Towards a Virtual Agent Control Architecture Using a Spatiotemporal Data Management Framework

Nikos Pelekis[1], Spyros Vosinakis[2], Themis Panayiotopoulos[1], Yannis Theodoridis[1]

[1]Department of Informatics, University of Piraeus
E-mail: {npelekis | themisp | ytheod}@unipi.gr

[2]Department of Product and Systems Design Engineering, University of the Aegean, Ermoupolis, Syros, Greece, Tel: (+30) 22810 97000
E-mail: spyrosv@syros.aegean.gr

Abstract

Virtual Agents are autonomous embodied entities in virtual environments that look and behave like living organisms. They are usually equipped with sensors that read the state of the environment and effectors that apply actions on other entities. Believable agent behavior requires non-omniscience: each agent should contain its own representation of the environment based on its sensing abilities and its interaction history. However, the storage and maintenance of a multitude of different knowledge bases in a dynamic environment with a large number of agents raises memory and performance issues, as data is duplicated and a lot of calculations are repeated. This paper presents an architecture for multi-agent virtual environments that is based on a spatiotemporal database, which maintains an abstract representation of the environment and the moving entities, and delivers all the sensory information required by the agents. The use of the spatiotemporal database enhances the agents' reasoning about the position and motion of other entities by providing a number of queries for spatiotemporal reasoning.

Keywords: Virtual Environments, Virtual Agents, Spatiotemporal Databases, Moving Object Databases, Multi-Agent Systems

1 Introduction

Virtual Agents are autonomous entities in virtual environments, whose appearance, motion and behavior imitate a living organism (Aylett et al, 2000). To satisfy this need for autonomy, the distinguishing characteristic of a virtual agent compared to other animated entities in the environment, is that it does not simply execute a predefined animation sequence, but in order to satisfy its goals, it is able to take decisions on its own and adapt its actions to the state of the environment. It is, therefore, equipped with mechanisms for monitoring changes in the environment (sensors), for executing actions upon it (effectors), and for taking decisions about its actions. Virtual Agents are an important element of Virtual Reality applications as they can enhance their believability, autonomy and interactivity, and they have been effectively used in various application areas, such as education, entertainment, simulation, etc.

An important factor for enhancing realism and believability of agent behavior, which is considered necessary in applications that simulate human activity, is embodiment, i.e. the case where agents are subject to the constraints of the environment. Embodiment implies that the agent sensors and effectors are limited by their abilities and their position in the environment. Consequently, they should perceive only the objects that lie within their field of view and they should be able to act efficiently in dynamic environments by making predictions about future events and assumptions about the state of objects that are not directly visible. These processes require sophisticated spatial and spatiotemporal representations and reasoning. In current implementations, each agent maintains individual representations based on its perceptual abilities and uses them to reason about its actions (e.g. Conde and Thalmann, 2006; Peters and O'Sullivan, 2002). However, this approach raises performance issues in multi-agent environments, especially in cases with a large number of agents, as data duplication and repeated calculations take place. The authors claim that multi-agent virtual environments could take advantage of recent advances in spatiotemporal databases and use a global data management framework to store the spatiotemporal representation of the environment and its dynamics. This approach should avoid unnecessary calculations of each individual agent's reasoning mechanism, and thus lead to implementations that balance between believability and performance.

This paper proposes an architecture for multi-agent virtual environments that takes into advantage a spatiotemporal data management framework. The core idea is to provide the virtual agent with the appropriate functionality so as it will not be necessary to store in memory all the knowledge about the surrounding environment, but whenever information is required to extract it from a spatiotemporal database. This implies that the system will have the capability to dynamically store and update the spatiotemporal setting of the world at different time instances, as well as to extract knowledge by querying the world. Such queries may concern not only the current state but historical and more often future states. To address this issue for a dynamic environment the authors employ a representation that preserves the *trajectory* followed by the agent so far, and is also aware of its current motion properties (i.e. position, speed, direction). Current Database Management Systems (DBMS) are not capable of efficiently storing and querying such complex moving objects, and therefore, solutions coming from the field of Moving Object Databases (MOD) (Güting et al, 2000) have been adopted. The authors present a prototype implementation of the proposed architecture and a simulation of an airport environment as a case study.

