TOWARDS AN AGENT-ORIENTED FRAMEWORK FOR SERIOUS GAMES

Architecting with Behavioural Software Agents

Aaron D. Tull, Tucker S. Smith and Kendra M. L. Cooper
The University of Texas at Dallas, 800 West Campbell Road, Richardson, Texas, U.S.A.

Keywords: Serious Game Framework, Agent-oriented, Software Architecture, Pattern, Agent behaviour, Component.

Abstract: Agent-oriented software architectures for serious games and game development frameworks are beginning to receive attention. They are viewed as potentially valuable solutions to support the rapid and inexpensive development of games with high usability and playability. This problem domain also has challenging technical quality of service requirements (modifiability, performance, flexibility, scalability, concurrency, portability, integration of diverse technologies), which need to be carefully considered in the architecture. Here, we present a collection of high level requirements (goals) for this problem domain and argue that a new, agent-oriented component based solution is needed that explicitly models agent-based behaviour. We propose an agent-oriented extension of the adaptive object-model pattern in this research; a running example on requirements engineering software engineering education is used to help explain the architecture. The architecture is discussed with respect to the technical quality of service requirements.

1 INTRODUCTION

Software Engineering (SE) is a knowledge intensive, specialized, rapidly changing discipline; it's educational infrastructure faces significant challenges including the need to rapidly, widely, and cost effectively introduce new or revised course material; encourage the broad participation of students; address changing student motivations and attitudes; support undergraduate, graduate and lifelong learning; and incorporate the skills needed by industry. Games have a reputation for being fun and engaging; more importantly immersive, requiring deep thinking and complex problem solving. We believe educational games are essential in the next generation of e-learning tools, which has lead us to research an SE educational Game Development Platform (GDP), called SimSYS (Cooper, 2011). The vision for the GDP includes a comprehensive collection of interacting tools, including a Game Play Specification IDE, Game Play Framework (engine, UI), Player Assessment, and Adaptive Game Play. An overview of the proposed GDP is in (Smith, 2011).

Our long term vision for SimSYS is to allow students to play games that explore and deeply understand the complex dependencies among SE stakeholders, their activities, and the engineering artefacts they create. For example, how does a decision made by a Requirements Analyst constrain the decisions made by a designer, tester, and project manager? Ultimately any one decision made by a team member can impact the entire project in unexpected ways.

The Agent-oriented Paradigm (AOP) is an alternative approach for constructing software systems that is well-suited for modelling human interaction such as collaboration, negotiation, conflicts, and so on. AOP is based on the concept of an agent, which are software entities that are situated, autonomous, flexible, and social (Wooldridge, 2009). Agents sense the environment and perform actions that change the environment. They have control over their own actions and internal states; they can act without direct intervention from humans. Agents are responsive to changes in the environment, goal-oriented, opportunistic, and take initiatives. They interact with other agents (software, human) to complete their tasks. The agent-oriented approach is beneficial in systems that (O'Malley, 2001): require complex/diverse types of communication; have behaviour that is not practical/possible to specify on a case-by case basis; involve negotiation, cooperation and competition among different

D. Tull A., S. Smith T. and M. L. Cooper K..
TOWARDS AN AGENT-ORIENTED FRAMEWORK FOR SERIOUS GAMES - Architecting with Behavioural Software Agents.
DOI: 10.5220/0003621001920199
In *Proceedings of 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2011), pages 192-199
ISBN: 978-989-8425-78-2

entities; must act autonomously; and is expected to expand or change. Recently, the agent-oriented paradigm has been applied to games. Agent-oriented design solutions have been proposed for intelligent gameplay, behaviour adaptation, and computer human interaction (Dignum, 2009; YingYing, 2009; Goschnick, 2008; Shukri, 2009). We believe AOP is an excellent match for the SE education GDP. Our conjecture is that a new agent-oriented version of an architectural style that explicitly models agent behaviour is needed.

