Hazard: A Framework Towards Connecting Artificial Intelligence and Robotics

Peter J. Andersson
Department of Computer and Information Science, Linköping university
petan@ida.liu.se

Abstract

The gaming industry has started to look for solutions in the Artificial intelligence (AI) research community and work has begun with common standards for integration. At the same time, few robotic systems in development use already developed AI frameworks and technologies. In this article, we present the development and evaluation of the Hazard framework that has been used to rapidly create simulations for development of cognitive systems. Implementations include for example a dialogue system that transparently can connect to either an Unmanned Aerial Vehicle (UAV) or a simulated counterpart. Hazard is found suitable for developing simulations supporting high-level AI development and we identify and propose a solution to the factors that make the framework unsuitable for lower level robotic specific tasks such as event/chronicle recognition.

1 Introduction

When developing or testing techniques for artificial intelligence, it is common to use a simulation. The simulation should as completely as possible simulate the environment that the AI will encounter when deployed live. Sometimes, it is the only environment in which the AI will be deployed (as is the case with computer games). As there exist many more types of environments than AI techniques, there is a need to reuse existing implementations of AI techniques with new environments. AI frameworks often come bundled with an environment to test the AI technique against and the environments are often not as easy to modify as a developer would want, nor is it easy to evaluate the framework in new environments without developing time-consuming middleware. In the case of robotics, there is an extended development period to develop low-level control programs. The AI then developed is often tailored to the low-level control programs and it is hard, if at all possible to reuse.

The Player-Stage project [Gerkey *et al.*, 2003] develops low-level drivers for robotic architectures and gives the developer an interface that can either be used together with the stand-alone simulator to evaluate the configuration or to control the actual robotic hardware. This kind of interface encourages focus on the high-level control of the robot, but since there are no wrappers to high-level AI frameworks, it does not encourage reuse of existing AI techniques. By developing a high-level interface between Player-Stage and AI frameworks, we will also allow AI researchers to take advantage of the Player-Stage project.

The Robocup initiative [Kitano *et al.*, 1997] uses both actual robotic hardware and simulation in competition. Yet, there exists no common interface for using simulation league AIs with robotic league robots. This can mean that the simulation interface is unintuitive for actual robotics, or that AIs developed with the simulation are not usable with actual robots. In either case it is a problem worth investigating.

The WITAS Unmanned Aerial Vehicle project [Doherty *et al.*, 2000] uses several simulators in their research, both for hardware-in-the-loop simulation of the helicopter hardware and for development of dialogue interaction with an actual UAV. A middleware translating actions and events from WITAS protocol to other protocols would allow experimentation with for example SOAR [Laird *et al.*, 1987] as a high-level decision system. It would also allow the application of developed agent architecture to other environments and problems.

By creating a middleware framework that can mediate between low-level drivers and high-level decision system, we hope to be able to alleviate the problems for AI researchers presented above and inspire researchers in both artificial intelligence and robotics to reuse existing implementations. Robotic researchers can reuse AI techniques that exist and AI researchers can test their AI implementation in off-the-shelf simulators before building an expensive robotic system. It would also allow separate research groups to work with low-level control and high-level decision issues.

As a step towards proposing an interface and middleware for connecting AI and robotics, we have designed and implemented a framework for developing agents and environments where we focus on the distinction between agent and environment. The design iterations and various implementations with the framework have allowed us to gain valuable experience in designing both interface and framework. The work is presented here together with an evaluation of the strengths, weaknesses and limitations.

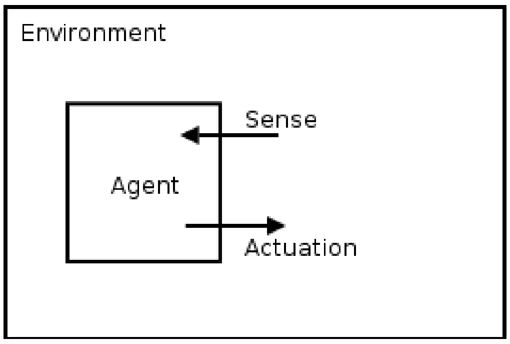

Figure 1: Agents and Environments.