The paper is structured as follows: In the next section we present the related work in moving object databases for the representation of the environment's dynamics. In section 3 we present the proposed virtual environment model and describe in detail the spatiotemporal data management and the agent architecture. In section 4 we present a case study using a prototype implementation of the environment. Finally, in section 5 we state the conclusions and the future work.

2 Moving Object Databases in Dynamic Virtual Environments

The last decade substantial research work has been carried out focusing on modeling and querying spatio-temporal databases. This section presents a parallel line of research focusing on modeling trajectories as spatio-temporal objects (the so-called moving objects). Researchers in the field of Moving Objects Databases (MOD) have been studying the representation issues of trajectories into computer systems aiming at keeping track of object locations, as well as supporting location-aware queries. In our case, a spatiotemporal data management framework will be employed to store the virtual environment and its dynamics. We adopt the idea of the MOD and extend it appropriately as to support perception and reasoning tasks in multi-agent environments.

A straightforward approach widely used in industry is to model a moving object by generating periodically a location-time point of the form (x, y, t), indicating that the object is at location (x, y) at time t. Points are stored in a database, and a database query language is used to retrieve the location information. This method has several shortcomings, as it does not enable interpolation or extrapolation, while it leads to a precision/resource trade-off, and inefficient software development. In the MOD literature, there are two main approaches to model trajectory data: one for querying current and future positions of the moving objects in (Sistla et al, 1997; Wolfson et al, 1998; Wolfson et al, 1999) and the second for querying past trajectories of the moving objects in (Erwig et al, 1999; Guting et al, 2000; Forlizzi et al, 2000; Lema et al, 2003).

Querying current and future positions must consider the problem of managing the positions of a set of entities in a database. However, to keep the location information up-to-date we encounter an unpleasant trade-off. If updates are sent and applied to the database very often, the error in location information in the database is kept small, yet the update load becomes very high. Indeed, keeping track of a large set of entities is not feasible. Conversely, if updates are sent less frequently, the errors in the recorded positions relative to the actual positions become large. This problem was explored by Wolfson et al. (Sistla et al, 1997; Wolfson et al, 1998; Wolfson et al, 1999) which have developed a model, called MOST, that allows one to store the motion vector rather than the objects' positions in the database, avoiding a very high volume of updates. In Wolfson and colleagues' work, the location of a moving object is simply given as a linear function of time, which is specified by two parameters: the position and velocity vector of the object. Thus, without frequent update messages, the location server can compute the location of a moving object at a given time t through linear interpolation: $y(t) = y_0 + v(t - t_0)$ with time $t > t_0$. An update message is only issued when the parameter of the linear function, i.e. v, is changed. It offers a great performance benefit in linear mobility patterns, but performance suffers when the randomness of the mobility pattern increases.

The need for capturing complete histories of objects' movement has promoted the investigation of continuously moving objects. Clearly as location data may change over time, the database must describe not only the current state of the spatial data but also the whole history of this development. Thus it should allow to go back in time at any particular instant, to retrieve the state of the database at that time, to understand the evolution, to analyze when certain relationships were fulfilled, and so forth. This approach was developed by Guting and colleagues (Erwig et al, 1999; Guting et al, 2000; Forlizzi et al, 2000; Lema et al, 2003). They described a new approach where moving points and moving regions are viewed as three-dimensional (2D space + time) or higher-dimensional entities whose structure and behavior is captured by modeling them as abstract data types. Such types and their operations for spatial values changing over time can be integrated as base (attribute) data types into object-relational, object-oriented, or other extensible database management system. The final outcome of this work was a system that implements the above described moving objects data model and query language completely integrated into a DBMS environment (Almeida et al, 2006). More specifically, the prototype has been developed as an algebra module in SECONDO's extensible environment (Dieker and Guting, 2000).