In this position paper we identify high level requirements (goals) related to agent-oriented architecture for SimSYS (Section 2). In Section 3, we present the preliminary architectural solution for (part of) SimSYS – the Game Runtime Systems component, including a decomposition of the Actor Framework. A discussion of how it addresses the requirements issues is in Section 4. Related work is briefly discussed in Section 5. Our conclusions and future work are in Section 6.

2 HIGH LEVEL REQUIREMENTS (GOALS) FOR SIMSYS

The key goals for our GDP are summarized here, along with their impact on the framework (engine, UI) solution.

Game Playability: To keep the games fresh and interesting to players over time, the game needs to present different "twists" each time. Never should a player, for instance, enter the same virtual project stakeholder meeting twice. It should be clear that the outcomes of the player's actions are never predetermined and always contain an element of risk. As such, the graphic interface and underlying engine need to support the dynamic generation of new scenarios and must determine the results and consequences of a player's interactions in real-time.

Furthermore, multiple levels of difficulty must be supported, which can be used to gently introduce players to the controls and the objectives of the game through early levels before thrusting them headlong into a chaotic and fast-paced environment. The stakeholder meeting from level 1 might involve a patient and informed customer that requires almost no player interaction. This should quickly give way in later levels; the player must aggressively build on previous experiences to complete later challenges. This can be seen as an improvement over burdening the player with a separate tutorial or user manual. Additionally, this capacity for variation in difficulty levels provides mechanisms to prevent the player from becoming frustrated with overly difficult or overly simple scenarios; the game should be difficult, but not too difficult, and most certainly not too easy.

Finally, immediate feedback to the player regarding their performance, or success, needs to be tracked and presented. For practical pedagogical reasons, the player must know the outcomes of their actions early. If the player does not know when that stakeholder meeting went sour, they will have difficulty identifying why. If they cannot identify why, they will have great difficulty learning what to do differently next time; which only serves to frustrate the player further.

Modifiability: The GDP is going to need to evolve over time and will need to be modified. For example, the agent behaviour in early releases may be quite straightforward; more interesting or better performing reasoning techniques from the AI community could be investigated and adopted over time. These early agents may be simple fuzzy state machines or rule-based systems, but later agents could go much further, performing as intelligent adversaries or collaborators. The complexity of games that can be scripted can become richer, and more sophisticated. The graphic interface can become easier to use. Component based solutions are well suited to address this issue.

Flexibility: New SE games are going to be needed for different target players including high school; college, university (undergraduate, graduate); and lifelong or industry training. New games will also be needed to keep pace with the evolving state-of-the-art in the discipline; the GDP needs to be flexible. For example, agile methods have recently been added to undergraduate SE courses. Additionally, it would be ideal for the GDP to have the capacity to be extended beyond SE education, and into other forms of education, particularly for business applications. Script-based solutions have been proposed to address this goal; we adopt this approach as a current "best practice" in game development.

Usability: A great game concept may be terribly received by the gaming community because of gameplay frustrations. For this reason it is crucially important to analyze player interactions to understand how user interface design can complement the game flow to reduce frustrations and empower the player. However, user interface design and game play design must be balanced with other quality constraints such as performance, concurrency, portability, and scalability to meet the appropriate level of responsiveness.

3 SimSys: AOP IN GAME DESIGN

3.1 Game Play

One of the primary goals of SimSys is to provide a robust learning experience that is not possible with less sophisticated systems. The running example in this paper will be that of requirements elicitation.

Requirements elicitation is the process of gathering system requirements from a stakeholder or stakeholders. Errors during this phase are extremely expensive in the long run. Unfortunately, this is also one of the most difficult processes to teach in games.

The high level of interaction between elicitor and stakeholder makes non-agent based approaches quite limited. Each interaction would need to be scripted beforehand. The cost of producing such an elaborate content model quickly becomes infeasible for even modestly sized scenarios.

