A Simulation Framework

Jagiello, J. [1], Eronen, M.[1]

[1] Defence Science and Technology Organisation, Canberra, Australia
Email: jerzy.jagiello@dsto.defence.gov.au, marko.eronen@dsto.defence.gov.au

Keywords: *simulation, modelling, robotics, framework, software*

EXTENDED ABSTRACT

Simulation models of complex systems can be implemented as stand-alone applications dedicated to a particular domain with dedicated and optimised GUIs and supporting tools. Alternatively, the same effect can be achieved by building a generic simulation framework and then implementing the simulation model as a specific instance of that framework. Our framework, build primarily as a testing environment for robotic applications, consists of the repository store and the set of controllers: internal, rule module, tapestry, sensor, output. It follows the standard client server model where the clients store their private states and the server is a repository of shared states for simulation entities and the environment. From the simulation framework point of view, an environment is a collection of entities that are not the focus of a simulation experiment but necessary aid for conducting such an experiment. Rules are used to enact basic changes to the entities within the environment. An emergent behaviour can be observed by the execution of a collection of rules on the environment. The server is the placeholder and distributor of information supplied to entities through their sensor components. Sensors are used to provide entities with a transformed or incomplete perception of the environment, resulting in the entities exhibiting variant behaviour. To allow for flexibility in the way simulation experiments are designed selected components can be reused, extended or redesigned by application developers. In order to overcome limitations of computing resources a distributed rule module protocol has been introduced as well. This paper will present three different types of applications to show the utility of the framework. For example, in the distributed Go*Team game (Figure 2) the players, instead of robots, played a

Figure 1. Asset Protection Simulator

Figure 2. Go*Team Simulator

game across the network. In the asset protection scenario (Figure 1) simulated robots, as well as their control, were supervised by the framework itself. For the mind storm application (Figure 3) a

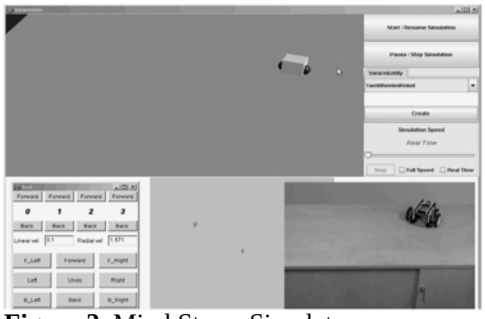


Figure 3. Mind Storm Simulator

physical as well as a simulated robot were controlled in an open loop by one application which was sending commands to a simulator as well as the robot itself.

The rationale behind the design and development of such a framework are presented in this paper together with the distributed rule module protocol for parallel processing of mathematical models.

1. MOTIVATION

In recent years there has been an explosion of research and development activities dedicated to the design and development of simulation frameworks, especially for real time robotic applications (Yeo *et al.* 2004, Xavier *et al.* 2002, Yalcin *et al.* 2005). When authors of this paper started looking for a simulator to test robotic middleware in 2000 the only suitable platform was UMBRA from Scandia (Gottlieb *et al.* 2001, Jagiello *et al.* 2006). Unfortunately, due to export restrictions it was impossible to acquire any technical information regarding the architecture and design not mentioning the software itself. It was decided in 2001 to embark on research activities to design and develop a high performance simulator which allows the testing of robotic applications in "virtual reality" where the dynamics of the robots were simulated by a simulator but control of the robots was performed outside the simulator across the network by robotic applications running on stand alone PCs or robots themselves. It was necessary to have a development environment where applications can be developed and tested outside a hardware platform and later transferred without any modifications to a target system. The fundamental requirement for a simulator was the ability to construct ad hoc a variety of robotic applications while not being constrained by functionality or/and specific domain requirements. It was necessary to design a platform with an application programming interface which allows the design of simulation experiments in a similar manner to the way the IT development community have been using object oriented languages such as C++ or Java. Thus the solution was a simulation framework for the rapid design and implementation/deployment of various applications which have to be built according to the API framework to take advantage of already existing mechanisms which otherwise would have to be developed from scratch.

