Proceedings of the 2011 Winter Simulation Conference
S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

PRODUCT DESIGN PATTERNS FOR AGENT-BASED MODELING

Michael J. North
Charles M. Macal

Argonne National Laboratory
9700 S. Cass Avenue
Argonne, IL 60439, USA

ABSTRACT

Since they were first introduced by architect Christopher Alexander in his classic book *The Timeless Way of Building*, design patterns have offered a powerful yet simple way to conceptualize and communicate designs in many disciplines. Design patterns became widely used for software development by the 1990's. These software design patterns have subsequently been shown to be of great value in improving the reliability and reducing the cost of software. Given that virtually all large-scale agent-based models are ultimately implemented in software, there is great potential for software design patterns to improve the practice of agent-based modeling. Several authors have discussed the use of patterns for agent-based modeling or agent-oriented software. This paper's contribution is that it provides an extensive set of both existing and new agent-based modeling design patterns, each of which is substantiated with at least three successful published example uses in models or modeling platforms.

1 INTRODUCTION

Since they were first introduced by architect Christopher Alexander in his classic book *The Timeless Way of Building*, design patterns have offered a powerful yet simple way to conceptualize and communicate designs in many disciplines (Alexander 1979). Design patterns form a shared lexicon for describing tried and true solutions for commonly faced software design problems. According to Alexander, "each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution." The first part of a design pattern characterizes the situation in which the problem occurs. The second part defines the problem to be solved. The third part describes a resolution to the outstanding issue as well as its positive and negative consequences. Every pattern has both fixed and variable elements. The fixed elements define the pattern. The variable elements allow the pattern to be adapted for each situation. Sets of adapted patterns are then used as a vocabulary to describe solutions to problems. Patterns have been widely studied including those for software (Gamma et al. 1995). This paper follows the current conventions of the software development community by presenting the parts of design patterns in a different order from that recommended by Alexander (1979). However, the essential content of each pattern follows Alexander's thinking. This paper's contribution is that it provides an extensive set of both existing and new agent-based modeling design patterns, each of which is substantiated with at least three successful published example uses in models or modeling platforms. This paper attempts to provide a complete set of such patterns. However, due to the rapidly evolving nature of agent-based modeling, no such list can yet be considered final.

It is important to note that design patterns are not intended to be new ideas or hypotheses. Rather, software design patterns are time-tested solutions to common computer programming problems that have been tried and proven to work on many different occasions. In a sense, patterns are not invented but are instead recognized after repeated use. This is well articulated by Schmidt, Johnson, and Fayad (1996) when they say that "good patterns arise from practical experience" and that "success is more important than novelty." An example is the Model-View-Controller (MVC) design pattern which is used as the conceptual foundation of nearly all modern user interface frameworks. MVC is discussed in more detail later in this paper.

Alexander's pioneering work focused on architecture, but it was later applied by many others to software design. In particular, the concept of software patterns, and more generally software pattern languages was pioneered by Beck and Cunningham (1987). Software patterns were later popularized by writers such as the software "Gang of Four" composed of Gamma, Helm, Johnson, and Vlissides (1995). These activities led to design patterns becoming widely used for software development. Patterns have subsequently been shown to be of great value in improving the reliability of software and reducing the cost of its development (Prechelt and Unger 1998; Hahsler 2003).

Today there are many widely accepted software design patterns. An example is the Memento pattern that can be used to store the state of programs to implement an undo command. Standard design patterns and associated categorization schemes can be found in Gamma et al. (1995) among others. Many of these patterns, as well as others, have a potential utility for specific agent-based models. However, this paper discusses patterns with proven utility in a range of agent-based models and agent modeling platforms.

In this paper, we consider one general kind of software design pattern namely, product patterns. Product patterns comprise the typical design patterns commonly discussed in the software patterns literature. The alternative is process patterns, which describe solutions to model development problems. Product patterns are used for crafting software (Gamma et al. 1995). Process patterns are found primarily in the business process community (Atwood 2006).

This paper discusses the use of design patterns to develop agent-based models. For an introduction to agent-based modeling itself, see Macal and North (2009), Macal and North (2011), and North and Macal (2007).

2 RELATED WORK

This section covers a range of published work related to design patterns and agent modeling. Grimm et al. (2005) suggest pattern-oriented modeling as a method for developing agent-based models. In Grimm et al. (2005)'s exposition, the term "pattern" refers to repeatedly observed behaviors in the referent under study. The patterns discussed by Grimm et al. (2005) are not directly related to the design patterns discussed in this paper. Three other kinds of related work are identified and discussed. First, alternative but generally complementary methods for describing agent-based models are considered. Second, papers that present agent-oriented software design patterns are discussed. Third, papers that directly apply design patterns to agent-based modeling are reviewed.

