Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

CONTROLLING SIMULATION GAMES THROUGH RULE-BASED SCENARIOS

Stijn-Pieter van Houten
Alexander Verbraeck

Section of Systems Engineering
Delft University of Technology
Delft, THE NETHERLANDS

ABSTRACT

In this paper we present a framework for scenarios in simulation games. It is used to support developing, using, and managing complex and dynamic simulation games, and it supports the achievement of the game's learning goal. Especially game facilitation is increasingly challenging due to the nature of these simulation games. The framework consists of a number of conditions, rules and actions, based on the concept of production systems from artificial intelligence, and it is presented as a component of Zeigler's experimental frame. This has been successfully used in the Distributor Game, which is the first of a series of management games developed for today's supply chain management challenges. Further research will focus on the extension of the set of conditions, rules and actions, and on visualizing and managing the interdependencies between the simulation game, its scenario, and the players.

1 INTRODUCTION

Within simulation games, we see a tendency to develop increasingly complex and dynamic simulation games to stimulate the creation of awareness of and training for real-world systems (Chiesl 1990, Dasgupta 2003, Gold and Pray 2001, Lainema and Makkonen 2003). These games may be used to support the exploration of a variety of decision making strategies. From a modeling and game play perspective, in these games, more interdependencies need to be modeled, which may lead to unexpected (and unwanted) behavior of the model during game play. Furthermore, modern simulation games are often continuous, as opposed to batch-based games, leading to more dynamic simulation games, which are more difficult to model and to manage.

In order to confront the player with a number of situations that are important to reach the learning goal, scenarios are often used that create the right state of the model at the right moment under the right conditions (Magerko and Laird 2002). A scenario consists of a set of predefined events that occur during game play. From a modeling perspective, a scenario does not seem much different from an experimental design. It involves setting the parameters on the computer-controlled parts of a simulation model, which in case of a simulation game also happens to have humans inside the model. When the desired state of the simulation game does not occur, a game facilitator usually intervenes and undertakes some kind of actions, for example the execution of some additional events, to achieve the desired state of the model.

Based on our experience with 4 game sessions for supply chain management, with a total of 120 (Executive) MBA students, we observed that the development and usage of scenarios for simulation games can be improved on a number of points. The first point deals with ensuring that a proposed effect, i.e. the goal of the scenario, does take place. Making sure, for example, that the bullwhip effect occurs is already difficult enough. Games designed for this effect, such as the Beer Game (Sterman 1989), typically involve a four-stage supply chain, have a limited number of players and have limited interaction. Furthermore, these games are often batch-based. The type of simulation games we develop and use vary in the number of players (for example 12 - 40 players) and have a large number of computer-controlled supply chain actors ($>$ 100) (van Houten et al. 2005). This leads to an increase of the (unpredictable) interaction between players and an increase in the complexity due to the larger number of actors in the simulation game. Hence, the design of scenarios becomes far from trivial. Secondly, we see that due to the continuous character of our games, and the increased complexity, it becomes very difficult for a game facilitator to manage a game at a certain moment of time and to predict over time the effects of actions that are executed. Thirdly, given that these games may be played over a long period of time, for example many days or even a whole semester, and in a globally distributed setting, the management of these games by a game facilitator and the introduction of interventions becomes unfeasible.

In this paper we present a framework that supports the development and usage of scenarios and hence supports simulation games. In Section 2 we present background information with regards to scenarios, based on the concept of production systems from artificial intelligence. Section 3 places the scenario framework into Zeigler's framework for modeling and simulation (Zeigler, Praehofer and Kim 2000). Measuring the effectiveness of a framework for scenarios is presented in more detail in Section 4. We describe and illustrate the framework in Section 5. In Section 6 we give our proof of concept by illustrating how the framework has been applied in the Distributor Game, which is a game developed for teaching supply chain management concepts for global supply chains (van Houten et al. 2005). We conclude this paper with conclusions and directions for further research.

2 RULE-BASED SCENARIOS

In simulation games, the term "scenario" is often used, although the term is rarely defined. Scenarios usually describe the context in which the game takes place, the desired development of the game over time, and some events that take place during game play to enhance the learning. Scenario's can therefore be seen as a scheme for game play, or as a script.