2. FRAMEWORK MODEL

The simulation framework can be represented as a non-linear discrete system as depicted in Figure 4.

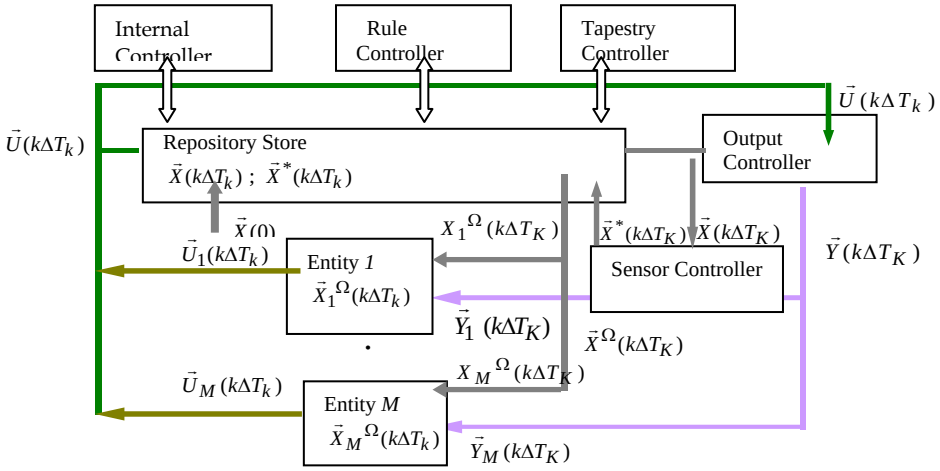


Figure 4. Simulation Framework Conceptual Model

Usually some element of the experimentation scenario becomes the major focus, while other elements constitute only the stimulus for entities of interest to respond. This stimulus is usually modelled as an environment which plays a role of virtual reality for embodied agents like humans and robots, and a natural environment for synthetic agents like software agents. In order to sense the environment, a sensor mechanism is necessary to feed the entity or group of entities with information that represents the perceived state of the environment. Sensors provide a way of transforming the "complete and true" state of the system into a perceived state for the entity. Entities can be physical or artificial, distributed across the network or local, and they can be stationary or mobile. They may be internal or external to the framework. An external entity can dynamically register with the framework and become a part of the simulation experiment as in a typical client server paradigm. Entities can interact with each other and the environment in real time or simulated time. Therefore entities are conceptual holders of their own state and define the agent. By agent we mean humans, computer programs, and robotic devices. To allow for definition of relationships between entities the rule module concept has to be introduced. The simulation framework represents a typical client server model where clients maintain their private state, and the server is a repository of shared states for all internal and external entities as well as the environment.

The Repository Store represents the true state of the system and has been introduced to maintain and preserve a consistent state of all entities taking part in the simulation. It is responsible for holding