2.1 Proposed Formats for Describing Agent-based Models

Several formats have been proposed for describing agent-based models. Chief among these standards is Grimm et al.'s (2006) Overview, Design Concepts, and Details (ODD) protocol. ODD proposes to describe models using a three-part approach: overview, concepts, and details. The model overview includes a statement of the model's intent; a description of the main variables; and a discussion of the agent activities and timing (Grimm et al. 2006). The design concepts include a discussion of the foundations of the model. The details include the initial setup configuration, input value definitions, and descriptions of any embedded models (Grimm et al. 2006). The ODD protocol is highly complementary to the design patterns approach presented in the paper since design patterns can be used within the ODD descriptive text.

A second protocol for describing models, "Model Representing Potential Objects That Appear in The Ontology of Human-Environmental Actions & Decisions" (MR POTATOHEAD) (Parker et al. 2006) is a second protocol for describing agent-based models. MR POTATOHEAD shares so much in common with ODD that MR POTATOHEAD seems to be late in the process of being subsumed by ODD. In addition to providing a standard for describing agent-based models, MR POTATOHEAD is described by its creators as a "conceptual design pattern." Parker et al. (2006) describe a high level template that defines the essential elements of a model description rather than a software design pattern as defined in this paper. Similarly to ODD, design patterns are complementary to MR POTATOHEAD in that they can be used within model descriptions.

2.2 Agent-Oriented Design Patterns

Agent-oriented design patterns are related to, but substantially different than, agent-based modeling design patterns. A wide range of agent-oriented design patterns have been discussed in the literature including those considered by Aridor and Lange (1998), de Araújo Lima et al. (2003), and Kendall et al. (1998). Aridor and Lange (1998) consider the Itinerary, Forwarding, Ticket, Master-Slave, Plan, Meeting, Locker, Messenger, Facilitator, and Organized Group patterns. de Araújo Lima et al. (2003) discuss the Itinerary, Star-Shaped, Branching, Master-Slave, MoProxy, Meeting, Facilitator, and Mutual Itinerary Recording patterns. Kendall et al. (1998) cover the Layered Agent, Reactive Agent, Deliberative Agent, Interface Agent, Intention, Prioritizer, Adaptable Active Object, Conversation, Facilitator, Proxy, Protocol, Emergent Society, Clone, Remote Configurator, Broker, Linker Layer, and Agent Builder patterns. In addition, Oluyomi, Karunasekera, and Sterling (2004) and Oluyomi (2006) separately detail two different categorization schemes for agent-oriented design patterns. Agent-oriented design patterns, while sometimes used to develop simulations, generally are intended to support the development of mobile and proxy agent systems. As such, they attempt to solve problems that are substantially different than those typically found in agent-based simulations that seek to faithfully reproduce real world behavior. This causes agent-oriented design patterns to be of limited utility for agent-based modeling.

2.3 Agent-based Modeling Design Patterns

Agent-based modeling design patterns have been proposed by many authors. These recommendations are reviewed in this section.

Nguyen, Ricken, and Wong (2004) discuss the use of several standard patterns as part of an agent-based marine biology simulation. It is important to note that this simulation is not an applied model but rather a classroom example intended for teaching purposes. The standard design patterns that are discussed are the Model-View-Controller (MVC), Visitor, Factory, Command, State, and Composite patterns. All of these patterns are of utility for agent-based modeling except for Command and Visitor. Command can be used for agent-based modeling, but is more commonly applied to user interfaces for tasks such as translating mouse clicks into calls to routines that provide appropriate responses. Visitor tends not to be used when the components of the dominant objects (i.e., agents) are self-directed. We discuss the rest of Nguyen, Ricken, and Wong's patterns later in the section on applied patterns.

Moser, Merdan, and Biffl (2010) use two standard software design patterns, namely Master-Slave and Blackboard, to develop a factory automation simulator. Master-Slave is primarily a concurrent computing pattern intended to take advantage of multiple processors. Following C. A. R. Hoare's admonishment, "premature optimization is the root of all evil in programming," it is recommended that models be correctly implemented and then optimized. Therefore, the Master-Slave pattern is of limited utility for designing an agent-based modeling, though it might be used later during model optimization. Blackboard is useful for artificial intelligence applications that need to distribute responsibility for solving a problem among several independent agents. Blackboard prescribes specific data structures for sharing information that makes Blackboard less useful for agent-based models. Therefore it will not be considered further in this paper.