In order to support dynamic location-aware queries for current or near future positions of virtual agents the model of Wolfson et al. would be sufficient, while for historical management of their actions the work of Guting et al. is adequate. However, in virtual multi-agent control architectures both requirements exist, while there is a further demand of extracting higher level knowledge from the dynamics of the environment that could be useful for the spatiotemporal reasoning of agents. As such, focusing on the representation and querying of continuously as well as discretely moving agents, we adopt the Hermes datatype-oriented model (Pelekis et al, 2006; Pelekis and Theodoridis, 2006) that satisfies the above precondition and supports the representation of moving objects under an object relational platform. More specifically, Hermes has been designed as an extension to any extensible Object-Relational Database Management System (ORDBMS). Its main functionality is to support the storing and querying of complex data types as the trajectories of continuously moving Intelligent Virtual Agents (IVA) are. Such a data type and its corresponding operations are defined, developed and provided as a data cartridge (Pelekis and Theodoridis, 2006). Embedding this functionality offered by the Hermes Data Cartridge (Hermes-DC) in a Data Manipulation Language (DML) as SQL, one obtains an expressive and easy to use query language for dynamic virtual environment that could be utilized for extracting high level knowledge from the observed motion of animated entities.

3 Virtual Environment Model

Recently, researchers are beginning to address the need for enhancing virtual environments with semantic information in order to obtain a unified knowledge representation layer for interconnecting virtual environments and for being able to embed high-level description of the environment in to the visualization model (Cavazza and Palmer, 2000). Especially in the case of virtual agents, the need to reason about the entities of the environment and their relations so as to take decisions that satisfy their goals, makes it necessary to use an abstract description of the scene and its dynamics, enhanced with domain-specific information in addition to the detailed geometrical description represented in the scene graph. In this section we propose a model of a virtual environment that is enhanced with a spatio-temporal data management framework that enables agents to obtain high-level knowledge concerning the entities and their motion. The generic architecture of the environment is presented in Figure 1.

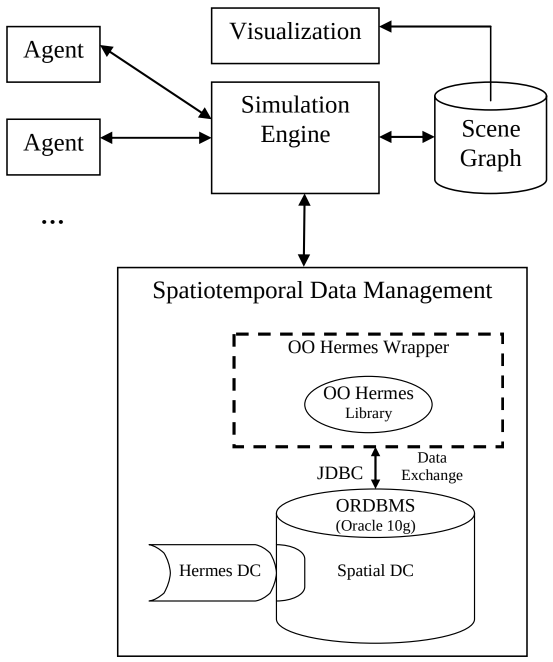

Figure 1: Virtual Environment Architecture

The proposed environment model consists of the following components:

- a typical *scene graph* containing the geometry of all visible objects and their structural relations (group structures, geometric transformations, etc),
- a *simulation engine* that is responsible for calculating the next state of the environment and animating it,
- a *visualization engine* that renders (part of) the environment after every timeframe,
- a *spatiotemporal data management* component that contains the animation history of the dynamic scene graph nodes of the environment and responds to queries concerning the position and motion of the entities and
- a number of *agent processes* that read the state of the environment, decide on their next tasks and send their actions back to the environment.

The scene graph contains all the information needed to render and animate the environment. It is created by the scene designer and can be reused in various kinds of applications and environments. On the other hand, the moving object database is created and updated dynamically based on the agents' actions and contains any spatio-temporal knowledge that may assist the agent's reasoning process.

