

REFERENCES

Balci, O. 1986. Requirements for model development environments. *Computers & Operations Research* 13:53-67.

Balci, O., A. I. Bertelrud, C. M. Esterbrook, and R. E. Nance. 1995. A picture-based object-oriented visual simulation environment. In *Proceedings of the 1995 Winter Simulation Conference*, ed. W. R. Lilegdon, D. Goldsman, C. Alexopoulos, and K. Kang, 1333-1340. IEEE, Piscataway, NJ.

Balci, O., A. I. Bertelrud, C. M. Esterbrook, and R. E. Nance. 1997a. The visual simulation environment. In *Proceedings of the 11th European Simulation Multiconference*, ed. A. R. Kaylan and A. Lehman, 61-68. SCS, San Diego, CA.

Balci, O., A. I. Bertelrud, C. M. Esterbrook, and R. E. Nance. 1997b. Dynamic object decomposition in the visual simulation environment. In *Proceedings of the 11th European Simulation Multiconference*, ed. A. R. Kaylan and A. Lehman, 69-73. SCS, San Diego, CA.

Balci, O., A. I. Bertelrud, C. M. Esterbrook, and R. E. Nance. 1997c. Developing a library of reusable model components by using the visual simulation environment. In *Proceedings of the 1997 Summer Computer Simulation Conference*, to appear. SCS, San Diego, CA.

Balci, O., A. I. Bertelrud, C. M. Esterbrook, and R. E. Nance. 1997d. The visual simulation environment technology transfer. In *Proceedings of the 1997 Winter Simulation Conference*, ed. S. Andradóttir, K. J. Healy, D. Withers, and B. L. Nelson. IEEE, Piscataway, NJ.

Balci, O., and R. E. Nance. 1987. Simulation model development environments: a research prototype. *Journal of Operational Research Society* 38:753-763.

Balci, O., and R. E. Nance. 1992. The simulation model development environment: an overview. In *Proceedings of the 1992 Winter Simulation Conference*, ed. R. C. Crain, J. R. Wilson, J. J. Swain, and D. Goldsman, 726-736. IEEE, Piscataway, NJ.

Derrick, E. J., and O. Balci. 1995. A visual simulation support environment based on the DOMINO conceptual framework. *Journal of Systems and Software* 31: 215-237.

Derrick, E. J., and O. Balci. 1997. DOMINO: a multifaceted conceptual framework for visual simulation modeling. *INFOR – Canadian Journal of Operational Research and Information Processing* 35.

AUTHOR BIOGRAPHIES

OSMAN BALCI is an Associate Professor of Computer Science at Virginia Tech and President of Orca Computer, Inc., developer of the Visual Simulation Environment. He received B.S. and M.S. degrees from Bogazici University in 1975 and 1977, and M.S. and Ph.D. degrees from Syracuse University in 1978 and 1981. Dr. Balci is the Editor-in-Chief of two international journals: *Annals of Software Engineering* and *World Wide Web*; Verification, Validation and Accreditation (VV&A) Area Editor of *ACM Transactions on Modeling and Computer Simulation;* Simulation and Modeling Category Editor of *ACM Computing Reviews;* Associate Editor of *INFORMS Journal on Computing;* and serves on five other editorial boards. He is currently a member of the Defense Modeling and Simulation Office (DMSO) VV&A technical working group. Dr. Balci has been a PI or Co-PI on research grants and contracts sponsored by the U.S. Navy with a total funding of $1.2 million. His current research interests center on software engineering, visual simulation and modeling, and world wide web. Dr. Balci is a member of Alpha Pi Mu, Sigma Xi, Upsilon Pi Epsilon, ACM, IEEE CS, INFORMS, and SCS.

ANDERS I. BERTELRUD is a Vice President of Orca Computer, Inc., developer of the Visual Simulation Environment (VSE). He received B.S. and M.S. degrees in Computer Science from Virginia Tech in 1993 and 1995. He is a member of Phi Beta Kappa, Upsilon Pi Epsilon, and ACM. He has been working on the development of VSE since September 1992.

CHUCK M. ESTERBROOK is a Vice President of Orca Computer, Inc., developer of the Visual Simulation Environment (VSE). He received a B.S. degree in Computer Science from Virginia Tech in 1996. He has been working on the development of VSE since September 1992.

RICHARD E. NANCE is the RADM John Adolphus Dahlgren Professor of Computer Science and the Director of the Systems Research Center at Virginia Tech. He is also Chairman of the Board of Orca Computer, Inc. He received B.S. and M.S. degrees from N.C. State University in 1962 and 1966, and a Ph.D. degree from Purdue University in 1968. Dr. Nance was the founding Editor-in-Chief of the *ACM Transactions on Modeling and Computer Simulation* (1990-96). He served as Program Chair for the 1990 Winter Simulation Conference. Dr. Nance has received awards from the TIMS College on Simulation and ACM SIGSIM. He is a member of Alpha Pi Mu, Sigma Xi, Upsilon Pi Epsilon, ACM, IIE, and INFORMS.