Boronea et al. (2009) detail a wide range of standard design patterns in three categories, namely, traveling patterns, task patterns, and interaction patterns. The traveling patterns are Itinerary, Star-shaped, Branching, Forwarding, and Ticket. All of the traveling patterns focus on moving agents around specific network topologies, usually completing tasks at each node they visit. These patterns seem to be of limited utility for general agent-based modeling, since this kind of regular migratory work on a network is unusual in applied models. The task patterns are Master-Slave and Plan. Master-Slave was previously addressed in the discussion of Moser, Merdan, and Biffl (2010) recommendations. Plan is related to the Master-Slave pattern and is intended to synchronize a series potentially concurrent agent activities. The interaction patterns are Meeting, Messenger, Organized Group, and Facilitator. Meeting is generally used to address application security constraints, which are not relevant for general agent-based modeling. Messenger either builds on Meeting or optimizes high volume message traffic. Similarly, Organized Group is designed to support the simultaneous migration of multiple agents between processors for performance reasons. Messenger and Organized Group may be useful for optimizing certain distributed simulations. Following Hoare's guidance, these optimizations should only be used after a model is implemented. The Facilitator pattern is a specialization of the Master-Slave design pattern. Like the Master-Slave pattern, the Facilitator pattern is not commonly useful for agent-based model and it will not be considered further in this paper.

Railsback and Lytinen (2006) briefly discuss the Template Method design pattern. We discuss this pattern later as a special case of the more general Strategy pattern.

Rixon, Robinson, and Boschetti (2006) provide a detailed exposition of what they describe as the Meme pattern. This new pattern seems useful for certain classes of social simulation. However, we view this as a domain-specific model rather than a general design for agent-based modeling. While popular, the concept of memes is also quite contentious in social science, as it is considered by many to be excessively reductionist and rigid (Fracchia and Lewontin 1999, 2005).

Klügl and Karlsson (2009) briefly mention a range of patterns but only define two on a formal basis, namely Perception Memory and Exponential Growth. Perception Memory is designed to address situations in which agents need to form an internal model of their environment. Agents are assigned a scanning distance and local memory to store the results of scanning their environment. This pattern seems to have some limited uses but does not seem to be widely used in agent-based modeling. Exponential Growth is a simple pattern that describes how agent populations can be made to grow in an exponential way. Klügl and Karlsson (2009) argue against the apparent triviality of this pattern, although we do not consider their argument to be persuasive.

3 DESIGN PATTERN USAGE EXAMPLES

As previously mentioned, design patterns are not intended to be new theories. Rather they are meant to be proven practices documented by a range of successful examples. This section briefly presents two kinds of examples, namely toolkits and models. These examples will be used later when discussing specific design patterns.

The first set of examples consist of agent-based modeling toolkits. These examples demonstrate methodological design patterns and also design patterns of such great utility that they are built directly into widely used tools.

The second set of examples consist of influential agent-based models. These examples demonstrate the use of design patterns to solve specific modeling problems.

3.1 Example Toolkits

Selected agent-based modeling toolkits will be used to document the use of the design patterns discussed in this paper. The primary selection criteria are that the toolkit be freely available for at least non-commercial use and that there is documentation in the open literature.

NetLogo is an educational system that leverages the Logo language (Feurzeig et al. 1970) to simplify learning and use. NetLogo (Wilensky 1999) is a Logo-based toolkit that uses a Java interpreter and interface. NetLogo includes a large models library containing a range of examples. NetLogo is freely available for non-commercial use (Wilensky 1999).

Repast Simphony (North et al. 2007) is a tightly integrated, richly interactive, cross platform Java-based modeling system that runs under Microsoft Windows, Apple Mac OS X, and Linux. It supports the development of extremely flexible models of interacting agents for use on workstations and small computing clusters. Repast Simphony models can be developed in several different forms including the ReLogo dialect of Logo, point-and-click flowcharts translated into Repast Simphony models, Groovy, or Java, all of which can be fluidly interleaved. NetLogo models can also be imported. Repast Simphony has been successfully used in many application domains. For example, first Repast (North et al. 2006) and then later Repast Simphony were used to implement the Virtual Market Learning Lab model (North et al. 2010) discussed later. Repast Simphony is free for all uses and is open source (Repast Organization for Architecture and Development 2011).

Repast for High Performance Computing (Repast HPC) (Collier and North 2011) is a next generation agent-based modeling system intended for large-scale distributed computing platforms. It implements the core Repast Simphony concepts (e.g., contexts and projections), modifying them to work in a parallel distributed environment. Repast HPC is written in cross-platform C++. It can be used on workstations, clusters, and supercomputers running Microsoft Windows, Apple Mac OS X, Linux, or Unix. Portable models can be written in either standard or Logo-style C++. Repast HPC has been successfully tested for scalability on Argonne National Laboratory's Blue Gene/P high performance computer. Repast HPC is free for all uses and is open source (Repast Organization for Architecture and Development 2011).

As a note, North is the leader developer and a principal investigator for the Repast Suite of tools including Repast, Repast S, and Repast HPC. Macal is also a principal investigator for the Repast Suite.

StarLogo is an educational system that focuses on ease of use and has been widely used for educational purposes. Like NetLogo, StarLogo (StarLogo 2011) is a Logo-based toolkit that uses a Java interpreter and interface. StarLogo is freely available and open source for non-commercial use (StarLogo 2011).