However if using an agent-based approach the agent's initial state is modelled and non-deterministic behaviour can be achieved through interactions with the player and other agents. To illustrate, we will use the following example scenario throughout our description of SimSys:

The player is a requirements engineer for a virtual company. This company has been contracted to design a new inventory-management system for a regional retailer chain. Today is the player's first meeting with the customer, and the player must walk away from this meeting with a good overall understanding of the customer's requirements.

After introductions, the player is given the opportunity to ask the customer, "Who uses your inventory-management system?"

The customer considers this question and responds, "Well, we have two types of users: store employees and website users."

At this point, the player is given the opportunity to ask for more detail about the concepts present in customer's response, "What exactly do you mean by 'website users'? What do they do?"

The customer responds, "Well, I guess there are two types of website users: customers viewing whether an item is in stock, and site administrators."

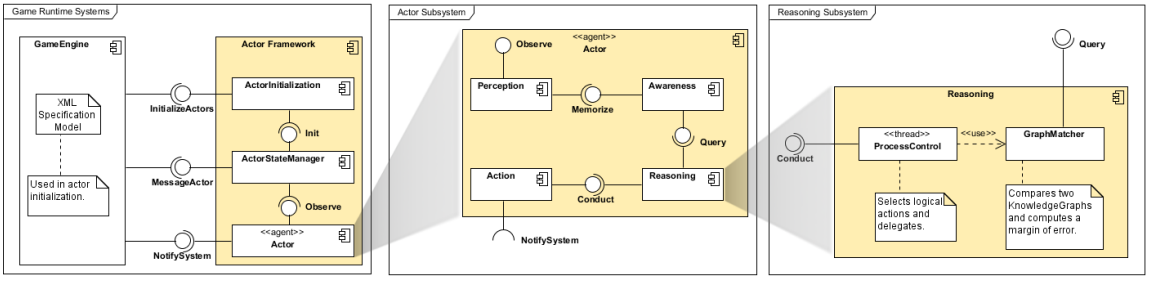
The player continues to ask more questions until the meeting is over. Slowly, the player is coaxing out a model of the customer's business domain.

3.2 Game Runtime Systems

The new architecture proposed in this work (Figure 1) is an agent-oriented, component version of the adaptive object-model architectural style (Yoder 2002). At the top level, the architecture has two components: Game Engine and Actor Framework (Figure 1a). The Game Engine is responsible for handling all graphics, audio, processing user input, and managing game mechanics (i.e. graphical) state. When the player enters the virtual stakeholder meeting, this component displays the office. When the player interacts with an object in the office, this component gathers and processes the physical input. When the player moves around the office, this component keeps track of their location. In this work we focus on the Actor Framework, as the strategic decision making capabilities (agent-oriented behaviour) are allocated to this component.

The Actor Framework has three components that handle the initialization, state, and agent-oriented strategic decision making capabilities of the Actors. Figure 1b illustrates the components of the Actor's strategic decision making, which includes components responsible for perception, awareness, reasoning and action. The reasoning component is further decomposed in Figure 1c. The Actor component is described in more detail in Section 3.3.

Decoupling the Actor Framework from the Game Engine provides the capacity to radically change agents while maintaining minimal changes to the overall game. From the perspective of player-actor interactions, modifying our stakeholder meeting to become, say, a project meeting then simply becomes a matter of replacing agents.



(a) Game Runtime Systems (b) Actor Subsystem (c) Reasoning Subsystem

Figure 1: Preliminary Agent-oriented Component-Based Architecture.

To integrate with the Actor Framework, the Game Engine must first provide an Actor content model to the ActorInitialization component. This contains, effectively, the identity of an actor: who they are, and what they can do. Second, it must notify the ActorStateManager when events pertinent to the Actors decision making processes occur, and make callback functions available for Actor Actions to notify the GameEngine of an Actor's actions.