2 Interface

The framework is designed around action theory and based on the agent-environment concept as found in [Russel and Norvig, 1995], see figure 1. The framework is thus divided into three parts; interface layer, agent module and environment module. The agent uses actuators to communicate its intention to the environment, the environment delivers the sensor impressions back to the agent. In this interface the actuation is handled by actions. Each action and sensor has an owning agent to which all data is passed. The interface encourages multi-threaded applications where the agent decision loop is in one thread and the environment is in another. This design has been helpful when building continuous, asynchronous environments such as simulations of robotic vehicles.

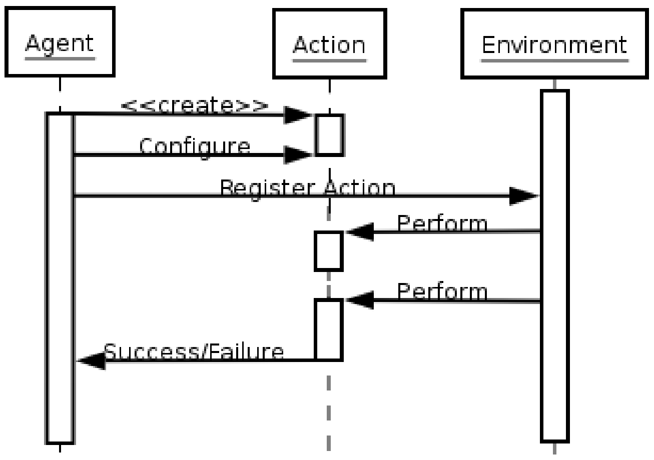

Figure 2: How actions are used.

Actions

When an agent has decided on using a certain action, it configures the action and notifies the environment that it wants to "register" the action. If the action is applicable in the current state of the environment, it is accepted and execution starts. The action contains its own executable code and is thus defined in the environment module. Actions are bundled with their own event recognition and execution monitoring. At each execution interval, they update the state of their owner in the environment. If applicable, they can send "checkpoint" reports back to the owner. When the action has reached its end criteria, it sends a "success" message. If it fails during execution, it sends a "fail" message. If it is an action to which success/fail has no meaning (for actions without an explicit goal), the action can send a "stop" message. All these messages are implemented as callback methods in the agent interface. The implementation is visualized in the sequence diagram in figure 2. Since actions are implemented in the environment, the agent must have knowledge of the environment to be able to use the actions. Actions are considered to be executed in continuous time, can be concurrent and are either discrete or durative regarding the end criteria.

Sensors

Sensors are permanent non-invasive actions that can be registered in the environment. Sensors contain their own executable code and should not modify the environment. A sensor analyzes the environment at a preset time-interval and stores the extracted information. When it has new data, the sensor can notify the owning agent via a callback routine. The agent can then fetch the new sensor data from the sensor via the sensor interface. What kind of data that is passed between sensor and agent is not defined by the interface but implemented on a case by case basis.

3 Framework

Together with the interface a framework has been designed. The framework has been named Hazard and helps developers to rapidly create new environments and agents. It can also be used to design wrappers for other environments or agents, thus allowing other implementations using Hazard to be quickly modified to use that agent implementation or environment. The design has been developed iteratively with several simulations as reference implementations. The experience from developing each simulator has given a simpler, more robust and better designed framework ready for the next project.

The goal of the framework has been both to allow development of new environments and agents, and to use it as a middleware between already existing frameworks. The design focus has been on forcing developers to make important design decisions at an early stage of the project and to leave simpler issues in the hands of the framework.

3.1 Environments

An overview of the environment module can be found in figure 3 and described in more detail below.

Environment

An environment in Hazard has the task of keeping track of Maps and execution of actions and sensors. The environment can contain several maps, dividing computation, and gives a more scalable structure.