The term "scenario" was taken from theatrical studies, later the concept was introduced into research on military and strategic gaming (Jarke, Bui, and Carroll 1998). Nowadays, this term appears in in a wide number of fields, ranging from software and system engineering to human computer interaction. Many definitions of a scenario exist (Nardi 1992), however, here we choose to follow the definition of a *goal-based* scenario given by Schank et al. (1993), given its appropriateness for simulation games. They describe goal-based scenarios as comprising a clear, concrete goal to be achieved, a set of target skills to be learned and practiced in the service of this goal and a task environment in which to work. Goal-based scenarios are especially appropriate for generating an understanding of complex systems. They allow decision makers systematically to refine their understanding and intuition of system behavior through exploration and iterative experimentation. By iteratively making decisions, observing the effects of their decisions, and refining those decisions, decision makers develop an intuition for how the system under investigation behaves and how to diagnose and optimize their real-world systems. We see that a scenario as such is a description of a complex future situation, of which the occurrence cannot be predicted for sure. Nor can we formulate the path that could lead from the present to the desired future state of the system (Gausemeier, Fink, and Schlake 1996).

The most important condition for success in the use of educational simulation games is that the teaching objectives are met. This is not always easy, because players have a certain freedom, and their actions in the game can not always be predicted. Therefore, games almost always have a *game facilitator*, who takes actions when the game execution deviates from its intended path (Hall 1994). The concept of taking corrective actions towards a certain goal links perfectly with the definition of a scenario as presented above. When simulation games get more complex, when more players participate in a game, and when players play from different locations, facilitation quickly becomes more difficult. One of the solutions might be to automate some of the (problem solving) tasks of a game facilitator.

Artificial intelligence literature shows the value of production systems (Newell and Simon 1972, Rich 1983, Winston 1984) for mimicking human problem solving behavior. Rich (1983) defines a production system as a set of rules, each containing a left hand side that represents the applicability of the rule, a condition or a pattern, and a right hand side that describes the action to be performed if the rule is applied. In addition, a production system contains a database with information appropriate for the particular task. Finally, a control strategy is needed that describes the order in which the rules will be compared to the database and a way of resolving conflicts when more than one rule matches at the same time. Winston (1984) extends the database concept with a working memory as we also see it in blackboard systems (Engelmore and Morgan 1988), where production rules can also write new information into memory, and where unused information slowly ages until it is discarded for further use. Production systems can be used for plain actions, if rule applies, carry out action; for problem solving, if problem occurs, search for action that makes the problem go away; and for goal satisfaction, see how a desired state can be used from the current state using the production rules. A simulation game can benefit from all these forms of behavior, and the state of the simulation model shows a neat correspondence to the 'database' or 'memory' that is needed for the left hand side of the production rules.

To illustrate the concept, let us look at a few common issues that a game facilitator might encounter during game play, and the production rules that can help address them.

- *Example 1:* In a supply chain management game, one of the player teams makes the mistake of buying too many materials, which cannot be sold. The team gets into financial trouble, and cannot continue to play. Clearly, it is not in the interest of the teaching purpose of the game to disallow this team to continue. A game facilitator interferes, for instance by granting a loan. When the game is played with hundreds of players in different locations, the request to automate this task becomes

clear. The rule that would apply here looks as follows:

```
player.bankaccount < 10000 →
   player.bankaccount += 50000
```

- *Example 2:* The "script" of a supply chain management game prescribes that after 2 months of game play, i.e. game time, not clock time, prices of the products "Desktop" and "Laptop" should increase by 10%. Normally, the game facilitator should continuously watch the clock during the game, to initiate the right actions at the right time, and often these interventions are forgotten or executed too late. When automating this task, the rule would look as follows:

```
gametime == 2 months →
   laptop.price *= 1.1
   desktop.price *= 1.1
```

- *Example 3:* When we execute the task of example 2, we would also like to inform the players that a price increase has taken place, enabling them to adapt their strategies and reach one of the teaching goals as a result:

```
gametime == 2 months →
   sendmessage("Important
message:
   Prices of Laptops and
   Desktops increase by 10%")
```

- *Example 4:* During the run of a supply chain management game, the game facilitator finds out that there is a risk that the learning goal can not be met, because the customer market is not big enough to sustain the number of players. Increasing the demands of the customers would be an option here. However, the market should also not get too big, as this would make it too easy on the players. The rule could look as follows, looking at the average monthly profit for each player, and increasing or decreasing the market when the average profit is not between 5000 and 15000:

```
average(player.monthlyprofit) < 5000
→
   market.demand(desktop) *= 1.1
   market.demand(laptop) *= 1.1

average(player.monthlyprofit) > 15000
→
   market.demand(desktop) *= 0.9
   market.demand(laptop) *= 0.9
```

For many games, there might be hundreds of potential interventions for a game facilitator. With a growing number of players in the simulation game, it becomes almost impossible to look at all potential variables for intervention. A clear, automated scenario shifts some of the interventions to the scenario, and can help the game facilitator to reach the teaching objectives more easily.