ActorStateManager Component. The ActorStateManager component is responsible for notifying Actors of environmental conditions affecting the Actors within the ActorFramework component as well as managing interactions with the GameEngine component. Managing the Actors includes activating/de-activating Actors (i.e. allot a thread from a thread pool) and communicating messages to Actors. In our meeting example, this component would act as our stakeholder's eyes, ears, and voice to the environment around him.

ActorInitialization Component. The ActorInitialization component is responsible for interpreting a client provided specification model of Actors and their configuration for the purpose of instantiating Actors. The ActorInitialization component also creates and instantiates all Actors and the meta-descriptions of Actors' goals, knowledge, and capabilities. This entails creation of an ontology graph that represents an actor's "brain". Concepts are initialized; relationships are created to link them; attributes are attached. Given a specification input, a real, functioning agent is produced as an output.

Actor Component. The Actor Component is responsible for receiving incoming stimuli from the AgentStateManager, interpreting the data, determining any necessary course of action to take, and executing a plan for handling the data.

Let us consider this process in light of the example stakeholder meeting in Section 3.1. In our example, the player's first question was a request for elaboration on the system's users. Within the GDP the following sequence of events would transpire to return a response to the player:

1. The player is given a multiple choice-multiple select menu of options to choose from and the player has chosen to ask for elaboration on the system's users. A message is sent to from the GameEngine to the ActorStateManager which interprets the player's selection and generates a logical expression is formed that expresses the statement "Who uses your inventory-management system?". (The meta-model for representing the Actor's knowledge is discussed in Section 3.3.2, in addition to an instantiated example).

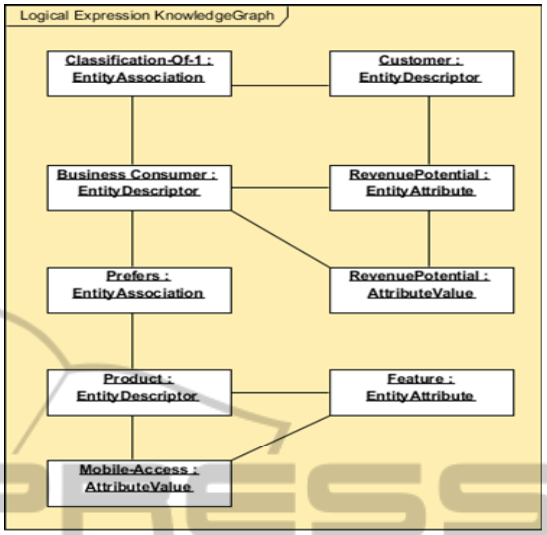


Figure 2: Model of an Actor KnowledgeGraph.

2. The ActorStateManager sends the LogicalExpression to the addressed stakeholder Actor and also sends a carbon copy to any active Actor also present in the game environment. This message is queued by the Actor as an observation to be interpreted.

3.3 Actor

The Actor system models a non-player character's decision making processes and may initiate interactions with players, other Actors, or the game environment. This is accomplished by interpreting the surrounding game environment (Perception), storing the stimuli as memories (Awareness), reasoning over goals with respects to knowledge (Reasoning), and taking action to fulfil the Actor's goals (Action). For further details of each component see Figure 1b and the detailed discussion of each component below.

3.3.1 Perception Component

The responsibility of the Actor's Perception component is to interpret incoming observations from the ActorStateManager into knowledge that is stored in the Awareness component. The Actor receives a message from the ActorStateManager. This message includes a LogicalExpression that describes the observation being made. The Perception component will parse the LogicalExpression into a KnowledgeGraph and

enqueue the message in the Awareness component. Later the Reasoning component can interpret the new knowledge and determine if there is an appropriate response.

The example we described in Section 3.1 refers to a stakeholder Actor that the player interacts with. In terms of game play the player is given a chance to ask the stakeholder to confirm their understanding of the requirements. The ActorStateManager will form a LogicalExpression describing a requirement in terms similar to the stakeholder's own conceptual model of knowledge. This model is expressed and captured in a **KnowledgeGraph** (refer to Figure 2).