the compound state of the system independent from the private and perceived states of individual entities. In order to maintain this state it is necessary for the simulation framework to have dead time where access by entities to their sensory information will be blocked in order to preserve a consistent view of the environment. In order to achieve this a simulation framework can be in one of two exclusive phases: the access phase where entities can enquire about the state of the environment and the deny phase where access is denied in order for a simulator to transition into a new state. Reflection is the simulation framework's method of ensuring that the private repositories of agents are synchronised with the global state repository. It guarantees that any agent accessing information from its own local repository will have an accurate state that is in sync with the simulation cycle. However, before the simulation framework can update the states, it must have a list of registered states that it should monitor. Private repositories can register states with the global state repository to receive synchronisation updates. Once registration is completed, the state repository identifies all changed states and propagates the changes to the private repositories at the end of every simulation cycle. Agent registrations are taken at the access phase while reflection happens as the last stage of the deny phase (see details below). The state vector $\vec{X}(k\Delta T_k)$ is controlled by the non-linear state transition function $\vec{\vec{f}} = \vec{f}_{TM} \circ \vec{f}_{RMM} \circ \vec{f}_R(\vec{X}(k-1)\Delta T_K, \vec{U}(k\Delta T_K - \tau), k\Delta T_K)$ where $\vec{f}_{RMM}(\vec{X}(k-1)\Delta T_K, \vec{U}(k\Delta T_K - \tau), k\Delta T_K)$ - rule manager state transition function, $\vec{f}_{TM}(\vec{X}(k-1)\Delta T_K, \vec{U}(k\Delta T_K - \tau), k\Delta T_K)$ - tapestry state transition function, $\vec{f}_R(\vec{X}(k-1)\Delta T_K, \vec{U}(k\Delta T_K - \tau), k\Delta T_K)$ - internal reasoner state transition function, $\vec{U}(k\Delta T_k) = :_{i=1}^{M} U_i(k\Delta T_k)$ concatenation of input vectors from entities, $\vec{X}_{RMM}(k\Delta T) \cup \vec{X}_{TM}(k\Delta T) \cup \vec{X}_R(k\Delta T) \subseteq \vec{X}(k\Delta T_k)$ and the residual is defined as follows: $\vec{R}(k\Delta T_k) = \vec{X}(k\Delta T_k) / (\vec{X}_{RMM}(k\Delta T) \cup \vec{X}_{TM}(k\Delta T) \cup \vec{X}_R(k\Delta T))$

The Rule Controller manipulates the $\vec{X}_{RMM}(k\Delta T_k) = \vec{f}_{RMM}(\vec{X}_{RMM}(k-1)\Delta T_k, \vec{U}(k\Delta T_K), k\Delta T_K)$ states and is responsible for modelling of non-linear interactions between entities in order to modify or overwrite the entities' own states. The rule modules represent the laws of the society that take precedence over the laws of individuals which may in some circumstances contradict each other. In order to reason about the appropriate laws and the order of their application an arbiter is necessary which in our case is called the rule module manager.

The Tapestry Controller manipulates the $\vec{X}_{TM}(k\Delta T_k) = \vec{f}_{TM}(\vec{X}_{TM}(k-1)\Delta T_k, \vec{U}(k\Delta T_K), k\Delta T_K)$ states and is responsible for creating, destroying, and manipulating both entities and rule modules.

The Internal Controller manipulates the $\vec{X}_R(k\Delta T_k) = \vec{f}_R(\vec{X}_R(k-1)\Delta T_k, \vec{U}(k\Delta T_K), k\Delta T_K)$ states and allows for direct manipulation of entities from within the framework on the contrary to the external agents who can indirectly manipulate the state of entities.

The Sensor Controller – converts the "true" state $\vec{X}(k\Delta T_k) = \vec{f}(\vec{X}((k-1)\Delta T_k), \vec{U}(k\Delta T_k), k\Delta T_k)$ into the perceived state $\vec{X}^*(k\Delta T_k) = \vec{S}_{\vec{X}}(\vec{X}(k\Delta T_k))$ of the simulator. The true state $\vec{X}(k\Delta T_k)$ of the system is filtered out by the sensor controller and the perceived state $\vec{X}^{\Omega}(k\Delta T_K) = \vec{X}_O(k\Delta T_K) \cup X^*(k\Delta T_K)$ is stored and maintained by the external agents where $\vec{X}_O(k\Delta T_K) \subseteq X(k\Delta T_K)$ the observable state of the system, $\vec{X}^*(k\Delta T_k) = \vec{S}_{\vec{X}}(\vec{X}(k\Delta T_k))$ perceived subset of the system state, $\vec{X}^{\Omega}(k\Delta T_K) = \vec{X}_O(k\Delta T_K) \cup X^*(k\Delta T_K)$ state of the system available for all external reasoners where: $\vec{X}_O(k\Delta T_K) \subseteq \vec{X}(k\Delta T_k)$ state available for all external reasoners, $\vec{X}^*(k\Delta T_K) = \vec{S}_{\vec{X}}(\vec{X}(k\Delta T_k))$ state seen by all sensors, $X_1^{\Omega}(k\Delta T_K) \ldots\ldots X_M^{\Omega}(k\Delta T_K) \subseteq \vec{X}^*(k\Delta T_k)$ state of the system available for the external reasoners, $\forall_{i=1,M} \vec{X}_i^{\Omega}(k\Delta T_K) \subseteq \vec{X}^{\Omega}(k\Delta T_K)$ where $\vec{X}_i^{\Omega}(k\Delta T_K)$ state of the system available for an i entity