Swarm (Johnson 2000) is an Objective-C-based system intended for applied modeling. Swarm also includes a Java wrapper that allows the Objective-C based features to be invoked in Java programs. Swarm was used to implement the InSTREAM model (Railsback et al. 2009) discussed later. Swarm is free for all uses and is open source (Swarm Development Group 2011).

3.2 Example Models

Several example models, which cover a wide range of application areas, document the use of the design patterns discussed in this paper. These models were selected based on their size, complexity, impact, and the availability of published information.

The Virtual Market Learning Lab (North et al. 2010) is a large-scale, agent-based model of consumer markets co-developed by Argonne National Laboratory and Procter & Gamble (P&G). It represents the shopping behavior of consumer households and the business behavior of retailers and manufacturers in a simulated national consumer market. All of the major participants associated with one category of products (e.g., laundry detergent) are simulated during each model run. Argonne and P&G successfully calibrated, verified, and validated the resulting agent-based model using several independent real-world data sets for multiple consumer product categories with over sixty comparison tests per data set. First Repast (North, Collier, and Vos 2006) and then later Repast Simphony (North et al. 2007) were used to implement the model. The model has been successfully applied by P&G to several challenging business problems where it has directly influenced managerial decision-making and produced substantial cost savings.

The Hydrogen Economy Model (Mahalik et al. 2007) is a Repast Simphony representation of the Los Angeles, California metropolitan area, with 5,000 square miles of detailed GIS-sourced interstate highways and omnipresent local roads. The model has driver and investor agents. Driver agents use their cars to move between their demographically-assigned home neighborhoods and their jobs. Drivers have a variety of characteristics including income levels, environmental concerns, risk aversion, and car type preferences (e.g., wanting conventional fuel cars versus hydrogen cars). Investor agents build, own and operate hydrogen fuel stations, based on the investor’s estimates of the potential for profits at each available geo-located site.

The InSTREAM model (Railsback et al. 2009), an agent-based simulation of salmon behavior, is an interesting example of an ecological simulation. InSTREAM is used to study how river salmon populations react to changes in both river ecology and management policies. InSTREAM uses a time step of one day to simulate individual fish behaviors including foraging location choice, eating, growth, reproduction, and death. Railsback et al. (2009) also present methods for using their model to address real-world fishery problems.

The Electricity Market Complex Adaptive Systems Model (EMCAS) (Conzelmann et al. 2004) is an extensive agent-based model of electric power markets with a focus on deregulated systems. Multiple and diverse market participants are each represented with their own unique set of objectives; risk preferences; business and bidding strategies; and decision rules. The success of an agent is a function not only of its own decisions and actions, but also of the decisions and actions of other market participants. EMCAS is now a commercial product used for and by many organizations around the world. EMCAS is implemented using Repast (North, Collier, and Vos 2006), the precursor to Repast Simphony.

4 PRODUCT DESIGN PATTERNS

Proven agent-based modeling product patterns are discussed in this section. Product patterns offer design solutions, as opposed to process patterns that describe ways to implement solutions. The exposition format for the design patterns in this section follows widely used conventions (Gamma 1995) that present the pattern's name as the title of the subsection, the fundamental problem being addressed, the context in which the problem occurs, the conflicting forces that make the problem difficult to solve, the resulting context after using the pattern, the rationale for the pattern, and example uses of the pattern. The text for each pattern is intentionally concise to highlight the essence of each pattern.

4.1 Scheduler Scramble

Problem: Two or more agents from the ABM pattern can schedule events that occur during the same clock tick. Getting to execute first may be an advantage or disadvantage. How do you allow multiple agents to act during the same clock tick without giving a long-term advantage to any one agent?

Context: Two or more agents from the ABM pattern may attempt to simultaneously execute behaviors during the same clock tick.

Forces: Two or more agents want to activate a behavior at the same time. Activating a behavior before other agents can be either an advantage or disadvantage for the agent that goes first. No agent should have a permanent advantage or disadvantage from the effect.

Solution: Schedule the competing behaviors in a single clock tick in a random order. Use a different random order for each clock tick in which there are competing behaviors.

Resulting Context: A sequential behavioral activation order is produced. Each of the behaviors can be executed sequentially.

Rationale: The ABM pattern should allow several agents to activate during the same clock tick. The Scheduler Scramble pattern is intended to allow this.

Example Uses: This agent-based design pattern is used in the Virtual Market Learning Laboratory (North et al. 2010), the Hydrogen Economy Model (Mahalik et al. 2007), InSTREAM (Railsback et al. 2009), EMCAS (Conzelmann et al. 2004), Repast HPC (Collier and North 2011), Repast Simphony (North et al. 2007), and Swarm (Johnson 2000).

4.2 Context and Projection Hierarchy

Problem: How do you organize complex spaces into a single unified form such that individual agents can simultaneously exist in multiple spaces and the spaces themselves can be seamlessly removed and added?