An example query that could be made by the player could be something similar to the following plain English statement "A Business customer prefers a product that is able to be accessed from a mobile device." The stakeholder would be given the chance to either agree or disagree with this statement and may elaborate on the player's statement by describing additional desirable product features or other types of customers who also would like a product that is accessible from a mobile device.

3.3.2 Awareness Component

The responsibility of the Actor's Awareness component is to manage the internal content model of the Actor's environment, goals, and capabilities (i.e., maintain its knowledge). This component enables Actors to analyse incoming observation stimuli, create associations to related memories, and use this information to process logical expressions. The Actor Awareness meta-model is illustrated in Figure 3. At the center of the meta-model is the KnowledgeGraph. It is a representation of the Actor's knowledge.

In the example statement "A Business customer prefers a product that is able to be accessed from a mobile device" the KnowledgeGraph will contain a logical definition (similar to Figure 2 above), where the EntityDescriptors from the meta-model are instantiated in terms of the Actor's goals.

3.3.3 Reasoning Component

The responsibility of the Actor's Reasoning component is to interpret stimuli and select the best course of action. A decomposition of the Reasoning component including conceptual classes used in the decision making process is shown in Figure 1c.

An Actor's Reasoning component will evaluate a LogicalExpression in the following steps:

1. The Reasoning component's ProcessControl will delegate to the GraphMatcher the task of comparing the next incoming stimuli to the Agent's knowledge.

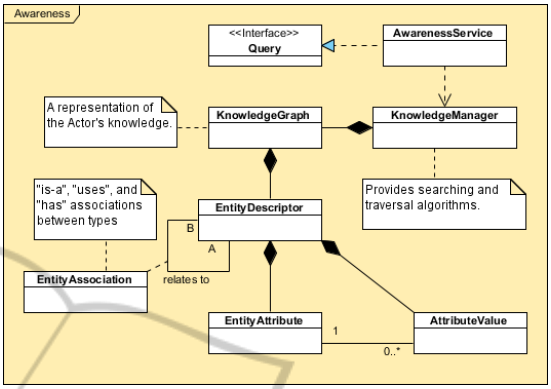


Figure 3: Actor Awareness Meta-Model.

2. The GraphMatcher will retrieve two KnowledgeGraphs from the Awareness component using the Query interface. The first graph is a representation of the Agent's knowledge and the second is a representation of the incoming stimuli.

3. The ProcessControl receives from the GraphMatcher a statistic representing the Actor's relative agreement with the stimuli. This figure is based on the amount of contradicting knowledge, supporting knowledge (i.e. matches), and the lack of supporting knowledge.

4. The ProcessControl uses the context of the stimuli to form an appropriate action to take. For instance, if the stimuli originated from a direct question, a heavily weighted response would be to respond with either a statement of agreement or disagreement and elaboration.

3.3.4 Action Component

The responsibility of the Actor's Action component is to maintain a list of actions to take and periodically execute them. One important example action is the process of integrating new stimuli into their knowledge representation; other actions include interacting with the environment, such as responding to the player's questions using a configured GameEngine callback. The use of GameEngine callbacks would need to be configured in the ActorBehaviormodel (see Section 4 for a discussion of the Actor in the XML input specification model).

3.4 XML Specification Model

The specification model provides instructions to the

GameFramework for generating Actors and their awareness of the environment and their capabilities. At runtime the client system will supply an ActorModel, an ActorKnowledgeModel, and an ActorBehaviorModel. These three content models will be interdependent meta-descriptions that when interpreted by the ActorInitialization component will fully describe the initial state of the Actor system. It should be noted that due to memory constraints in video game systems it is likely that the ActorInitialization functions must be capable of conducting resource allocation of Actors at runtime.

The ActorModel declares the identifiers that the client system will use to address Actors using the MessageActor interface. This would also prescribe a specific ActorKnowledgeModel and an ActorBehaviorModel. By separating these meta-descriptions the system could reuse content models for multiple Actors.

