REQUIREMENTS FOR A SOFTWARE FRAMEWORK FOR MULTI-AGENT MACHINE LEARNING

CONNOR P. DOYLE
UNIVERSITY OF WISCONSIN - LA CROSSE
connor@enmas.org

VERSION 0.1.2

Date: October 9, 2011.

Contents

1. Introduction

1.1. **Purpose.**

This document outlines the functional and nonfunctional requirements for a software framework for machine learning. This document is intended to be used to support the development for such a software product. This is intended to be a "living" document that will evolve as development of the product proceeds.

In another sense, this document serves as an agreement of mutual understanding of the goals and scope of the product under development by the implementor and the project sponsor.

1.2. **Document Conventions.**

Functional requirements are given using the following format:

Index	1.2-R1
Name	
Purpose	
Input Parameters	
Action / Result	
Output Parameters	
Exceptions	
Remarks	
Cross-References	

- *Index* refers to a unique designation assigned to each individual functional requirement. Indices will be used for cross-referencing functional requirements throughout the documents associated with this project.
- *Name* is a descriptive name given to a functional requirement. This name is not required to be unique.
- *Purpose* is a short description (in a line or two) of the function. It is used to quickly understand the functionality and is also used to search the required function when all the functionalities are browsed through.
- *Input Parameters* refer to a set of parameters that the given function accepts as input. These parameters are required in order to design and implement the current function. No type information will be included for parameters in this document.
- *Action* refers to a set of tasks or activities that must be performed in order to satisfy this requirement. These tasks/activities are listed in no particular order. The design will take care of sequencing the tasks.

- *Output Parameters* refer to a set of parameters that are output/exhibit by the current function, when implemented. No type information will be included for parameters in this document.
- *Exceptions* refer to a set of conditions, each of which indicates a situation in which the function will stop. Notice that this column only lists a set of exceptions that might occur but does not suggest any action that must be taken when the exceptions occur. These actions will be included in the design document.
- *Remarks* include a set of comments that explain more about the functionality. It also describes hints to the designer and implementer that are suggested by the requirements analyst.
- *Cross-References* refer to a set of other functional requirements that are related to the current function. The related item will be identified by its *index*.

1.3. **Intended Audience.**

This document should be read by the internal development and advisement team for this project as well as the project sponsor. In the case of this project, the project sponsor and the primary adviser are the same person (Dr. Marty Allen).

1.4. **Additional Information.**

This project fulfills in part the requirements for the Master of Software Engineering degree at the University of Wisconsin - La Crosse. This project is currently being developed by Connor Doyle under the advisement of Drs. Marty Allen and Kenny Hunt, both professors in the Computer Science department at the University of Wisconsin - La Crosse.

1.5. **Contact.**

For information contact Connor Doyle [connor@enmas.org].

The official project web site at http://enmas.org will host a wiki tracking development progress, as well as relevant user-level documentation as it becomes available.

2. Overall Description

2.1. **Product Perspective.**

The goal of this project is to facilitate and conduct multi-agent systems research. EnMAS (Environment for Multi-Agent Simulations) will be an environment simulator server hosting client AI agents which will collaborate to achieve some goal, as well as an API for implementing those agents. A parallel goal is to use the environment simulator and framework as a teaching tool for students in machine learning or artificial intelligence classes to implement and evaluate existing and new techniques for planning and learning.

2.2. **High Level Product Functions.**

The product has two main functional goals:

(1) To support researchers working in the areas of Machine Learning, AI, or Robotics. The tool facilitates this work in two main ways. First, the tool will make shorter work of experiment design and coding. Second, the tool will define a common language and format for communicating experiments and results.
(2) To support educators teaching Machine Learning, AI, or Robotics concepts.

2.3. **User Classes and Characteristics.**

2.3.1. *Students and Enthusiasts.*

These users will interact with the system exclusively via the client API, which allows them to implement the behavior of an AI agent within an experiment run using a pre-existing problem scenario.

2.3.2. *Researchers and Educators.*

These are expert users who will interact with the system via the client API as described above, but also via a more detailed API which allows for the definition of new problem scenarios.

2.4. **Operating Environment.**

2.4.1. *Operating System.*

Current versions of Windows, Mac OS, and Linux (must have the Java Virtual Machine version 1.6 or above installed).

2.5. **Design and Implementation Constraints.**

- The system must employ a client-server architecture, allowing for the synchronization of disjoint modules running on several computers in a local area network.