3 SCENARIOS AND THE EXPERIMENTAL FRAME

A simulation game can be viewed as a special type of simulation model, where some of the actions in the model are executed by human players. As the execution of the simulation model is *influenced* by the game facilitator and by the scenario, we can view the scenario and the game facilitator as a part of the experimental frame as described by Zeigler, Praehofer, and Kim (2002). Davis and Anderson (2003, p.77) regard the experimental frame as a system in itself, which interacts with the simulation, the real-world system, and possibly other simulations. Specifying an experimental frame includes the way in which users, in our case the game facilitator, will operate on output data to generate whatever they actually need for their applications. The relation between the simulation model, simulator, experimental frame, players, scenario, and the game facilitator is illustrated in Figure 1.

We consider two types of input in this particular case. The first type is a collection of data, control conditions, and initialization conditions for a simulation model (Zeigler, Praehofer, and Kim 2002). The scenario is partly included in this type of input since it contains the initial state of a simulation model, for example a number of initial events for the event list. The second type of input is provided by a game facilitator. During game play, a game facilitator provides input to the simulation model using interventions. The output of a simulation model is sent to an experimental frame and may be collected for analysis by a game facilitator.

Having made clear where a scenario is positioned with regards to a simulation model and the kind of input and output related to a scenario, in the next section we present how to measure the effectiveness of our framework.

4 EFFECTIVENESS OF A FRAMEWORK FOR SCENARIOS

The effectiveness of a scenario framework can be described in terms of the *usefulness* of the framework, for example the value that it adds to the goal of supporting simulation games, the *usability*, for example the mesh between users, players and a game facilitator, processes and technology, and the *usage*, for example its flexibility, its adaptivity and its suitability to the context of a simulation game (Keen and Sol 2006). Furthermore, we consider the different activities and roles, for example a developer or a game

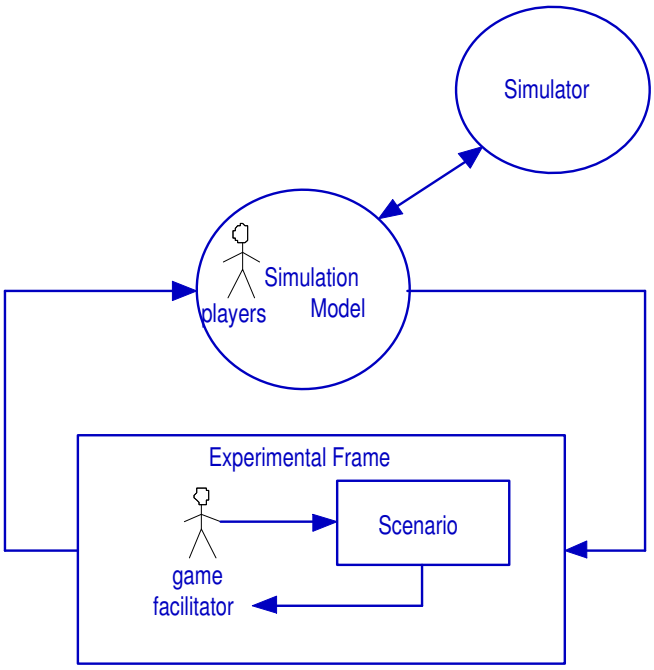

Figure 1: Relations Between Experimental Frame, Players, Game Facilitator and Scenario (Based on the Framework from Zeigler, Praehofer, and Kim 2002)

facilitator, involved in these activities while developing and using scenarios for simulation games.

- **Usefulness:** The first item related to effectiveness is that of providing a credible simulation game context to players. The framework should support this credibility in terms of complexity and dynamics. Furthermore, the framework should support players making decisions on their own (Magerko and Laird 2002, Thorsteinsson 1998) to further increase the credibility of a simulation game. This means that the framework should allow players to come up with their own strategies and explore them as they would in the real-world, without however disrupting a simulation game to the extent that it becomes unstable.
- **Usability:** To leverage a game facilitator in managing a simulation game, the framework should include scenarios that are able to make decisions based on predefined measures. Furthermore, the framework should enable visualization and management of the interdependencies of a scenario, to support scenario development.
- **Usage:** The framework should be adaptive in the sense that it should support the achievement of a goal during game play. When a certain desired state does not occur, the framework should be able to adapt the state of the simulation game, for example by adding events to the event list or by changing attribute values of other objects. The framework can measure these effects since it is able to analyze the output of a simulation model (see Figure 1). The framework should be suitable for different types of simulation games. It should support games focusing on application domains other than that of simulation games for supply chain management as done in this research.