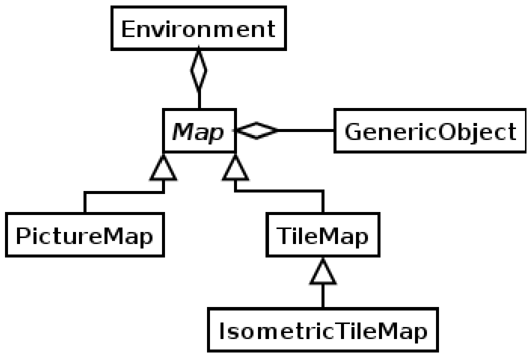

Figure 3: General structure of the environment module.

Map
: A map in Hazard can contain a number of objects and agents, where agents are a subset of the objects. The framework has a number of default types of maps that can be used by developers. Each of these maps is data-driven and can be inherited or used as is.

GenericObject
: GenericObject is the template implementation of all objects in the environment. For most objects, the GenericObject class can be used as is. For specific needs, the class can be inherited to create unique objects specific to a certain environment.

The environment module also implements a basic graphics engine that can handle two dimensional graphics.

3.2 Agents

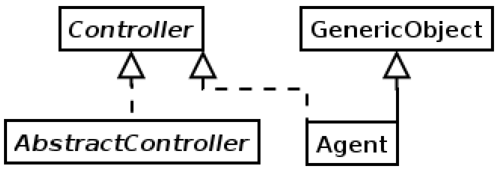

Figure 4: General structure of the agent module.

Agents are controlled by either AI or a user. To generalize, both cases implement the interface “Controller”. A controller can control one or more objects in the environment, effectively turning them into agents. Controllers implement methods for registering/unregistering actions and sensors in the environment. They implement the callbacks for actions and sensors, but have otherwise no limitations. A GenericObject that implements the Controller interface is called an agent. The Agent module also contains classes and methods for class reuse, action and sensor design and abstract classes for different types of controllers.

4 Iterations of Design and Implementation

The strength of a design proposal may be measured in term of its development history. Designs that have undergone several iterations under different conditions are much more mature than designs without practical grounding. The Hazard framework was derived from the Haphazard Role-Playing Game project. The derivation was redesigned and reimplemented to form the first version of the framework. The Haphazard project was redesigned to use the new framework and experience from that work was used to implement the traffic simulator. The framework was redesigned again, and was used to implement a development support simulator for a dialogue system, leading to the present Hazard framework. The Haphazard Online Role-Playing Game and the Simulator for Support of Dialogue System Development are currently using the latest version of the Hazard framework while the Subsumption Simulator is using an older version.

4.1 Haphazard - An Online Role-Playing Game

The Haphazard Role-Playing Game [Andersson and Beskid, 2003] started as an open-source project for creating a simple online role-playing game with focus on AI implementations. It was from this game project that the first version of Hazard was extracted. The framework was then redesigned and Haphazard was reimplemented to use the new framework. Haphazard has the most complex environment of all implementations up to date, but only rudimentary AI. The Haphazard project was the most prominent project when designing the framework for environments.

Environment

The environment in Haphazard is a grid-based model using tiles for visualisation. Objects in the world are either static or dynamic. Static objects include houses, trees and other scenery. Dynamic objects are the objects that can be manipulated by the player. Haphazard includes a novel inventory system, environment specific enhancements to the Hazard framework that include minor agent interaction and a dynamic environment system. Players can roam between different maps which allows for a distributed environment. The environment is visualized by an isometric tile-based graphics engine. The environment is data-driven. Both environment and graphics can be changed at run-time.

Agents

The main focus is on the agent that allows the user to interact with the environment. It implements a graphical user interface which allows the user to have direct control of the agent. The user interface implements movement control, skill management and inventory management. The agent does not have complete vision of the world, but has a small field of vision that is represented by grayed out tiles in the user interface. The agent does not make use of sensors and has complete access to the environment. The available actions for each agent are Move, Pack, Drop, Equip, Eat, Fight, Say, Pull, Push and Stop. Sensors used in the AI-implementation are Closest Object, Global Position and Closest Item. Some small AI agents have been implemented, for example roving barbarians that hunt the player on sight. All agents are data-driven and can be added or removed from the environment at run-time.