2.6. **Assumptions and Dependencies.**

(1) It is assumed that all users possess a minimal level of programming expertise. It is assumed that the minimal level of technical expertise possessed by researchers and educators is sufficient to read technical documentation and to write code that utilizes an API.

3. External Interface Requirements

3.1. **User Interfaces.**
In addition to the API-level interfaces available to users, the system must have a graphical user interface capable of invoking the main features of the system.

3.2. **Hardware Interfaces.**

- **Display**
  A computer monitor, projector or similar device and any hardware required to send output to that device is required to use the optional graphical output feature.
- **Network Adapter**
  A device capable of transmitting and receiving data over TCP/IP and UDP is required on both the client and server machines, unless this is the same machine.

3.3. **Software Interfaces.**

(1) AI agent API – See section 4.2
(2) **DEC-POMDP** problem API – See section 4.3
(3) Graphics plugin API – See section 4.5

3.4. **Communication Protocols and Interfaces.**
Custom communication protocols for interaction between the client and server systems will be developed. The details of these protocols will be determined during the design phase. The design document shall include detailed diagrams of each such protocol.

4. System Features

4.1. **The system must be capable of modeling a DEC-POMDP.**

4.1.1. *Description and Priority.*
Description: DEC-POMDP stands for "Decentralized Partially Observable Markov Decision Process". A formal description of a DEC-POMDP follows.

An n-agent **DEC-POMDP** is defined by a tuple $\langle S, A, P, R, \Omega, O \rangle$ where:

- S is a finite set of world states, with a distinguished initial state s^0
- $A = A_1 \times A_2 \times \cdots \times A_n$ is a finite set of joint actions. A_i indicates the set of actions that can be taken by agent i.
- $P : S \times A \times S \to \mathbb{R}$ is the transition function. $P(s'|s, (a_1 \ldots a_n))$ is the probability of the outcome state s' when the joint action $(a_1 \ldots a_n)$ is taken in state s.
- $R : S \times A \times S \to \mathbb{R}$ is the reward function. $R(s, (a_1 \ldots a_n))$ is the reward obtained from taking joint action $(a_1 \ldots a_n)$ in state s and transitioning to state s'.
- $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$ is a finite set of joint observations. Ω_i is the set of observations for agent i.
- $O : S \times A \times S \times \Omega_n \to \mathbb{R}$ is the observation function. $O(s, (a_1 \ldots a_n), s', (o_1 \ldots o_n))$ is the probability of agents 1 through n seeing observations o_1 through o_n (agent i sees o_i) after the sequence $s, (a_1 \ldots a_n), s'$ occurs.

(Bernstein et al. [1])

Augmentations to the given model

- **Individual Rewards**
  The system should support a more fine-grained reward function instead of a single global award shared by all agents. This in effect generalizes the model to handle a superset of problems called **POSG**s (Partially Observable Stochastic Games).

Priority: High.

4.1.2. *Functional Requirements.*

Index	4.1-R1
Name	Create simulation
Purpose	The system must be able to create a new **DEC-POMDP** simulation given a specification in the manner defined by the problem specification API. (see section 4.3)
Input Parameters	A **DEC-POMDP** specification
Action / Result	• Validates the **DEC-POMDP** specification. • Creates a new **DEC-POMDP** simulation from the supplied specification.
Output Parameters	None
Exceptions	The **DEC-POMDP** specification is invalid.
Remarks	None
Cross-References	None

Index	4.1-R2
Name	Send actions
Purpose	The system must be able to send a set of actions to an agent. The set of actions sent to the agent must be derived from the actions function in a **DEC-POMDP** specification.
Input Parameters	• The agent to send actions to • A **DEC-POMDP** model
Action / Result	• Computes the actions set for the specified agent with respect to the supplied **DEC-POMDP** model. • Sends the computed actions set to the specified agent.
Output Parameters	• The actions set for the specified agent.
Exceptions	None
Remarks	None
Cross-References	None

Index	4.1-R3
Name	Send reward
Purpose	The system must be able to send a reward to an agent. The reward sent to the agent must be derived from the reward function in a **DEC-POMDP** specification.
Input Parameters	• The agent to send a reward to • A **DEC-POMDP** model
Action / Result	• Computes the reward for the specified agent with respect to the supplied **DEC-POMDP** model. • Sends the computed reward to the specified agent.
Output Parameters	• The reward for the specified agent.
Exceptions	None
Remarks	None
Cross-References	None