The simulation engine is receiving the agents' actions and is calculating their effect on the entities based on a set of rules (e.g. gravity, collision, friction, etc.) that depend on the nature of the environment. It is frequently updating the spatiotemporal data management with the current position and orientation of the animated entities, thus ensuring a consistent abstraction of the environment. Finally, it is responsible for generating the agent's sensory input, based on their perceptual abilities: it returns the geometric properties of the entities that the agents are able to 'see', and it serves as a medium between the agents and the spatiotemporal data management that lets them reason about the spatiotemporal dynamics of the environment.

3.1 Spatiotemporal Data Management

Usually, a MOD stores and manipulates a 3D polyline representing the historical trajectory of an object (i.e., a sequence of 3D line segments (x, y, t), where each segment represents the continuous development of the moving object during sampled locations). Hermes-MDC extends this idea in a way that the trajectory of an IVA can be defined as a sequence of different types of simple functions. The general idea is to decompose the definition of the agent's motion into several definitions, one for each of the simple functions, and then compose these sub-definitions as a collection to define the trajectory. Each one of the sub-definitions corresponds to a so-called unit moving type. In order to define a unit moving type, we need to associate a period of time, called *Period<SEC>*, with the description of a simple function, called *Unit_Function*, that models the behavior of the moving type in that specific time period. The *Period<SEC>* implies a closed-open time interval (i.e. [b, e), where b is the beginning and e is the ending timepoint of the period) with granularity at the second level (other granularities are also supported i.e. minute, hour etc). The *Unit_Function* is an object type, defined as a triplet of (x, y) coordinates together with some additional motion parameters. The first two coordinates represent the initial (x_i, y_i) and ending (x_e, y_e) coordinates of the sub-motion defined, while the third coordinate (x_c, y_c) corresponds to the centre of a circle upon which the object is moving in case of circular motions. Whether we have constant, linear or arc motion between (x_i, y_i) and (x_e, y_e) is implied by a flag indicating the type of the simple function. Since the management of both motion history and current (and near future) motion of an IVA's is required, we further let a *Unit_Function* to model the case where an IVA is located at (x_i, y_i) and moves with initial velocity v and acceleration a. Whenever this sub-motion ends we appropriately update the corresponding *Unit_Function*. For a more analytical discussion the interested reader is referred to (Pelekis and Theodoridis, 2006).

The proposed architecture of a virtual environment enhanced with a spatio-temporal data management framework is presented Fig. 1. In order for the API of our IVE development framework to interact with our spatiotemporal data management infrastructure we have developed a wrapper in Java wherein we simulate the datatypes and the corresponding methodologies embedded in the ORDBMS. In this way we can embed PL/SQL scripts that invoke object constructors and methods from Hermes-DC. The JDBC pre-processor integrates the power of the programming language with the database functionality offered by the extended SQL and together with the ORDBMS Runtime Library generate the application's executable. All the data transfers between the two parts are exchanged using XML-based documents.

3.2 Agent Architecture

The proposed agent architecture (Fig. 2) is based on the SimHuman environment presented in (Vosinakis and Panayiotopoulos, 2001; Vosinakis and Panayiotopoulos, 2005) and has been enhanced with the support of the spatiotemporal data management framework. It employs the perception and task execution processes as a means to connect high-level behavior with low-level actions and to allow virtual agents to interact with the environment in a believable manner. Each agent is using perception to collect information about the objects that are currently in its field of view and to pose queries concerning their spatiotemporal properties such as type of motion, path similarities, etc. The perception process generates and updates its beliefs and, based on an abstract representation of the environment, the agent takes decisions that result in high-level tasks. Those are further analyzed to actions by the task execution module and executed as animation sequences in continuous timeframes.