Context: Several different ways to model space (e.g., two dimensional spaces, three dimensional spaces, networks, and geographical information systems) need to be combined in complex connected ways.

Forces: Different views of space need to be stored and manipulated simultaneously.

Solution: Use the Composite pattern to create named, hierarchically nested containers for agents (i.e., "Contexts"). Represent spatial patterns in each Context using a set of Projections. There is one Project type for each space type (e.g., two dimensional spaces, three dimensional spaces, networks, and geographical information systems). Projections store the appropriate spatial information for each agent in the Project's associated Container. As many Projections of each type can be added to a space as needed. Every agent in a given context is in all of the Projections in that Context. Agents can be placed at any level in a Context hierarchy. An agent in a given Context of a Context hierarchy is also in all parents of the containing Context. Agents can be in as many Contexts as required. Agents can be placed in a Context with simple add and remove operations. Contexts be asked to provide lists of their contained agents and Projections.

Resulting Context: There is a hierarchically nested tree of Context, each potentially with several spatial Projections.

Rationale: There is a need to represent complex nested spaces and externalize storage away from agents.

Example Uses: This agent-based modeling design pattern is used in the Virtual Market Learning Laboratory (North et al. 2010), the Hydrogen Economy Model (Mahalik et al. 2007), Repast HPC (Collier and North 2011), and Repast Simphony (North et al. 2007).

4.3 Diffuser

Problem: How do you represent the flow of values (e.g., heat or pheromones) in scalar fields?

Context: There is a scalar field of values that needs to be updated over time.

Forces: Values in the field are deposited and then diffuse and evaporate.

Solution: Represent the scalar field using a uniform lattice of individual values. Represent deposition using agent rules to add to the values at the lattice points. Represent diffusion and evaporation using strategy design patterns that allow developers to specify how values at each point are affected by neighboring points and the passage of time. Allow the strategies to be activated over time as needed.

Resulting Context: There is a simple representation of flow in scalar fields.

Rationale: Flow in scalar fields is commonly needed in agent-based models.

Example Uses: This pattern is used in Repast HPC (Collier and North 2011), Repast Simphony (North et al. 2007), Swarm (Johnson 2000), NetLogo (Wilensky 1999), and StarLogo (StarLogo 2011).

4.4 Strategy

Problem: How do you let a clients invoke rules that may be defined long after the clients are implemented?

Context: There are a set of rules that need to be dynamically selected while a program is running. There is a need to separate rule creation from rule activation.

Forces: There are sets of rules which may be updated long after the client program is implemented.

Solution: Define an interface with a stub method that represents the fundamental behavior of interest. Implement one concrete version of the interface (i.e., a class) for each of the different rules of interest. Configure the dynamic behavior selection point to activate the method for a variable with the interface type. Rules can be dynamically selected by assigning an object of the appropriate concrete class to the variable.

Resulting Context: The assignment of a variable can dynamically select the set of rules to run.

Rationale: There is a need for dynamic rule selection long after the client program is implemented.

Example Uses: This is a standard design pattern (Gamma et al. 1995). This pattern is also used in the Virtual Market Learning Laboratory (North et al. 2010), the Hydrogen Economy Model (Mahalik et al. 2007), EMCAS (Conzelmann et al. 2004), Repast HPC (Collier and North 2011), Repast Simphony (North et al. 2007), and Swarm (Johnson 2000). Railsback and Lytinen (2006) also briefly discuss a special case of Strategy called the Template Method design pattern.

4.5 Logo World

Problem: What is a simple but highly flexible ontology for agent-based modeling?

Context: There is a need for a simple but generic modeling ontology.

Forces: The ontology needs to represent movable agents, the environment, and connections between agents.

Solution: Following NetLogo (Wilensky 1999), define four kinds of agents, namely Turtles, Patches, Links, and an Observer. Turtles are agents that move in a two dimensional Cartesian world. Turtles can sense neighboring Turtles and underlying Patches. Patches are immobile agents that form a simple rectangular tiling of the Cartesian world. Links are agents that connect Turtles. The Observer is a Singleton that provides global coordination.

Resulting Context: There is an simple ontology that can be used to model many complex systems.

Rationale: The fundamental components (i.e., Turtles, Patches, Links, and an Observer) provide a straightforward way to model many circumstances.

Example Uses: This agent-based modeling design pattern is used in the Repast HPC (Collier and North 2011), Repast Simphony (North et al. 2007), NetLogo (Wilensky 1999), and StarLogo (StarLogo 2011).

4.6 Learning

Problem: How can agents adapt or learn?

Context: There is a need for agents to change their behavior over time based on their experiences.

Forces: Agents have changing experiences which should influence their behaviors.