Index	4.1-R4
Name	Send observation
Purpose	The system must be able to send an observation to an agent. The observation sent to the agent must be derived from the observation function in a **DEC-POMDP** specification.
Input Parameters	• The agent to send an observation to • A **DEC-POMDP** model
Action / Result	• Computes the observation for the specified agent with respect to the supplied **DEC-POMDP** model. • Sends the computed observation to the specified agent.
Output Parameters	• The observation for the specified agent.
Exceptions	Exceptions
Remarks	None
Cross-References	None

Index	4.1-R5
Name	Receive action
Purpose	The system must be able to receive an action from an agent. This action represents the intended action for the sending agent for the next iteration.
Input Parameters	• The agent that sent the action • The intended action sent by the agent • A **DEC-POMDP** model
Action / Result	Action
Output Parameters	None
Exceptions	• Timeout Error. This exception occurs if the iteration mode is set to *synchronous* and the maximum time to receive an action decision from an agent has been exceeded. When this exception is encountered, the system must take two remedial actions. One is that the agent shall be assigned a placeholder "NULL" action for the current iteration. The other is to supply the agent with a notification message that the time to receive an action decision has been exceeded. Rectification of this condition becomes the responsibility of the agent implementation.
Remarks	None
Cross-References	• 4.1-R6

Index	4.1-R6
Name	Iterate
Purpose	The system must be able to iterate a **DEC-POMDP** model.
Input Parameters	• A **DEC-POMDP** model • A current state • A set of $(agent, action)$ pairs
Action / Result	• Each agent in the model is sent a reward and an observation. These are defined by the **DEC-POMDP** problem specification. • The system waits for action decisions from the agents in the model. If the iteration mode is set to *synchronous*, this waiting period is undefined. That is, the system must wait until it has received an action decision from every agent in the model, no matter how long that takes. If the iteration mode is set to *asynchronous*, the system must wait for action decision from the agents up to some maximum interval before proceeding with iteration. • The system invokes the transition function defined in the **DEC-POMDP** problem specification on the input state and the input set of $(agent, action)$ pairs, producing the next state. • Logs the iteration, storing it for future retrieval.
Output Parameters	• The state produced by the transition function
Exceptions	None
Remarks	None
Cross-References	• 4.1-R2 • 4.1-R3 • 4.1-R4 • 4.1-R5 • 4.1-R7 • 4.3-R6 • 4.3-R7 [12]

Index	4.1-R7
Name	Log iteration
Purpose	The system must be able to log the data for every iteration of a **DEC-POMDP** model.
Input Parameters	• A **DEC-POMDP** model • A state • A set of $(agent, observation)$ pairs • A set of $(agent, reward)$ pairs • A set of $(agent, action)$ pairs
Action / Result	The input is stored permanently for future retrieval.
Output Parameters	• A logged version of the iteration.
Exceptions	None
Remarks	• Logging output must be persistent across multiple program runs. • Ideally, the logging output should be stored in a format that is easy to share by email or similar fashion. • It is perhaps preferable, but not necessary, that the logging output is human-readable.
Cross-References	• 4.1-R8

Index	4.1-R8
Name	Replay from log
Purpose	The system must be able to read a series of logged **DEC-POMDP** iterations and play it back.
Input Parameters	A series of logged **DEC-POMDP** iterations, in the format specified by the logging function.
Action / Result	The logged iterations are "played back".
Output Parameters	None
Exceptions	• Invalid Input Format.
Remarks	• While iterating through the log, the rest of the system should behave exactly as if the experiment were live.
Cross-References	• 4.1-R6 • 4.1-R7

4.2. **The system must provide an API for implementing AI agent behavior.**

4.2.1. *Description and Priority.*

Description: The AI agent API is the main software interface between student code and the system. This API must provide functionality to allow users to specify AI agents within the **DEC-POMDP** model in the most general sense and should assume as little about the problem domain as is practical.

Priority: High.