The ActorKnowledgeModel content is interpreted to generate an internal representation, referred to as the meta-ontology, which will describe things, types, and capabilities in an Actor's Awareness component. This internal awareness model will describe Entities, EntityTypes, and Actions (Yoder, 2002). Actors may also maintain associativity and identity relationships between EntityTypes, which could describe polymorphic relationships between Entities. Note that we use the Adaptive Object-Model ontology to describe the conceptual entities (Yoder, 2002).

The ActorBehaviorModel first declares a meta-pedagogy of motivations and goals that the Actor attempts to maintain or achieve. Secondly, Entities and Actions in the ActorKnowledgeModel are associated to these goals; this combination drives the Actor Reasoning process managed by the Awareness Component.

4 DISCUSSION

We recognize that not all QOS Attributes have been addressed with this architecture. One point of future research is the prioritization of other quality of service attributes and a subsequent re-evaluation of the architecture. We have summarized our current knowledge of concerns in discussions below.

Game Playability. Implementation of an agent-oriented paradigm in video games will enable games to respond to non-linear game play options. By scripting goal oriented scenario objectives in the to be selected and played at runtime will enable players to exercise creative problem solving skills. Also, because the interactions (human-actor as well as actor-actor) will be governed by goals the Actor content model can be scripted to respond to stimuli from both the player(s) as well as other Actors creating unexpected behaviour and a richer immersive game world.

Game Scenario Flexibility. The Actor content model could be written in a way that enables both simple scripted behavior as well as non-deterministic behavior in Actors. Creative content design of the Actor specification would be performed without the need to recompile game engine source code. The only time that source code would need to be modified or recompiled is when the Actors need a new parameterized callback function to exercise some activity they didn't have the ability to perform before. In most cases, this would not be the case. New responses could be scripted in the content model using existing callbacks (such as movement, attack, or other game specific behavior).

GDP Modifiability. Future enhancements to the ActorFramework component infrastructure could include features to handle performance constraints. Due to the non-deterministic nature of agents, predicting and profiling specific Actor behavior is likely to be problematic. There are different approaches possible to handle performance analysis and although it is not clear at this time the most effective way to provide insight into the performance of non-deterministic systems it is clear that this is an area for future research.

Framework Overhead. The additional cost of packaging the SimSys Actor Framework needs to be reviewed. The usage of callback functions for Actions is a measure in the presented architecture to mitigate the performance concerns of an additional layer of abstraction. Additionally, performance gains could be attained through the process of fine-tuning the interpreted interactivity model. Optimizations can be made to tailor to a more precise desired Actor behaviour and potentially realize performance improvements through simplification of the Actor's meta-ontology or meta-pedagogy. This would have the desirable side effect of teaching the Actor to skip unnecessary steps in logic calculations.

Concurrency. At this time, the presented architecture is capable of supporting Actors in a multi-threaded environment. In the case that the game platform is not capable of a multi-threaded architecture, the system would need alternative functionality to run in a single threaded process. We plan to evaluate game developer needs and the

feasibility of implementing a single-threaded configuration of the ActorFramework component.

Portability and Integration. Integrating the GameFramework into other systems would require an undergraduate level of experience programming with the C++ programming language with knowledge of behavioural design patterns such as the Observer pattern or the Command pattern.

5 RELATED WORK

Research on agent-oriented software architecture is emerging as a topic of great interest and value. Due to space constraints, we select sample research that relates to technical quality of service requirements and modelling the behaviour of agents.

At the architecture level, response-time performance, agent behaviour and communication have received attention (Shukri, 2008; Jepp, 2010); less work is available on the concurrency (Duvigneau, 2003), scalability (Luo, 2010), or extensibility/evolution of games (Lee, 2002) or GDPs.