5 A FRAMEWORK TO SUPPORT SCENARIOS FOR SIMULATION GAMES

The framework we have developed is based on the Java based simulation suite called DSOL (Jacobs 2005), which is based on the event scheduling paradigm and uses a publish-subscribe mechanism to broadcast state changes of objects. We present an overview of the framework in Figure 3. We start with describing the abstract classes and interfaces of the framework, which form the basis. We continue with describing how these classes, and classes extending the base classes, address the requirements for the effectiveness of a scenario framework as described above.

- `AbstractScenario`: This class is a placeholder for all the objects and methods related to a scenario. It contains a reference to the conditions part of a scenario and a reference to the simulator that is used while playing a simulation game. Classes extending the `AbstractScenario` must imple-

ment code for parsing events into classes extending the `AbstractCondition`, i.e. a scenario class must implement the `parseConditions()` method.

- `AbstractCondition`: This class requires that conditions extending it execute a rule when a certain condition is met. Examples of conditions are the `PublishSubscribeCondition` and the `RepeatedCondition`. When the return value of the rule is 'true', a set of actions is executed. The default implementation of the `AbstractCondtion` has a `DefaultRule` which always returns true when checked.
- `RuleInterface` and `CompareRule`: The interface requires every rule to implement the `check()` method, which is invoked when a certain condition is met. An example of a rule is the `CompareRule`, which provides some basic functionality for comparing two objects with each other based on six different compare types. If a rule is satisfied, a set of actions is executed.
- `ActionInterface` and `AbstractAction`: An action is executed when a rule is checked and the return value is 'true'. Each action has a method called `execute()`, which is required according to the `ActionInterface`. An example of an action is one that upon execution adjusts the value of an attribute of type `Number`, for example a `Double`.

In what way does this framework support simulation games? First of all, a scenario contains a list of predefined events, which are executed once at the moment the clock of the simulation game reaches a certain value. An example is given in Section 2, where Example 2 contains an event that should be executed after 2 months of game play. These conditions are of a type called `TimedOnceCondition`. The execution of the predefined event is not different from the execution of a normal simulation event in DSOL, where the method of a certain object is invoked at the simulation time of the event. The list of predefined events supports a certain development of the game, that leads players from the present to the future state of the system (Gausemeier, Fink, and Schlake 1996) and it helps in showing the purpose (Jarke, Bui, and Carroll 1998) of the scenario. When carefully designed, such a list of events adds to the credibility of a simulation game.

Secondly, players should be able to explore a variety of strategies. Using a set of conditions, in combination with rules and actions, to analyze the output of a simulation game, the input of a simulation game may be adjusted to keep certain values of state variables within a desired bandwidth. This supports the overall stability of a simulation game. One of the functions of indicating a bandwidth is to prevent a player from being able to disrupt a whole simulation game.

We use the following piece of code to illustrate how the management of a simulation game is leveraged by helping players who got bankrupt (see Example 1 in Section 2).

```
//construct the action
AdjustNumberFieldAction action1 =
   new AdjustNumberFieldAction(
      manufacturer.getBankAccount(),
      "balance", ADJUSTMENT_TYPE.INCREASE,
      50000);

//construct the condition
new PublishSubscribeCondition(
   new ActionInterface[] { action1 },
      BankAccount.BANK_ACCOUNT_CHANGED_EVENT,
      manufacturer.getBankAccount(),
      new FieldConstantRule(manufacturer.
      getBankAccount(), "balance",
      CompareType.LT, new Double(-10000)));
```

It shows that when the value of a `BankAccount` of a manufacturer is less than -10000, its value is increased by 50000. One might view this as an interest free loan. To accomplish this, a `PublishSubscribeCondition` is used that subscribes itself to `BANK_ACCOUNT_CHANGED` events. Using this kind of conditions, a game facilitator is supported, since a lot of the interventions a game facilitator normally does, are done automatically by the framework. Another way to support a game facilitator in presumably better managing a simulation game is to notify a game facilitator when an intervention is needed, for example by a `NotifyAction`. A game facilitator can then decide what to do. With automated publish / subscribe actions that continuously monitor the value of certain state variables, the game facilitator does not need continiously to check the state of a simulation game. Supporting a game facilitator by decreasing its interaction with a simulation game increases its validity. The validity of a simulation game increases when less, often unpredictable and difficult to repeat, game facilitator initiated interactions take place with it. However, the added value of this needs to compared to the amount of interaction that takes place between players and the simulation model.