Solution: Create “rules and rules to change the rules” (Casti 2002). The base-level rules make the regular decisions. The higher-level “rules to change the rules” modify the base-level rules to allow the agent to act differently over time. The higher-level rules normally store measures of success to determine how the base-level rules should be changed. Any mechanism for decision making can be used at the higher levels including artificial intelligence techniques such as neural networks and genetic algorithms as well as statistical techniques such as regression. The Strategy design pattern is sometimes used to make it easier to dynamically change the base-level rules.

Resulting Context: Agents can change their behavior over time in response to their environment.

Rationale: Agents need to change their behavior in a systematic way.

Example Uses: This agent-based modeling design pattern is used in the Virtual Market Learning Laboratory (North et al. 2010), the Hydrogen Economy Model (Mahalik et al. 2007), InSTREAM (Railsback et al. 2009), EMCAS (Conzelmann et al. 2004), Repast HPC (Collier and North 2011), Repast Simphony (North et al. 2007), Swarm (Johnson 2000), NetLogo (Wilensky 1999), and StarLogo (StarLogo 2011).

4.7 Double Buffer

Problem: Different agents from the ABM pattern often simultaneously view and change shared values during the same clock tick. The results of the updates and changes should not be dependent on the order of event execution. How do you allow different agents to use and change the same item during the same clock tick without introducing dependencies on the behavioral activation order?

Context: Two or more agents from the ABM pattern may need to simultaneously use and change shared items during the same clock tick while avoiding behavioral execution order dependencies.

Forces: You want to allow the competing agents to use or view a constant value while also allowing them make changes to the value. The changes should be visible after the tick has completed.

Solution: Use two storage locations for each value to be accessed. The first location is always used for reading or viewing. The second location is used for temporary storage of updates during a clock tick. The temporary value is copied to the reading location at the end of each clock tick.

Resulting Context: Multiple storage locations are used for holding values. The copy operation requires a special behavior to occur at the conclusion of each clock tick.

Rationale: The ABM pattern should allow different agents to simultaneously view and change shared values during the same clock tick. The Double Buffer pattern is intended to allow this.

Example Uses: This agent-based modeling design pattern is used in the Virtual Market Learning Laboratory (North et al. 2010), EMCAS (Conzelmann et al. 2004), Repast HPC (Collier and North 2011), and Repast Simphony (North et al. 2007).

4.8 Model-View-Controller (MVC)

Problem: How to separate application-specific logic from user interface code?

Context: There is a need for a user interface with application-specific logic.

Forces: A general user interface framework needs to work with domain code.

Solution: Create a domain-specific model that is connected to a user interface view. The view is then managed by a controller.

Resulting Context: There is a user interface with a cleanly divided domain-specific model, user interface view, and management controller.

Rationale: There is a need to separate user interface code from domain specific code.

Example Uses: This is a common approach to user interface design that discovered prior to the introduction of design patterns into software development (Burbeck 1987, Reenskaug 1978-1979). Following the widespread use of patterns in software development, MVC became accepted as a standard design pattern. This pattern is used in the Virtual Market Learning Laboratory (North et al. 2010), the Hydrogen Economy Model (Mahalik et al. 2007), EMCAS (Conzelmann et al. 2004), Repast Simphony (North et al. 2007), Swarm (Johnson 2000). This pattern is also discussed by Nguyen, Ricken, and Wong (2004).

5 CONCLUSIONS

Modern software development methods approaches cleanly separate the design of software, such as agent-based models, from the details of implementation, and provide approaches and procedures to facilitate this separation. Design patterns are one approach to facilitating the software and model design process. Design patterns promote reusability and communication of solutions to important problems that regularly reoccur in software development and, especially, in modeling; they can be nested and combined, and used as building blocks to comprise software solutions of increasing complexity. The paper has identified design patterns that have proven themselves useful in developing specific agent-based models and agent-based modeling platforms. These design patterns represent a promising and useful tool in the agent-based modeler's toolbox.

ACKNOWLEDGMENTS

This work was supported by the US Department of Energy under contract number DE-AC02-06CH11357.

REFERENCES

Alexander, C. 1979. *The Timeless Way of Building*. Oxford: Oxford University Press.

Aridor, Y., and D. B. Lange. 1998. "Agent Design Patterns: Elements of Agent Application Design." *2nd International Conference on Autonomous Agents*. 108–115.

Atwood , D. 2006. "BPM Process Patterns: Repeatable Design for BPM Process Models." *BP Trends*. May.

Beck, K. and W. Cunningham. 1987. "Using Pattern Languages for Object-Oriented Programs (Tektronix Technical Report CR-87-43." Presented at *The 1987 Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA-87) Workshop on Specification and Design for Object-Oriented Programming*.

Boronea, S., F. Leon, M. H. Zaharia, and G. M. Atanasiu. 2009. "Design Patterns for Multi-Agent Simulations." *Management & Marketing*. 4 (4), 15–26.

Burbeck, S. 1987. "Applications Programming in Smalltalk-80: How to use Model-View-Controller (MVC)." notes stored in the University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive. Accessed October 25. http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

Casti, J. 2002. Personal communication. March.