The Output Controller – coverts the state $\vec{X}(k\Delta T_k)$ into the output defined as $\vec{Y}(k\Delta T_k) = \vec{h}(\vec{X}(k\Delta T_k), \vec{U}(k\Delta T_k), k\Delta T_k)$

The Entity 1 to M *(M = number of entities)* – stores perceived state and generates the input vector $\vec{U}_i(k\Delta T_k)$ in order to propose change in the state of the simulator. These inputs can be postponed or accepted by the simulator. It is achieved by introducing two phased scheduling system. A simulator can be in the access or deny phase. While in the access phase it will accept requests for change of state or access to data. Whilst in the deny phase requests are delayed until change of state is completed. During the deny phase all submitted requests for change of state are processed including internal, rule, tapestry, output and sensory controllers. This process can be represented as depicted in Figure 5.

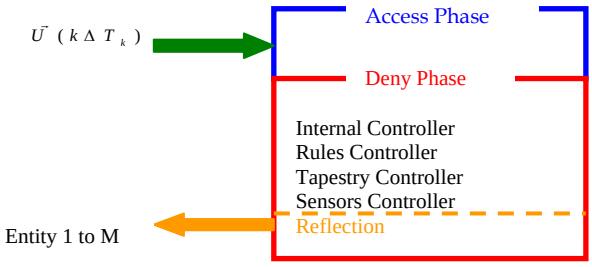
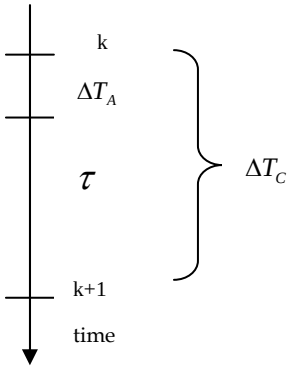


Figure 5. Framework Access/Deny Phase

One simulation circle ΔT_c is the amount of time it takes to move simulator to a new state as follows $\Delta T_C = \Delta T_A + \tau$, ΔT_A - allocated time for submission of requests to change the state of a simulator represented by the $\vec{U}$ vector, τ - variable time that defines the necessary time to calculate and update the state of the simulator which is a function of dynamically calculated state transition functions $\tau(\vec{f}_{TM}, \vec{f}_{RMM}, \vec{f}_R)$ plus time necessary to update private repositories of external entities. The simulation time step has no impact on τ under the condition that the numerical accuracy of simulation is excluded from our considerations. The τ = max ($\Delta T_{S_{Min}}, \Delta T_{SR}$) where $\Delta T_{S_{Min}}$ - min allocated time to complete state change of the simulator, ΔT_{SR} - actual time to complete state change of the simulator. As it is not difficult to predict that the processing of state transition functions is the most critical and computer intensive process during the deny stage of the simulation cycle. The complexities of some models are so overwhelming that sometimes it is not possible to process them in an acceptable time frame due to limitations of computing power. Due to the numerical characteristic of some models or numerical accuracy requirements some numerical algorithms may need to be changed on the "fly" between the simulation cycles. Another aspect is the complexity of model structure and order of processing models in order to guarantee data integrity. Processing order is often dynamic and driven by the nature of the simulated processes. Response time and the ability to simulate complex mathematical models is sometimes a critical factor. Parallel processing of some models can improve the overall performance of the simulation process under the condition that there is no interdependency between data generated by different models. Distribution of models between many computers can improve performance significantly under the assumption that network delays are negligible and the dependency of data can be mitigated. The HLA protocol has been proposed as a solution to interoperability issues



between different models with no regard to optimisation of computer resources across the network and numerical accuracy requirements (Kuhl *et al.* 1999). Our proposal is an attempt to address this issue by introducing the Distributed Rule Module Protocol. Now we describe some ideas behind such a protocol.