Support by the framework to achieve a goal of a simulation game is illustrated by the following piece of code:

```
Product[] products = Products.
resolveProduct("Laptop");

// construct the condition for period
// of 15-100 days, market share should
// be below threshold
new RepeatedCondition(new ActionInterface[]
{
```

```
   new AdjustCapacityAction(markets,
     products, true, 10.0),
   new AdjustPriceAction(manufacturers,
     products, true, 20.0) },
/* the rule */
new RegionMarketShareRule(this.simulator,
 15, 100, 30, true,
 RegionInterface.Util.getPredefinedRegions(),
 RegionInterface.ASIAN_REGION, products,
 StatisticsStoreInterface.actorType.
 MANUFACTURER), 5, TimeUnitInterface.DAY,
 this.simulator);
```

The above piece of code illustrates how we are able to control the development of market shares in a simulation game for supply chain management. The RegionMarketShareRule was specified specifically for the supply chain management games we are building. It states that there is a `RepeatedCondition` with two `Actions` and one `Rule`. It controls the market share of manufacturers in the Asian region for Laptops. In the period of 15 - 100 days, this market share should not be higher than 30%. The value for the market share is checked against this threshold every 5 days. If the market share exceeds this threshold, the two actions are executed, resulting in an increasing demand from the markets in the Asian region for Laptops, and an increase in the price that manufacturers ask for their Laptops.

To increase the usability of the scenario framework, we use it in the *coding* of the model, and as input for the simulation game in XML format, which will be illustrated in the next section. The requirement with respect to visualizing the various interdependencies that are part of a scenario is not yet supported by this framework. Further research is required here.

6 PROOF OF CONCEPT

The framework has been tested in the Distributor Game, a game developed for supply chain management issues centered around globalization (Houten et al. 2005). To support the teaching of globalization issues, the world has been divided into three regions: the U.S., Europe, and Asia, each with a number of suppliers, manufacturers, player-controlled distributors and markets. The products are consumer electronics such as Desktops, Laptops, etc. In each region, local and global markets are present. The global markets support global demand. The scenario we have played a number of times in real classes, focuses on a demand surge for Laptops and a shrinking demand for Desktops in the Asian region. The manufacturers in the Asian region are modeled in such a way that they can not meet the increased demand of the distributors. Hence, a shift in demand from Asian distributors to manufacturers in the European region is expected. If a player team recognizes this shift on time, for example via news messages, or by observing the right statistics, the team can anticipate and gain a competitive advantage over its competitors.

The initial set of events we use consist of a number of events as shown by the piece of XML code below. This event contains the information that at a certain time, the demand by the ASIAN market for the product Desktop is decreased by 25 percent, analogous to Example 2 in Section 2. Similar events were specified for other markets in the Asian region.

```
 <startTime>2005-06-15T00:00:00</startTime>
<marketEvent>
<market>ASIAN MKT</market>
<changeDemand>
<product>Desktop</product>
<increase>false</increase>
<percentage>25</percentage>
</changeDemand>
</marketEvent>
```

In combination with these events, we used news messages, see Example 3 in Section 2, that are also parsed into `TimedOnceConditions`. A news message is sent to players two weeks before the actual changes take place. This provides players with the opportunity to adapt their strategies and reconsider their relations with other supply chain actors. Similar events are used to adjust the demand for other consumer electronics products.

```
 <startTime>2005-05-30T00:00:00</startTime>
<NewsMessageEvent>
<subject>Desktop sales fall
</subject>
<content>With the rising mobility of
computer users...  but the economy is
slowly recovering.</content>
</NewsMessageEvent>
```

To make sure that the Asian distributors shift their demand to European manufacturers, a number of conditions are modeled that monitor the demand pattern of the Asian distributors. These conditions take into account when the demand pattern are supposed to shift. At the given time, and at intervals afterwards, rules are checked to see whether the shift took place, see Example 4 in Section 2. If not, actions are executed that increase the price that Asian manufacturers ask for their Laptops, and that increase the demand of the Asian market. This continues until a specified threshold is reached. We show the actual development of the market share for Laptops of manufacturers in the Asian region in Figure 2. The `MarketShareRule` as illustrated in the previous section became active after 15 days, and managed the market share until a threshold of 30% was reached. The point in time where the influence of the