4.2 Simulator for Evaluating the Subsumption Architecture

An implementation of the subsumption architecture [Brooks, 1985] was introduced to the agent part of the framework as part of the work towards evaluation the subsumption architecture for use in obstacle avoidance systems for cars [Woltjer and McGee, 2004]. This implementation allowed us to evaluate the agent part of the framework and enhance it. The subsumption architecture was integrated with an editor which allowed the user to change the subsumption network during simulation runs to experiment with new behaviours.

Environment

The subsumption agent was used in two environments. The architecture was developed and tested in the Haphazard environment, but mainly used in a traffic simulator. The traffic simulator used a straight road without any static obstacles. It used an approaching car as a dynamic obstacle and the user's input was merged with the subsumption architecture to get user driven obstacle avoidance.

Agents

The user controlled a car with acceleration and turning and the agent merged the user's input with sensor data in a subsumption network before it was actuated. The sensor data was a vector to the closest object within a preset distance, possible actions were accelerate, break, steer left, steer right. Another agent was also implemented, a car travelling with constant speed in the opposite direction.

4.3 Simulator for Dialogue System Development Support

Within the WITAS Unmanned Aerial Vehicle (UAV) project [Doherty *et al.*, 2000], the Hazard framework was used in implementing a simulator that supports development of a dialogue system for command and control of one or more UAVs. The new simulator replaced an old simulator that had been in development by several persons totalling approximately one man year. The implementation achieved about the same functionality (more in visualization, less in camera control) but better integration by one person in three weeks.

Environment

The environment consists of a three-dimensional map containing roads and buildings. The roads consist of several segments, intersections and dead ends. All roads have a name. Buildings are visualized as polygons and have a height. All buildings also have a name, color, material and one or more windows which can be observed by sensors. The environment is fully data-driven and new environments using these types of objects can easily be constructed.

Agents

There exist three types of agents:

WITAS UAV
: The WITAS UAV agent mainly consists of a socket interface which can receive commands from the dialog system, execute these and report their progress. The agent has a camera sensor and can detect cars and buildings that come within the camera's field of vision. A WITAS UAV can take off, land, ascend, descend, hover, fly to positions/buildings/heights, follow cars and follow roads to accomplish it's mission. A camera sensor detects buildings and cars within a preset distance. It is an interactive agent that can build plans and follow the commands of a user.

COMETS UAV
: The COMETS UAV agent was implemented as an experiment together with the COMETS project [Ollero *et al.*, 2004]. It implements an interface to the COMETS Multi-Level Executive [Gancet *et al.*, 2005] and uses the same set of actions and sensors as the WITAS UAV agent. Since the WITAS project and the COMETS project deal with approximately the same environment and agents, the interface could reuse most of the WITAS UAV implementation. The integration of the COMETS UAV into the simulation was made in about 12 man hours.

Car
: Cars drive along roads with a set speed. They can either drive around randomly or follow a preset route, but can't detect or interact with other Car agents. Cars are completely autonomous agents. They use only one action, "drive", and no sensors.

All agents are data-driven and can be added to the environment before the start up. They cannot be added during run-time (at present). The simulation is transparent and can be fully or partly substituted by a connection to a real world UAV. The visualization is 2D, but current work is extending both camera view and world view to three dimensions.

5 Evaluation

The evaluation of the framework is based on our own experience in developing simulations. It is focused on development and how suitable the framework is for use as middleware and as a framework for development of high-level and low-level AI for robotic applications.

5.1 Strengths

Clear design guidelines
: The framework gives clear design guidelines due to its structure. The flow for developing is generally:

- What type of environment (discrete-continuous) shall I use?
- What important objects exist in the environment?
- What actions are available?
- What information do agents need that is not available through success/fail information for actions? (i.e. what sensors are needed?)
- What agents do we need?

Rapid development
: The different implementations have shown that the framework rapidly gives a working system. The only comparison that has been done on development time is with the replacement of the Simulator for Dialogue System Development Support. The implementation using

the Hazard framework cut the development time radically.