3. DISTRIBUTED RULE MODULE PROTOCOL

Processing mathematical models locally is as simple as calling local solvers in the order defined by the rule module manager during the execution of a simulation cycle (see Figure 6). Sequential order of execution is reinforced by the blocking method call. Although models are executed sequentially one after another there is the possibility of changing the processing order by

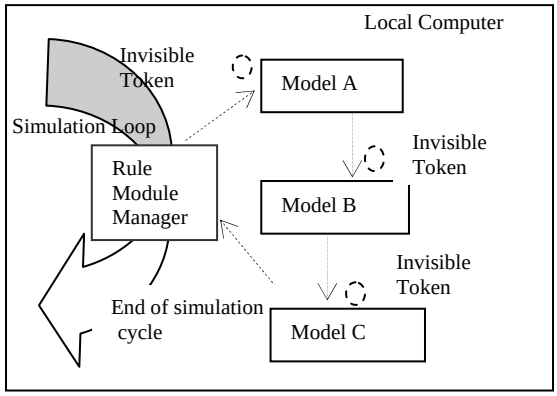


Figure 6. Processing Models on Local Computer

reordering or introducing a token that can "travel" between them. If we physically place models on separate computers then we need to inform the distributed frameworks when to start processing their models and in what order. A data structure representing a tree of relationships between rule modules will constitute the content of that token that will travel between computers and activate calculation of an appropriate model at every stage of a simulation cycle. Details of the protocol can be found in Jagiello *et al.* 2006.

A distributed simulator is a network of fully functional simulators hosting their own processing components. Each individual simulator will

become a part of the whole distributed repository store. Only one simulator at the time will drive the simulation cycle across the network. The simulator responsible for driving a simulation cycle can be selected dynamically from the set of participating simulators. The simulator responsible for driving the simulation loop will generate a set of tokens that are sent to selected simulators across the network. These tokens represent independent branches of the tree or data sets that can be processed in parallel by distributed computers. The distributed computers will process their models according to a defined pattern, and a returning token will be sent back to resume the next simulation cycle. This pattern represents the tree structure of distributed rule modules and numerical requirements and process by which numerical results are acquired. New tokens will be generated dynamically when processing parameters have to be changed. The tokens will contain only the differences in the configuration of processing parameters. The order and hierarchy of models to be processed will be determined by a dynamically configurable table defined inside the token. A token or set of tokens has to be returned back to the rule module manager responsible for driving the simulation loop in order to complete the simulation cycle as depicted in Figure 7. The framework infrastructure and its API was implemented in Java and published in Jagiello *et al.* 2006.

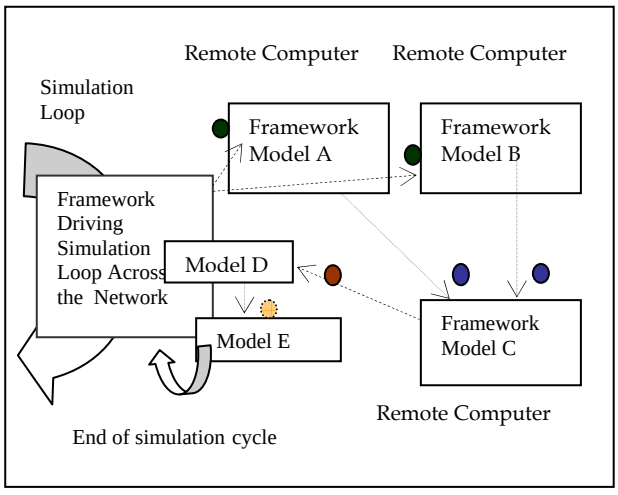


Figure 7. Processing Models on Distributed Computers

4. APPLICATIONS

The framework infrastructure has been mainly used as a prototyping tool for testing robotic applications. Although, recently it has been deployed for a distributed gaming application called Go*Team played across the network. This paper will present three different types of applications to show a utility of the framework. Namely, they are:

1. The Asset Protection scenario
2. The Mind Strom application
3. The Go*Team game

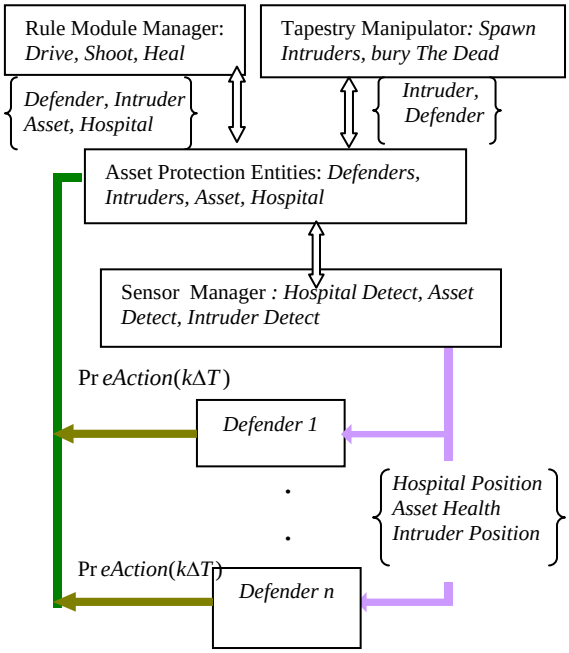


Figure 8. Asset Protection Implementation Model

The Asset Protection scenario simulates the Mobile Detection Assessment Response System (MDARS). The MDARS is a robotic system for physical security and automated inventory of high-value or critical assets (Carroll *et al.* 2006). The scenario for the asset protection application is very simple. A group of defenders are protecting an asset by patrolling a confined area. Intruders appear randomly and attempt to destroy the asset. The defenders in order to destroy intruders have to inform each other about the encroachment and collaborate to surround and destroy the intruder. The defenders have a limited level of health and have to visit the "hospital" from time to time in order to repair damage inflicted by intruders. The defenders based on the current circumstances have to calculate their own goals and make a decision about when to engage in patrolling, pursuing, fighting or damage repair activities. The asset protection implementation model can be represented as depicted in Figure 8. The asset protection scenario is comprised of the following entities and their objectives (Figure 9).

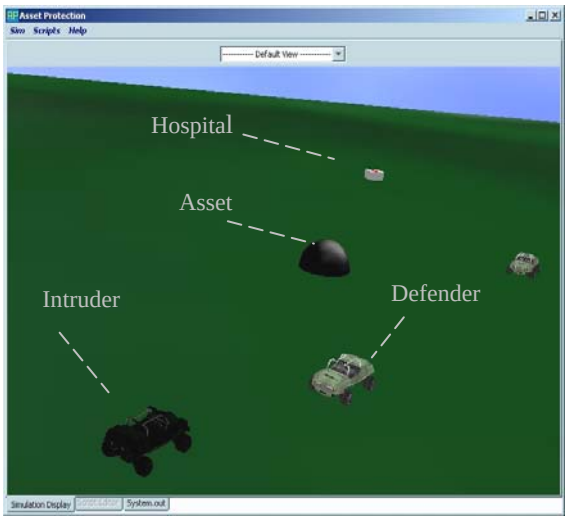


Figure 9. Asset Protection Simulator

Assets - represent the critical infrastructure. Assets are stationary and are assigned a health value.

When their health value reaches zero, the asset is destroyed. In the simulation, assets are represented as black domes.