From the real-time community, Lee's architecture (Lee, 2002) is the closest with respect to our component based design concept. The components in this architecture are encapsulated with well-defined interfaces; components of different characteristics, functionalities, and implementations can be used to form real-time agents. The task-scheduling component deals with requests arriving at unexpected time points that are of different levels of urgency and importance. It dynamically manages overload conditions by plugging in alternative components. To include additional components for system evolution, however, would require code modifications to the game (not scripted). The architecture presented in (Lee, 2002) does not focus on the agents from a social, behavioural perspective.

From the AI community, the multi-agent architecture by (Kobti, 2007) focuses on the social, learning, behavioural perspective. Here, agents are intelligent, interact, and make decisions in positional games. The overall architecture is based on the General Game Playing architecture (http://games. stanford.edu). The architecture presented in (Kobti, 2007) does not focus on the technical quality of service requirements.

6 CONCLUSIONS

The characteristics of the SimSYS GDP include a rich collection of quality of service requirements: modifiability of the platform; flexibility of the games; playabilty of the games; and usability of the games and platform. From a software architecture perspective, the evolvability and playability requirements have lead us to a domain specific architecture that is agent-oriented and component based; new/modified games are defined with a domain specific modelling language. Our position is that our unique combination of the agent-oriented paradigm and component based design is well suited for this problem domain. In the future we plan to refine and rigorously verify the architecture; model checking the agent-oriented architecture would be of great interest. An empirical evaluation with a prototype is underway. In addition, the usability requirements (response time performance, concurrency, portability, scalability) will be thoroughly investigated; their trade-offs with the playability and system modifiability requirements assessed.

REFERENCES

Cooper, K., 2011. SimSYS Project Website, available at: http://www.utdallas.edu/~kcooper/SimSYS.

Dignum, F., Westra, J., van Doesburg, W., and Harbers, M., 2009. Games and Agents: Designing Intelligent Gameplay, in *International Journal of Computer Games Technology*.

Duvigneau, M., Moldt, D., and Rölke, H., 2003. Concurrent Architecture for a Multi-agent Platform, *Agent-Oriented Software Engineering III, LNCS.*

Goschnick, S., Balbo, S., and Sonenberg, L., 2008. ShaMAN: An Agent Meta-model for Computer Games, in *Proceedings 2nd Conference on Human-Centered Software Engineering.*

Jepp, P. Fradinho, M., and Pereira, J. M., 2010. An Agent Framework for a Modular Serious Game. In *Proceedings 2nd International Conference on Games and Virtual Worlds for Serious Applications*.

Kobti, Z. and Sharma, S., 2007. A Multi-Agent Architecture for Game Playing. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games*.

Lee, J. and Zhao, L., 2002. A Real-Time Agent Architecture: Design, Implementation and Evaluation. In *Intelligent Agents and Multi-Agent Systems, LNCS.*

Luo, J. and Chang, H., 2010. A Scalable Architecture for Massive Multi-player Online Games Using Peer-to-Peer Overlay. In *Proceedings 2010 12th International Conference on Advanced Communication Technology*.

O'Malley, S. and DeLoach, S., 2001. Determining When to Use an Agent-Oriented Software Engineering Paradigm, in *Proceedings of the Second International Workshop On Agent-Oriented Software Engineering*.

Shukri, S. and Shaukhi, M., 2008. A Study on Multi-Agent Behavior in a Soccer Game Domain, in *World Academy of Science, Engineering and Technology*.

Smith, T., Tull, A., Cooper, K., and Longstreet, C, 2011. Can simulation training games help shift the paradigm of software and systems engineering education?, SIMULTECH 2011 (submitted).

Wooldridge, M., 2009. *Introduction to MultiAgent Systems*, John Wiley & Sons, 2nd edition.

Yingying, S. and Grogono, P., 2009. An approach of real-time team behavior control in games, in *Proceedings 21st IEEE International Conference on Tools with Artificial Intelligence.*

Yoder, J and Johnson, R., 2002. The Adaptive Object-Model Architectural Style. In *Proceeding of the 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance.*