4.2.2. *Functional Requirements.*

Index	4.2-R1
Name	Receive actions
Purpose	The agent API must provide a facility for receiving a set of actions.
Input Parameters	• A set of actions
Action / Result	The agent is updated with a current set of available actions.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	None

Index	4.2-R2
Name	Receive observation
Purpose	The agent API must provide a facility for receiving an observation.
Input Parameters	Input • An observation
Action / Result	The agent is updated with a current observation.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	None

Index	4.2-R3
Name	Receive reward
Purpose	The agent API must provide a facility for receiving a reward.
Input Parameters	• A reward
Action / Result	The agent is updated with a current reward.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	None

Index	4.2-R4
Name	Take action
Purpose	The agent API must provide a facility for indicating an action to take.
Input Parameters	None
Action / Result	The system is updated with the intended action for this agent in the current iteration.
Output Parameters	• An action
Exceptions	None
Remarks	None
Cross-References	None

4.3. **The system must provide an API for defining DEC-POMDP problems.**

4.3.1. *Description and Priority.*
Description: The problem specification API is the main software interface between researcher code and the system. This API must provide functionality to allow users to specify **DEC-POMDP** problems in the most general sense and should assume as little about the problem domain as is practical.

Priority: High.

4.3.2. *Functional Requirements.*

Index	4.3-R1
Name	Define initial state
Purpose	The problem specification API must provide a facility for defining an initial state.
Input Parameters	• A state
Action / Result	The initial state of the **DEC-POMDP** described by the specification is defined to be the input state.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	None

Index	4.3-R2
Name	Define actions function
Purpose	The problem specification API must provide a facility for defining an actions function. The actions function serves to implicitly define the finite set of joint actions within a **DEC-POMDP** .
Input Parameters	• An actions function
Action / Result	The actions function of the **DEC-POMDP** described the specification is defined to be the input actions function.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	None

Index	4.3-R3
Name	Define transition function
Purpose	The problem specification API must provide a facility for defining a transition function.
Input Parameters	• A transition function
Action / Result	The transition function of the **DEC-POMDP** described the specification is defined to be the input transition function.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	None

Index	4.3-R4
Name	Define reward function
Purpose	The problem specification API must provide a facility for defining a reward function.
Input Parameters	• A reward function
Action / Result	The reward function of the **DEC-POMDP** described the specification is defined to be the input reward function.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	None

Index	4.3-R5
Name	Define observation function
Purpose	The problem specification API must provide a facility for defining an observation function.
Input Parameters	• An obervation function
Action / Result	The observation function of the **DEC-POMDP** described the specification is defined to be the input observation function.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	None

Index	4.3-R6
Name	Set iteration mode
Purpose	The problem specification API must provide a facility for setting the mode of iteration for the **DEC-POMDP** simulation. There are two possible options for the iteration mode: *synchronous* and *asynchronous*. In *synchronous* mode, each iteration waits indefinitely for an action decision from every agent in the **DEC-POMDP** model. In *asynchronous* mode, each iteration waits up to some maximum interval for an action decision from all agents before proceeding.
Input Parameters	Mode: one of *synchronous*, *asynchronous*
Action / Result	The **DEC-POMDP** simulation henceforth operates in the specified mode.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	• 4.1-R5 • 4.1-R6 • 4.3-R7

Index	4.3-R7
Name	Set timeout
Purpose	The problem specification API must provide a facility for setting maximum amount of time to wait for action decisions from agents.
Input Parameters	Timeout
Action / Result	If the mode is *synchronous*, has no effect. Otherwise, the **DEC-POMDP** simulation henceforth uses the specified timeout value as the maximum amount of time to wait for action decisions from agents before proceeding with iteration.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	• 4.1-R5 • 4.1-R6 • 4.3-R6

4.4. **The system must provide facilities for optional graphical output.**

4.4.1. *Description and Priority.*
Description: Graphical output should be optional for a subset of **DEC-POMDP** problem specifications.

Priority: Medium.

4.4.2. *Functional Requirements.*

Index	4.4-R1
Name	Toggle graphics
Purpose	The system must have the capability to enable or disable graphical output.
Input Parameters	The graphics output mode, either *enabled* or *disabled*
Action / Result	The system henceforth uses the input graphics output mode.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	• 4.4-R2

Index	4.4-R2
Name	Display graphics
Purpose	The system must be able to display graphical output.
Input Parameters	• A set of graphics plugins • A state
Action / Result	For each graphics plugin currently loaded by the graphics subsystem: • Invokes the plugin's *render* function using the input state and displays the output.
Output Parameters	Graphical output, as defined by the input set of graphics plugins.
Exceptions	None
Remarks	None
Cross-References	• 4.4-R3 • 4.5-R2

Index	4.4-R3
Name	Load graphics plugin
Purpose	The system must be able to load a new graphics plugin given a specification in the manner defined by the graphics plugin API (see section 4.5).
Input Parameters	• A graphics plugin, specified in the manner defined by the graphics plugin API.
Action / Result	The input graphics plugin is included in the set of active graphics plugins.
Output Parameters	None
Exceptions	None
Remarks	None
Cross-References	None

4.5. **The system must provide an API for defining graphics plugins.**

4.5.1. *Description and Priority.*
Description: The graphics subsystem must be modular to allow for arbitrary (i.e. user-defined) output.