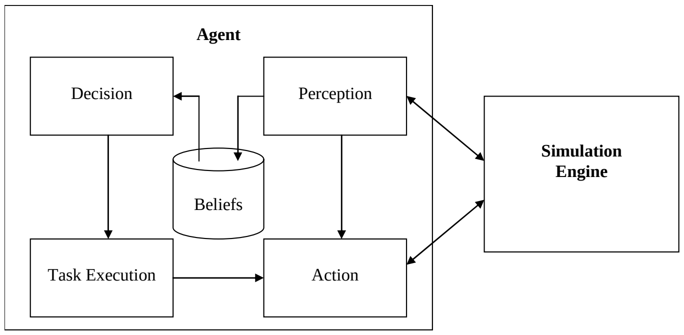

Figure 2: Agent Architecture

A virtual environment is usually a place where dynamic and unpredictable changes take place, and it is therefore necessary for a virtual agent to use its senses frequently and to maintain a model of the world as coherent as possible. The perception mechanism is the part of the agent that models its 'eyes'. It receives from the environment the list of entities that are in its field of view and their current attribute values, as well as any possible messages from other agents or the world. Additionally, it submits queries to the simulation engine concerning spatiotemporal properties of the observed entities and receives the results, and, finally, it generates and updates the set of beliefs. A detailed description of this process can be found in (Vosinakis and Panayiotopoulos, 2003a). The beliefs define an abstract representation of the world and its entities that is used by the decision process to derive its intentions.

The most important feature of a virtual agent is to be autonomous, i.e. to act on its own. Therefore, there has to be a process that defines its behavior. In the proposed architecture, this process lies in the decision module and is the part that models the agent's brain. It can initialize the execution of a complex task or terminate the running one, based on the current state of the environment and the agent itself. It does not only depend on the beliefs, but can also submit queries directly to the perception module to examine if a certain condition is true. There are a number of methods and implementations available in the literature concerning the decision process of an agent based on a symbolic representation of the environment, such as rule-based systems, planning, etc.

The Task Execution module's duty is to analyze tasks into simple actions and execute them sequentially, conditionally or in parallel, sending the appropriate orders to the Action module. Each task is an action combination that achieves a specific result, e.g. having lunch, going to sleep, meeting someone, etc. The way in which a task is executed can be programmed using a scripting language, which defines in a procedural way how the actions will be combined (Vosinakis and Panayiotopoulos, 2003b).

The available actions are the agent's built-in abilities, which may include inverse kinematics, walking, playing an animation sequence, sending a message, etc. Each action is using a set of arguments and is executed by sending *primitive action* sets to the simulation engine. We call primitive actions the commands that an agent can perform in one timeframe, e.g. changing the geometric properties of entities (rotation, translation, position in scene graph), adding new entities to or removing entities from the world, and sending messages to other agents. Each action results to a sequence of primitive action sets that are executed in continuous timeframes. This sequence can be *predefined* or *goal-oriented*. Predefined sequences describe simple actions that are usually based on keyframing, e.g. scratch head, walk a known path, etc. On the other hand, goal-oriented sequences are defined by complex actions, such as catching an object, finding a path to a target, etc.

Primitive actions receive feedback from the world in case they fail, e.g. cannot move arm due to a collision. It, then, depends on the action itself if it will continue its execution despite such a failure or not. There are also cases where an action might fail, not because of a failure of one of its primitive actions, but because the state of the world is such that the action cannot be directly executed, e.g. an inverse kinematics problem that cannot be solved. In cases of failure, the Action module sends the appropriate feedback to the Task Execution.

5. Case Study

In order to demonstrate the applicability of the proposed architecture, we have developed a prototype implementation of a virtual environment that animates people moving around an airport. The application visualizes a simplified model of an airport environment that contains a number of check-in places, shops and departure gates. A flight schedule is randomly generated and continuously updated during the simulation. Each flight is served by one or more check-ins and departs from a single gate at a given time. The environment contains a number of clerks that are serving clients at the check-ins and shops. Travelers enter the airport having a predetermined target flight, move to a check-in place, wander around the airport and finally move to the gate until the flight departs. They interact with one or more clerks during their stay in the airport. The case study has been implemented in Java using Java3D for the visualization of the virtual environment. A screenshot of the environment is presented in Fig. 3.