Collier, N.T., and M.J. North. In Press 2011. "Repast HPC: A Platform for Large-scale Agent-based Modeling." in W. Dubitzky, K. Kurowski, and B. Schott, eds. *Large-Scale Computing Techniques for Complex System Simulations*. Wiley.

Conzelmann, G., Boyd, R. Cirillo, V. Koritarov, C.M. Macal, M.J. North, P.R. Thimmapuram, and T. Veselka. 2004. "Analyzing the Potential for Market Power using an Agent-based Modeling Approach: Results of a Detailed U.S. Power Market Simulation." In *Proc. of the International Conference on Computing, Communication and Control Technologies*. vol. VI, The University of Texas at Austin and the International Institute of Informatics and Systemics. Austin, TX USA. 109–114.

Feurzeig W., S. Papert, M. Bloom, R. Grant, and C. Solomon. 1970. "Programming-Languages as a Conceptual Framework for Teaching Mathematics." *Interface*, 4 (2).

Fracchia, J. and R. C. Lewontin. 2005. "The Price of Metaphor." *History and Theory*, 44 (1) (Feb.), 14–29.

Fracchia, J. and R. C. Lewontin. 1999. "Does Culture Evolve?" *History and Theory*. 38 (4) (Dec.), 52-78

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Wokingham, U.K.: Addison-Wesley.

Grimm, V., Berger, U., Bastiansen, F., Eliassen, S., Ginot, V., Giske, J., Goss-Custard, J., Grand, T., Heinz, S.K., Huse, G., Huth, A., Jepsen, J.U., Jørgensen, C., Mooij, W.M., Muller, B., Pe'er, G., Pou, C., Railsback, S.F., Robbins, A.M., Robbins, M.M., Rossmanith, E., Ruger, N., Strand, E., Souisi, S., Stillman, R.A., Vabø, R., Visser, U., DeAngelis, D.L. 2006. "A Standard Protocol for Describing Individual-based and Agent-based Models.' *Ecological Modelling*, 198 (1–2), 115–126.

Grimm, V., E. Revilla, U. Berger, F. Jeltsch, W. M. Mooi, S. F. Railsback, H.-H. Thulke, J. Weiner, T. Wiegand, and D. L. DeAngelis. 2005. "Pattern-Oriented Modeling of Agent-Based Complex Systems: Lessons from Ecology." *Science*. 310 (5750), 987–991.

Hahsler, M. 2003. *A Quantitative Study of the Application of Design Patterns in Java*. Working Papers on Information Processing and Information Management Nr. 01/2003, Institute of Information Processing and Information Management.

Johnson, P. Accessed 2000. *Swarm User Guide*. http://www.swarm.org/swarmdocs-2.1.1/userbook/userbook.html.

Kendall, E. A., P. V. Murali Krishna, C. V. Pathak, C. B. Suresh. 1998. "Patterns of Intelligent and Mobile Agents." In *Proceedings of the 2nd International Conference on Autonomous Agents*.

Klügl F., and L. Karlsson. 2009. "Towards Pattern-Oriented Design of Agent-Based Simulation Models." In *Proceeding of the 7th German Conference on Multi-Agent System Technologies*, edited by L. Braubach, W. van der Hoek, P. Petta, and A. Pokahr, 41-53. Lecture Notes in Computer Science.

de Araújo Lima, E., P. de Lima Machado, J. César Abrantes de Figueiredo, and F. R. Sampaio. 2003. "Implementing Mobile Agent Design Patterns in the JADE Framework." 3(3) *Search of Innovation*.

Macal, C.M. and M.J., North. 2010. "Tutorial on Agent-based Modeling and Simulation." *Journal of Simulation*, 4, 151-162. Palgrave Macmillan, New York, NY USA, No. 4.

Macal, C.M. and M.J. North. 2009. "Agent-based Modeling and Simulation." In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, 86–98. Piscataway, New Jersey: Institute of Electrical and Electronic Engineers, Inc.

Mahalik, M.R., G. Conzelmann, C.H. Stephan, M.M. Mintz, T.D. Veselka, G.S. Tolley, and D.W. Jones. "Modeling the Transition to Hydrogen-Based Transportation." In *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*. Argonne National Laboratory. Argonne, IL USA (November 2007).

Moser, T., M. Merdan, and S. Biffl. 2010. "A Pattern-Based Coordination and Test Framework for Multi-Agent Simulation of Production Automation Systems." *2010 International Conference on Complex, Intelligent and Software Intensive Systems*.

Nguyen, D., M. Ricken , and S. Wong. 2004. "Design Patterns for Marine Biology Simulation." *2004 Special Interest Group on Computer Science Education Symposium*.

North, M.J., N.T. Collier, and R.J. Vos. 2006. "Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit." *ACM Transactions on Modeling and Computer Simulation*, 16 (1), 1-25. ACM, New York, NY USA.