Priority: Medium.

4.5.2. *Functional Requirements.*

Index	4.5-R1
Name	Get graphics context
Purpose	The graphics plugin API must provide a facility for obtaining a graphics context on which to draw.
Input Parameters	None
Action / Result	The graphics plugin is updated with a graphics context on which to draw graphics output.
Output Parameters	A graphics context
Exceptions	None
Remarks	None
Cross-References	None

Index	4.5-R2
Name	Render state
Purpose	The graphics plugin API must provide a facility for rendering a given state.
Input Parameters	• A state
Action / Result	Creates graphics output as defined by the implementation of the plugin.
Output Parameters	Graphics output, as defined by the implementation of the plugin.
Exceptions	None
Remarks	None
Cross-References	• 4.4-R2

5. Other Nonfunctional Requirements

5.1. **Performance Requirements.**

(1) The system must be scalable. More specifically, it must be able to run on a commodity laptop as well as utilize a high-performance cluster of computers.

5.2. **Safety Requirements.**

None currently identified.

5.3. **Security Requirements.**

(1) One of the goals is to be adopted for use by researchers outside of the department, therefore security and reliability must be emphasized.
(2) The protocol for communication between clients and the server must employ encryption techniques to ensure integrity of message content and integrity of source of transmission. Confidentiality of message content is not a major concern for the purposes of this application.

5.4. **Software Quality Attributes.**

(1) The system must be extensible so that additional client API implementations and task scenarios can be added. A balance must be struck between useful flexibility and the programming overhead involved with defining new scenarios.
(2) Care must be taken to minimize coupling among modules to avoid propagation of changes. Instability of the API could be a hindrance to acceptance by the target user community.
(3) The client API must be simple enough for students to grasp quickly, yet powerful enough to be useful to experts for research applications.

5.5. **Project Documentation.**
Minimally, the following project documents are planned:

(1) Project proposal
(2) Requirements (this document)
(3) Use cases
(4) Architectural design
(5) Detailed design
(6) Project summary

5.6. **User Documentation.**
Minimally, the following user documents are planned:

(1) High level user manual including installation guide
(2) AI agent specification API
(3) **DEC-POMDP** problem specification API
(4) Graphics plugin API

References

[1] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of Markov decision processes," *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, 2002.

Appendix A. Terminology / Glossary

Multi-Agent Systems: refers to a sub-discipline of machine learning characterized by the interaction of decentralized autonomous agents, whose knowledge of the state of the system is at least partially disjoint. Even in simple cases, calculating optimal solutions can be intractable – especially under real-time constraints. For this reason, algorithms that provably and tractably approximate the optimal are desirable.

An n-agent DEC-POMDP : is defined by a tuple $\langle S, A, P, R, \Omega, O \rangle$ where

- S is a finite set of world states, with a distinguished initial state s^0
- $A = A_1 \times A_2 \times \cdots \times A_n$ is a finite set of joint actions. A_i indicates the set of actions that can be taken by agent i.
- $P : S \times A \times S \to \mathbb{R}$ is the transition function. $P(s'|s, (a_1 \ldots a_n))$ is the probability of the outcome state s' when the joint action $(a_1 \ldots a_n)$ is taken in state s.
- $R : S \times A \times S \to \mathbb{R}$ is the reward function. $R(s, (a_1 \ldots a_n))$ is the reward obtained from taking joint action $(a_1 \ldots a_n)$ in state s and transitioning to state s'.
- $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$ is a finite set of joint observations. Ω_i is the set of observations for agent i.
- $O : S \times A \times S \times \Omega_n \to \mathbb{R}$ is the observation function. $O(s, (a_1 \ldots a_n), s', (o_1 \ldots o_n))$ is the probability of agents 1 through n seeing observations o_1 through o_n (agent i sees o_i) after the sequence $s, (a_1 \ldots a_n), s'$ occurs.

(Bernstein et al. [1])