Hospital - were added to heal and replenish defenders. For a defender to be healed, they must be within range of the hospital. Hospitals are stationary and cannot be destroyed. Hospitals are represented in the simulator as a white cube with a red cross on the roof.

Intruders - are entities whose sole objective is to destroy the asset. Intruders are mobile and can 'shoot' (reduce the health of) any entity within their range. Intruders must move to the asset to destroy it. Intruders can destroy defenders, but their primary role is to destroy the asset. Intruders have a health, which when is zero, results in the intruders being destroyed. There are no limits to the number of intruders and intruder will appear randomly. Intruders are represented as black jeeps.

Defenders - Defenders are responsible for protecting the asset. This means that the defenders will have two tasks to perform; to patrol around the asset to ensure that no intruders are present, and to repel and destroy any intruders that are attacking the asset. Including their personal goal of survival, the following is a list of goals and objectives for the defenders.

PATROL – Patrol the asset to discover intruders.

ATTACK – Close in and destroy any intruders.

DEFEND – Return to the asset and protect it from attacking intruders.

HEAL – When health is low, move to the hospital to be healed.

Defenders are mobile, with a health and a range for its attack. When its health is zero, the defender is destroyed. There are only a fixed number of defenders in the simulation. Defenders follow different patrol paths to increase the coverage and protection of the asset. Defenders are represented as green jeeps in the simulator.

In the Mind Storm application a physical robot and its model is controlled simultaneously by one robotic application. The rigid body model has been used to model a robot by the simulation framework in order to respond in real time to control command from the application (Baraff 1989). See picture of real and simulated robot (Figure 10).

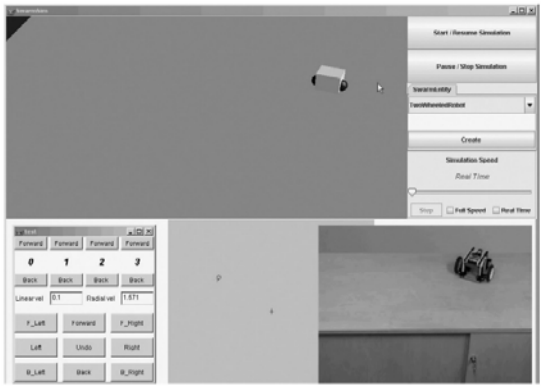


Figure 10. Mind Storm Simulator

The Go*Team game was implemented as a multiplayer network computer game using our framework (Jagiello *et al.* 2007). Physically dispersed teams with individual players will play against each other. Teams can form alliances to simulate coalition forces. A game can be played by many teams on many boards with a limited number of allocated resources (stones). In order to introduce the "fog of war" each player can see

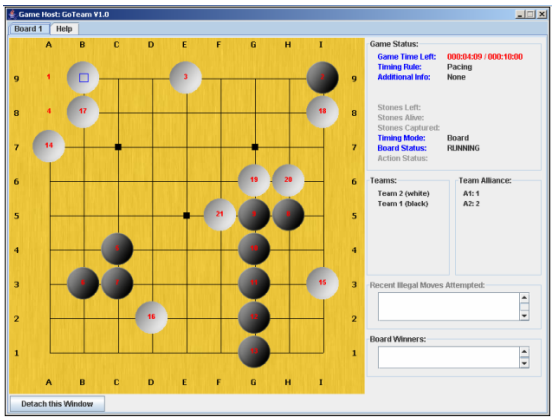


Figure 11. The global Go*Team Situation Awareness (viewed via the server).

only the partial state of the game while only the game host can enjoy the full view of the game (Figure 11, 12, 13).