Scalability
: The framework is very scalable in the form of developing new agents or objects for an environment. It has not been tested how scalable it is in regard to the number of existing objects/agents or actions/sensors in an environment.

5.2 Weaknesses

Agents are tightly linked to the environment
: Since the framework was extracted from an integrated system, the agents are tightly linked to the environment and can have complete knowledge of the world without using sensors. This is a big drawback as it allows developers to "cheat" and use information that shouldn't be available to the AI implementation. The agent module should be completely separate from the environment.

Agent to agent interaction
: Since the design does not distinguish between success/fail messages for an action and sensor data, it is hard to develop agent to agent interactions. A solution for this problem could be to remove the success/fail notion from the environment side of the action and let the agent side sensor interpreter decide when an action has been successful or failed. This solution would also allow research into event/chronicle recognition.

New developers
: The system is well documented, but lacks examples and tutorials. This makes it harder for new developers to use the framework.

5.3 Limitations

Mid-level functionality
: Since the framework only supports high-level actions and is confusing on the part of sensor data, mid-level functionality is not intuitively supported. By mid-level functionality is meant functionality such as event/chronicle recognition, symbol grounding and similar robotic tasks. This is a disadvantage if the system is used for developing AI techniques for robotic systems since a developer can easily "cheat" with constructed events instead of having to identify them from sensor data.

Pre-defined set of actions
: Since the actions are pre-defined in the environment, both with regard to execution and evaluation of the execution (success/fail), an agent cannot learn new actions or interpret the result in new ways. Also, since the actions can be of different level of abstraction, it is hard to combine actions concurrently.

6 Related Work

Currently, the International Game Developers Association (IGDA) is pushing towards AI standard interfaces for computer games and is in the process of publishing the first drafts of a proposed standard. These standards are geared towards enabling game AI developers to reuse existing AI middleware and to concentrate on higher level AI tasks. IGDA is working on interfaces for common AI tasks and has currently working groups on interface standards for world interfacing, path planning, steering, finite state machines, rule-based systems and goal-oriented action planning. The work presented here is closest to the work on world interfacing, but since the draft was not available at the time of writing, it was impossible to compare.

The Testbed for Integrating and Evaluating Learning Techniques (TIELT) [Aha and Molineaux, 2004] is a free software tool that can be used to integrate AI systems with (e.g., real-time) gaming simulators, and to evaluate how well those systems learn on selected simulation tasks. TIELT is used as a configurable middleware between gaming simulators and learning systems and can probably be used as a middleware between general environments and agent frameworks with some modification. The middleware philosophy of TIELT differs from our implementation, TIELT sits as a black box between environment and agent while Hazard is only meant as a transparent interface without any real functionality, except if the framework is used on either side. The black box functionality would hide event/chronical recognition, symbol grounding, etc... in a robotic system. This means that special care has to be taken if the system is to be used with actual robotics.

The System for Parallel Agent Discrete Event Simulator (SPADES) [Riley and Riley, 2003] is a simulation framework with approximately the same concept as the Hazard framework with regards to the separation between agents and environments. It focuses on repeatability of simulations and uses a concept of Software-in-the-Loop. Software-in-the-Loop in SPADES measures the actual time an agent executes. Using this technology, it can give a large number of agents the same time-share by slowing down the simulation without sacrificing simulation detail and repeatability. SPADES is a discrete event simulator and uses a sense-think-act loop for the agents which limits its deliberation time to the time between receiving events until it has decided what to do. This limitation is minimized by allowing agents to tell the simulation that it wants to receive a *time notification* which works as a sense event. Our framework on the other hand sacrifices repeatability and agent timesharing to obtain a continuous, asynchronous time model which is more inline with robotic architectures than a discrete model. The agent developer can then decide to translate into a discrete time model or keep the continuous one.

There is also research on modules and frameworks that can be used in conjunction with a set of interfaces for cognitive systems, in particular DyKnow [Heinz and Doherty, 2004], a framework to facilitate event and chronicle recognition in a robotic architecture.