North, M.J., T.R. Howe, N.T. Collier, and J.R. Vos,. 2007. "A Declarative Model Assembly Infrastructure for Verification and Validation." in S. Takahashi, D.L. Sallach and J. Rouchier, eds. *Advancing Social Simulation: The First World Congress*. Springer, Heidelberg, FRG.

North, M.J., and C.M. Macal. 2007. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. Oxford, New York, NY USA.

North, M.J., C.M. Macal, J. St. Aubin, P. Thimmapuram, M. Bragen, J. Hahn, J. Karr, N. Brigham, M.E. Lacy, and D. Hampton. 2010. Multi-scale Agent-based Consumer Market Modeling. *Complexity*, 15 (5), 37–47.

Oluyomi, A. 2006. "Patterns and Protocols for Agent-Oriented Software Development." Doctoral Dissertation Department of Computer Science and Software Engineering. The University of Melbourne. Australia.

Oluyomi A., S. Karunasekera, and L. Sterling. 2004. "An Agent Design Pattern Classification Scheme: Capturing the Notions of Agency in Agent Design Patterns." In *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*.

Parker, D.C., D. G. Brown, J. G. Polhill, P. J. Deadman, and S. M. Manson. 2006. "Illustrating a New 'Conceptual Design Pattern'" for Agent-based Models of Land Use via Five Case Studies—the MR POTATOHEAD Framework." in *Agent-Based Modelling in Natural Resource Management*. A. López Paredes and Cesáreo Hernández Iglesias, eds.

Prechelt, L. and B. Unger. 1998. "A Series of Controlled Experiments on Design Patterns: Methodology and Results." *Proceedings of Softwaretechnik*.

Railsback, S. F. and S. L. Lytinen. 2006. "Agent-based Simulation Platforms: Review and Development Recommendations." *SIMULATION*. 82 (9), 609-623.

Railsback, S. F., B. C. Harvey, S. K. Jackson, and R. H. Lamberson. 2009. *InSTREAM: The Individual-Based Stream Trout Research And Environmental Assessment Model*. General Technical Report PSW-GTR-218. U. S. Department of Agriculture Forest Service, Pacific Southwest Research Station, Albany, California.

Reenskaug, T. 1978-1979. *The Original MVC Reports*. Dept. of Informatics University of Oslo.

Repast Organization for Architecture and Development. Accessed 2011. Repast Suite. http:// repast.sourceforge.net/.

Rixon, A. P. Robinson, and F. Boschetti. Submitted 2006. "The Meme as a Design Pattern for Social Learning in Agent-Based Models." *Journal of Artificial Societies and Social Simulation*.

Schmidt, D. C., R. E. Johnson, and M. Fayad. 1996. "Software Patterns." *Communications of the ACM*, 39 (10), 36–39.

StarLogo. Accessed 2011. StarLogo. http://education.mit.edu/starlogo/.

Swam. Accessed 2011. Swarm. http://www.swarm.org/.

Wilensky, U. Accessed 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

AUTHOR BIOGRAPHIES

MICHAEL J. NORTH is the Deputy Director of the Center for Complex Adaptive Agent Systems Simulation within the Decision and Information Sciences Division of Argonne National Laboratory; a Senior Fellow in the joint Computation Institute of the University of Chicago and Argonne; and an Adjunct Professor in the Institute of Design at the Illinois Institute of Technology. Dr. North has over twenty years of experience developing and applying advanced modeling and simulation applications for the U.S. federal government; U.S. state governments; international agencies; private industry; and academia. Dr. North is the lead developer and is a principal investigator for the widely used free and open source Repast agent-based modeling suite (http://repast.sourceforge.net). Dr. North is the lead author of the book *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation* (Oxford 2007) and has authored or co-authored over fifty journal articles and conference papers. Dr. North has been a keynote speaker at a variety of national and international events. Dr. North has been recognized as a Senior Member by the Association for Computing Machinery (ACM), the Institute of Electrical and Electronics Engineers (IEEE), and the American Society for Quality (ASQ). Dr. North holds ten college degrees, including a Ph.D. in Computer Science from the Illinois Institute of Technology. His email address is north@anl.gov.

CHARLES M. MACAL is a Senior Systems Engineer and Director of the Center for Complex Adaptive Agent Systems Simulation within the Decision and Information Sciences Division of Argonne National Laboratory; a Senior Fellow in the joint Computation Institute of the University of Chicago and Argonne; and an Adjunct Professor at the University of Chicago where he teaches a course on Complex Adaptive Systems for Threat Management and Emergency Preparedness. Dr. Macal is a principal investigator for the development of the widely used Repast agent-based modeling toolkit. Dr. Macal is a co-author of the book *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation* (Oxford 2007). Dr. Macal holds a Ph.D. in Industrial Engineering from Northwestern University and is a registered professional engineer in the State of Illinois. His email address is macal@anl.gov.