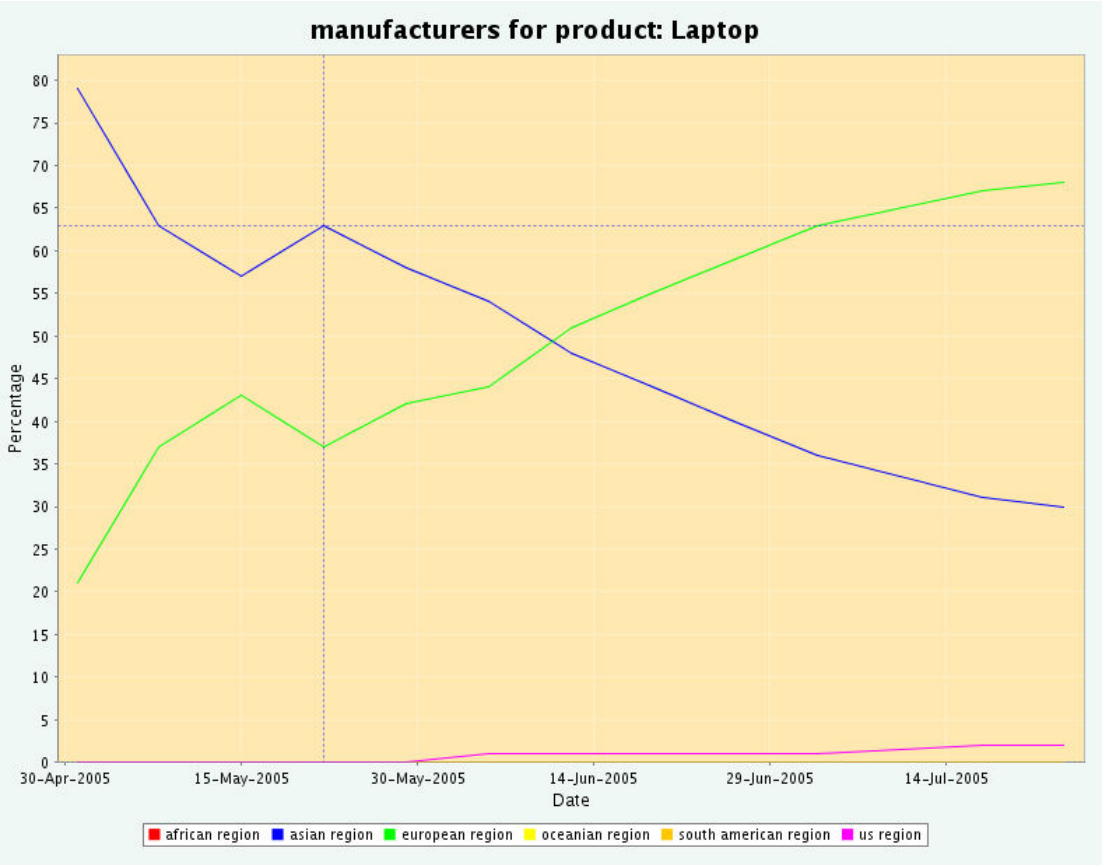

Figure 2: Development of the Market Share for Manufacturers for Laptops

`MarketShareRule` became noticeable is indicated by the dotted lines.

To leverage a game facilitator, conditions are used to monitor the financial status of supply chain actors, not only the players, in the simulation game. If necessary, loans are granted, and the actions are logged for debriefing.

From a modeling perspective, using XML files to represent the scenario worked well. However, coding the conditions, rules and actions, especially those that monitor the demand patterns proved to be a challenging task. Furthermore, it is not trivial to manage the interdependencies between the events. From a game facilitator perspective, the framework leverages the management tasks considerably. The framework monitors the execution of the scenario, and adjusts the financial state of the actors when necessary. When comparing playing games without scenarios and with scenarios, facilitation is much easier when using the scenarios that are based on our framework.

7 CONCLUSIONS

Simulation games are getting more complex and dynamic to better mimick today's systems, such as global supply chains. Hence, the development, usage and management of these games is becoming more challenging and difficult. In this paper we present a framework that supports the development, usage and management of simulation games through rule-based scenarios. The framework is based on the concept of production systems drawn from the domain of artifical intelligence, and consists of a number of conditions, (goal-based) rules and actions.

The framework eases the design of a scenario, supports a game facilitator during game play by decreasing the number of needed interventions, it supports achievement of the learning goal(s) of a simulation game, and it enhances the credibility of a simulation game. The latter is supported by providing players with the freedom to explore a variety of strategies without jeopardizing the scenario, its learning goal(s) and the stability of a simulation game.