7 Conclusions and Future Work

The iterative development of framework and interfaces has enabled us to gain valuable experience in designing interfaces that are adequate for both robotic systems and simulated environments without sacrificing detail or ease of use. Our goal is to develop an architecture for connecting AI and robotics with the following characteristics:

- Rapid environment/agent development
- Connects agents and environments
- Able to reuse both existing agents and environments
- Capable of acting as middleware between existing frameworks
- Usable in research of both simulations and actual robotic systems

Hazard is a mature system which has undergone several design iterations. It allows rapid development and reuse of agents and environments. It contains intuitive interfaces between agent and environment and can be used as middleware between existing frameworks. But Hazard has been found unsuitable for development of AI or simulations for actual robotic systems due to its inherent limitation in event recognition and actuator control. To be usable in a robotic AI implementation, the interfaces need to be layered to allow both for high-level AI frameworks and middle-level event and chronical recognition on the agent side. The framework for agents and environments also need to be structured in layers to support both discrete event and continuous time simulations with action/event and actuator/sensor interfaces.

Currently, work has been started on a new generation of interfaces and framework. This work is called CAIRo (Connecting AI to Robotics) and a first implementation with the framework has already been done. The development will focus on first validating the new framework with the current implementations and then continue the validation with middleware functionality and implementations of robotic systems.

8 Acknowledgements

This research work was funded by the Knut and Alice Wallenberg Foundation, Sweden, and by the Swedish National Graduate School in Computer Science (CUGS).

References

[Aha and Molineaux, 2004] D.W. Aha and M Molineaux. Integrating learning in interactive gaming simulators. In D. Fu and J. Orkin, editors, *Challenges of Game AI: Proceedings of the AAAI'04 Workshop (Technical Report WS-04-04)*, San Jose, CA, 2004. AAAI Press.

[Andersson and Beskid, 2003] Peter J. Andersson and Lucian Cristian Beskid. The haphazard game project. http://haphazard.sf.net, 2003.

[Brooks, 1985] R. A. Brooks. A robust layered control system for a mobile robot. Memo 864, MIT AI Lab, September 1985.

[Doherty *et al.*, 2000] P. Doherty, G. Granlund, G. Krzysztof, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund. The witas unmanned aerial vehicle project. In *ECAI-00*, Berlin, Germany, 2000.

[Gancet *et al.*, 2005] Jérémi Gancet, Gautier Hattenberger, Rachid Alami, and Simon Lacroix. An approach to decision in multi-uav systems: architecture and algorithms. In *Proceedings of the ICRA-2005 Workshop on Cooperative Robotics*, 2005.

[Gerkey *et al.*, 2003] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal, June 2003.

[Heinz and Doherty, 2004] Fredrik Heinz and Patrick Doherty. Dyknow: An approach to middleware for knowledge processing. *Journal of Intelligent and Fuzzy Systems*, 15(1), 2004.

[Kitano *et al.*, 1997] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York, 5–8, 1997. ACM Press.

[Laird *et al.*, 1987] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(3):1–64, 1987.

[Ollero *et al.*, 2004] Aníbal Ollero, Günter Hommel, Jeremi Gancet, Luis-Gonzalo Gutierrez, D.X. Viegas, Per-Erik Forssén, and M.A. González. Comets: A multiple heterogeneous uav system. In *Proceedings of the 2004 IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR 2004)*, Bonn (Germany), May 2004.

[Riley and Riley, 2003] Patrick F. Riley and George F. Riley. Spades - a distributed agent simulation environment with software-in-the-loop execution. In S. Chick, P. J. Sanchez, D. Ferrin, and D.J. Morrice, editors, *Proceedings of the 2003 Winter Simulation Conference*, pages 817–825, 2003.

[Russel and Norvig, 1995] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[Woltjer and McGee, 2004] R. Woltjer and K. McGee. Subsumption architecture as a framework to address human machine function allocation. Presented at SimSafe, http://130.243.99.7/pph/pph0220/simsafe/dok/ simsafe05.pdf, 2004.