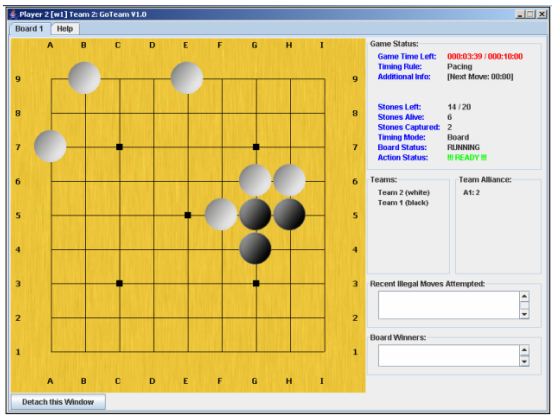


Figure 12. The local view of one of the two black players, who can see only their own stones plus those stones of white that are closer to their own stones than those of any other player on the black team.

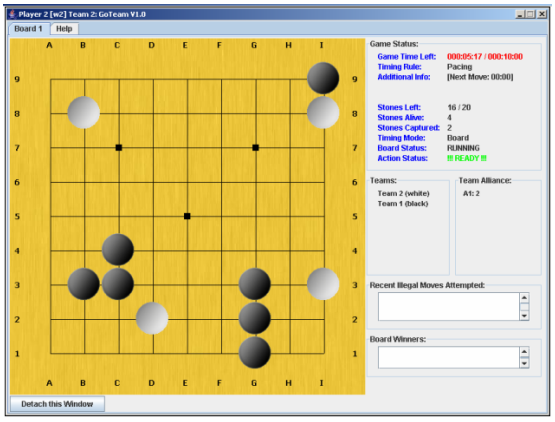


Figure 13. The local view of one of the other black players

5. CONCLUSIONS

The developed infrastructure and its API's seem to be very effective tools especially for prototyping purposes. Interfaces provided by the framework and supporting infrastructure allows for an effective way of developing a variety of simulation models from different domains.

6. AKNOWLEDGMENTS

The authors are grateful to Nicholas Tay for his contribution to the development of concepts and ideas and his participation in the implementation phase of the Simulation Framework.

7. REFERENCES

Yeo, S., J. Kim, S.H. Lee, F.C. Park, W. Park, J. Kim, C. Park, I. Yeo (2004), A modular object-oriented framework for hierarchical multi-resolution robot simulation, *Robotica, Cambridge University Press New York, NY, USA, Volume 22 , Issue 2, Pages: 141 - 154*

Xavier, P.G., E. Gottlieb, M. McDonald, F.J. Oppel, J.B. Rigdon (2002), The Umbra Simulation Framework as Applied to Building a Federates**,** *Proceedings of the 2002 Winter Simulation Conference*.

Yalcin, A., R.K. Namballa (2005), An object-oriented simulation framework for real-time control of automated flexible manufacturing systems, *Computers and Industrial Engineering, Pergamon Press, Inc. Tarrytown, NY, USA, Volume 48 , Issue 1, Pages: 111 – 127*

Gottlieb, E., R. Harrigan, M. McDonald, F. Oppel, P. Xavier (2001), The Umbra Simulation Framework, *Intelligent Systems and Robotics Center Sandia National Laboratories, SAND2001-1533 Unlimited Release*.

Jagiello, J., N. Tay, M. Eronen (2006), A Robotic Middleware, *DSTO-TR-1824.*

Kuhl, F., R. Weatherly, J. Dahmann (1999), Creating Computer Simulation Systems: An Introduction to the High Level Architecture, *Prentice-Hall International, ISBN 0-13-022511-8.*

Jagiello, J., N. Tay, M. Eronen (2006), A Simulation Framework, *DSTO Report.*

Carroll, D., H.R. Everett, G. Gilbreath, K. Mullens (2006),Extending Mobile Security Robots to Force Protection Missions, *Space and Naval Warfare Systems Center, San Diego, http://www.spawar.navy.mil/robots*

Baraff, D. (1989), Analytical methods for dynamic simulation of non-penetrating rigid bodies" *Computer Graphics* (*Proc. SIGGRAPH*), volume 23, pages 223–232.

Jagiello, J., M. Eronen (2007), Go*Team, an instance of the simulation framework, *MODSIM Conference, Christchurch, NZ.*