Several topics can be addressed in future research. The first topic concerns the visualization and the management of interdependencies between the several parts of a scenario that are supported by the framework. For example the interdependencies between the predefined list of events for the initial state of a simulation game and the interventons during game play. A second topic is the development of a generic set of conditions, rules and actions suitable for supply chain management games, since the framework is

currently used in these type of games. A third topic is to research to what extent agent-based frameworks, such as the Belief-Desire-Intention framework (Rao and Georgeff 1995), can contribute to developing and using scenarios, and hence our framework. Agent-based frameworks often have functionalities for goal-based behavior, are based on rules, and have the ability to be used to measure the effectiveness over time of decisions made.

REFERENCES

Chiesl, N. E. 1990. Interactive real time simulation. In *Guide to business gaming and experiential learning*, ed. J. W. Gentry, 141–158. East Brunswick: Nichols/GP Publishing.

Dasgupta, S. 2003. Guest editorial: Internet-mediated simulation and gaming. *Simulation & Gaming: An Interdisciplinary Journal* 34(1):8–9.

Davis, P. K., and R. H. Anderson. 2003. Improving the composability of department of defense models and simulations. Report MG-101. Santa Monica: The RAND Corporation.

Engelmore, R., and T. Morgan (Eds.). 1988. *Blackboard systems*. Reading: Addison-Wesley.

Gausemeier, J., A. Fink, and O. Schlake. 1996. Scenario-management: planning and leading with scenarios. *Futures Research Quarterly* 12(3).

Gold, S. C., and T. F. Pray. 2001. Historical review of algorithm development for computerized business simulations. *Simulation & Gaming: An Interdisciplinary Journal* 32(1):66–84.

Hall, J. 1994. Computerized tutor support systems: the tutor's role, needs and tasks. In *The Simulation and Gaming Yearbook, Volume 2*, ed. R. Armstrong, F. Percival, and D. Smulders. Chapter 26. Kogan Page Limited.

Jacobs, P. H. M. 2005. The DSOL simulation suite. Enabling multi-formalism modeling in a distributed context. PhD thesis, Delft University of Technology, Delft.

Jarke, M., X. T. Bui, and J. M. Carroll. 1998. Scenario management - an interdisciplinary approach. *Requirements Engineering Journal* 3(3-4):155–173.

Keen, P. W. G., and H. G. and Sol. 2006. *Rehearsing the future.* To appear.

Lainema, T., and P. Makkonen. 2003. Applying constructivist approach to eduational business games: Case REALGAME. *Simulation & Gaming: An Interdisciplinary Journal* 34(1):131–149.

Magerko, B., and J. Laird. 2002. Towards building an interactive, scenario-based training simulator. In *Proceedings of the Behavior and Representation and Computer Generated Forces Conference*.

Nardi, B. A. 1992. The use of scenarios in design. *Special Interest Group on Human Computer Interaction (SIGCHI) Bulletin* 24(4):13–14. ACM Press.

Newell, A. and H. A. Simon. 1972. *Human problem solving*. Englewood Cliffs: Prentice-Hall.

Rao, A. S., and M. P. Georgeff. 1995. BDI agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*, 312–319.

Rich, E. 1983. *Artificial intelligence*. Singapore: McGraw-Hill.

Schank, R. C., A. Fano, B. L. Bell, and M. Y. Jonea. 1993. The design of goal based scenarios. *The Journal of the Learning Sciences* 3(4):305–345.

van Houten, S. P. A., A. Verbraeck, S. Boyson, and T. Corsi. 2005. Training for today's supply chains: an introduction to the distributor game. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 2338–2345. Available online via <http://www.informs-cs.org/wsc05papers/293.pdf> [accessed January 05, 2006].

Thorsteinsson, U. 1998. Computer games versus experiments. In *Games in operations management*, ed. J. Riis, R. Smeds, and R. van Landeghem, 35–46. Dordrecht: IFIP / Kluwer Academic Publishers.

Winston, P. H. 1984. *Artificial intelligence*. 2nd ed. Reading: Addison-Wesley.

Zeigler, B., H. Praehofer, and T. Kim. 2002. *Theory of modeling and simulation*. 2nd ed. San Diego: Academic Press.

AUTHOR BIOGRAPHIES

STIJN-PIETER VAN HOUTEN is a Ph.D. student at Delft University of Technology. His research is focused on services for decision support environments, specializing in interactive distributed simulation. His e-mail address is <s.p.a.vanhouten@tbm.tudelft.nl> and his web page is <www.tbm.tudelft.nl/webstaf/stijnh>.