The 3D environment has a predetermined arrangement of static objects and places. However, the simulation may be parameterized in terms of the number of initial agents, the rate of incoming travelers, the frequency of flights and the execution speed. Furthermore, each traveler agent has different sensorimotor abilities that are randomly assigned to it upon entering the environment. These abilities include its maximum speed of movement, and its rate, distance and angle of perception. Each traveler agent has also different goals concerning its destination flight and its priorities concerning the shops it would like to visit, and the time between a traveler’s arrival and the scheduled flight departure is variable as well.

The actions supported by the agents in this prototype implementation are keyframing animation, walking and messaging other agents. The locomotion engine is using the A* pathfinding algorithm on a simplified graph representation of the airport environment to find a safe path to a given target position or place. Additionally, it is equipped with a simple collision avoidance mechanism to ensure that the agents will follow their path without being blocked by other static or animated agents. An agent can perceive each entity (object or agent) that is visible by it and lies within its perception distance and angle.

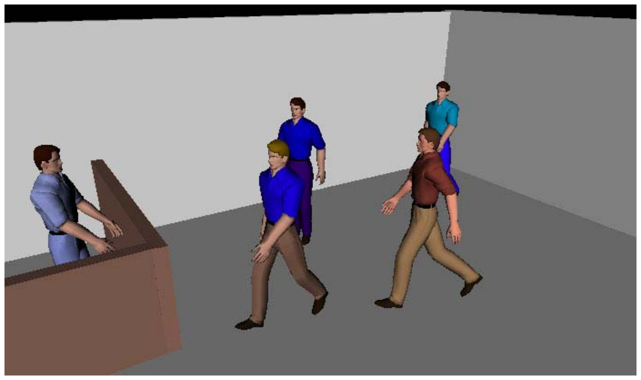
Figure 3: Screenshot of the environment

The spatiotemporal database contains a simplified 2D representation of the airport environment that includes the ground projection of all static geometries and all regions of interest (i.e. gates, shops, check-ins) represented as polygons. It is also continuously updated by the simulation engine about the position and velocity of all agents in the environment that are represented as moving objects. The implementation contains a variety of trajectory manipulation and analysis methods, such as:

- *Queries on stationary objects of the world*: e.g. how many agents are inside a region of interest, which is the closest region of interest of a certain type (e.g. check-in), find the agents that will possibly enter / leave a region in the next minute.
- *Queries on other moving agents*: e.g. agent(s) that will possibly collide with each other or that will have a given spatial relation in a point in time.

All queries are filtered by a simple visibility algorithm, which detects all entities that are currently visible by a given agent. As a result, each agent can submit spatiotemporal queries, but the perception results will include only the visible entities, thus ensuring non-omniscience.

During the execution of their tasks, the traveler agents are utilizing the spatiotemporal database to perceive the status of the environment and decide on their next actions. They combine queries in the following cases: a) to find the check in place that serves their flight and has the minimum people waiting, b) to detect the end of a queue in a check in and move to the respective position, c) to find the nearest shop of interest whilst wandering inside the environment and d) to detect agents that will possibly collide with it and initiate the collision avoidance process.

6. Conclusions and Future Work

We have presented an architecture for multi-agent virtual environments that takes advantage of a spatiotemporal data management framework and maintains an abstract representation of the environment and its dynamics. The proposed system allows virtual agents to perceive the visible entities and reason about their position or motion, find path similarities and predict future position using the manipulation and analysis methods provided by the database. Furthermore, the use of a central database for representing the position and motion of animated entities overcomes the memory and processing burden of keeping internal world representations and observation histories in each single agent. In the future, we are planning to develop more complex case studies involving multiple roles of agents in order to evaluate and refine the proposed system.

Acknowledgements

This research is partially supported by the Pythagoras EPEAEK II Programme of the Greek Ministry of National Education and Religious Affairs, co-funded by the European Union.