ALEXANDER VERBRAECK is an associate professor in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology, and a part-time full professor in supply chain management at the R.H. Smith School of Business of the University of Maryland. He is a specialist in discrete event simulation for real-time control of complex transportation systems and for modeling business systems. His current research focus is on development of generic libraries of object oriented simulation building blocks in C++ and Java. His e-mail address is <a.verbraeck@tbm.tudelft.nl>, and his web page is <www.tbm.tudelft.nl/webstaf/alexandv>.

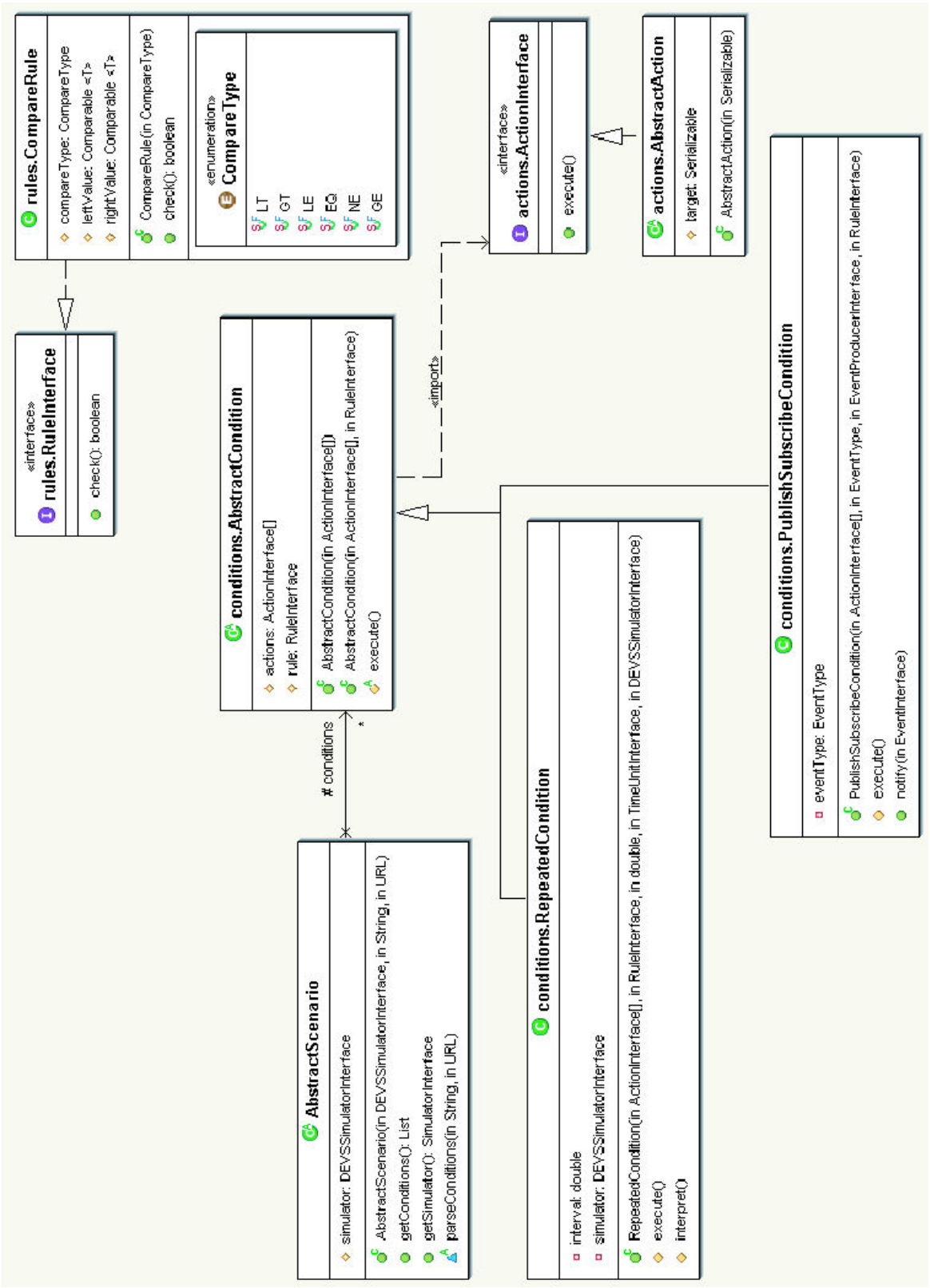

Figure 3: An Overview of the Framework for Scenarios