References

Almeida, V. T., Guting R. H. and Behr, T. (2006): Querying Moving Objects in SECONDO. In *Proceedings of MDM 2006*.

Aylett, R. and Luck, M. (2000): Applying artificial intelligence to virtual reality: Intelligent virtual environments. In *Applied Artificial Intelligence* 14, 3-32.

Cavazza M. and Palmer I. (2000): High-level Interpretation in Virtual Environments. In *Applied Artificial Intelligence* 14, 1, 125-144.

Conde, T. and Thalmann, D. (2006): An Integrated Perception for Autonomous Virtual Agents: Active and Predictive Perception. In *Computer Animation and Virtual Worlds* 17, 3, 457-468.

Dieker, S. and Guting, R. H. (2000): Plug and Play with Query Algebras: Secondo. A Generic DBMS Development Environment, In *Proceedings of IDEAS 2000*.

Erwig, M., Güting, R.H., Schneider, M. and Vazirgiannis, M. (1999): Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. In *GeoInformatica* 3, 3, 265-291.

Forlizzi, L., Güting, R. H., Nardelli, E., Schneider, M. (2000): A Data Model and Data Structures for Moving Objects Databases. In *Proceedings of SIGMOD 2000*.

Güting, R.H., Bohlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M. and Vazirgiannis, M. (2000): A Foundation for Representing and Querying Moving Objects. In *ACM Transactions on Database Systems* 25, 1, 1-42.

Lema, J. A. C., Forlizzi, L., Güting, R. H., Nardelli, E. and Schneider, M. (2003): Algorithms for Moving Objects Databases. In *The Computer Journal* 46, 6, 680-712.

Pelekis, N. and Theodoridis, Y. (2006): Boosting Location-Based Services with a Moving Object Database Engine. In *Proceedings of MobiDE 2006*.

Pelekis, N., Kopanakis, I., Ntoutsi, I., Marketos, G. and Theodoridis, Y. (2007): Mining Trajectory Databases via a Suite of Distance Operators. In *Proceedings of STDM 2007*.

Pelekis, N., Kopanakis, I., Ntoutsi, I., Marketos, G., Andrienko, G. and Theodoridis, Y. (2005): Similarity Search in Trajectory Databases. In *Proceedings of TIME 2005, Morgan Kaufmann Publishers*.

Pelekis, N., Theodoridis, Y., Vosinakis, S. and Panayiotopoulos, T. (2006): Hermes - A Framework for Location-Based Data Management. In *Proceedings of EDBT 2006*.

Peters, C and O'Sullivan, C (2002): Synthetic Vision and Memory for Autonomous Virtual Humans. In *Computer Graphics Forum* 21, 4, 743-752.

Sistla, P., Wolfson, O., Chamberlain, S. and Dao, S. (1997): Modeling and Querying Moving Objects. In *Proceedings of ICDE 1997*.

Vosinakis, S. and Panayiotopoulos T. (2003a): Programmable Agent Perception in Intelligent Virtual Environments. In *Lecture Notes in Computer Science* 2792, 202-206.

Vosinakis, S. and Panayiotopoulos, T. (2001): SimHuman: A Platform for real time Virtual Agents with Planning Capabilities, In *Lecture Notes in AI* 2190, 210-223.

Vosinakis, S. and Panayiotopoulos, T. (2003b): A Task Definition Language for Virtual Agents, In *Journal of WSCG* 11, 3, 512-519.

Vosinakis, S. and Panayiotopoulos, T. (2005): A tool for constructing 3D Environments with Virtual Agents. In *Multimedia Tools and Applications* 25, 2, 253-279.

Wolfson, O., Sistla, A. P., Chamberlain, S. and Yesha, Y. (1999): Updating and Querying Databases that Track Mobile Units (1999): In *Distributed and Parallel Databases* 7, 3, 257-387.

Wolfson, O., Xu, B., Chamberlain, S. and Jiang. L. (1998): Moving Objects Databases: Issues and Solutions. In *Proceedings of SSDBM 1998*.