# Agent-Oriented Programming Framework Design for Real-Time Strategy AI

**XXXXXX** and **XXXXXXX** and **XXXXXXX** and **XXXXXXX**
123 Alphabet Lane
Somewheresville, NotYourState 11111-2222

**Abstract**

We present a Multi-Agent System Framework, KhasBot, to handle the Artificial Intelligence in a Real-Time Strategy Game StarCraft: BroodWar. This Framework will be tested in the SC AI 2011 competition. In this paper we will discuss our reasoning for selecting JADE as our AI backbone, our design strategy, and our creation process in creating KhasBot.

## Introduction

Real-Time Strategy(RTS) Games are an area of interest in Artificial Intelligence Systems for many reasons. They must adapt quickly to changing environments, make decisions on non-complete information, and involve multiple complex AI issues in general (for example, reactive planning, pathfinding, and resource scheduling). StarCraft, a RTS created by Blizzard in 1998, involves three different alien races, in a battle for resources and superiority on a large variety of maps. A human player of this game must keep track of many things at once, like resources, map locations, enemy player actions and locations, building placements and army units. Some players even operate at a level of 200-300 actions per minute(APM) at very high levels of play. This dynamic and complex environment poses a great challenge in creating an AI. For our AI, we selected one race to operate, the Protoss, and we will load our AI with pre-set build orders, a particular battle plan, and operations. Based on the enemy's race selection and map layout, our AI will proceed with one of the appropriate build orders. As the game progresses, based on scouting information and enemy actions, we will change our build order according, in a effort to find the most effective way to defeat our opponent. KhasBot, at this stage, is being built for one-on-one battles.

## Background

What we do in this project is combine two different areas together, BWAPI (BroodWar API) and JADE (Java Agent DEvelopment Framework). StarCraft does not have any officially released AI API, so the BWAPI is injected into StarCraft at runtime, and through the used of a specially designed Proxy AI, the JADE Framework can interact externally with the game. BWAPI and JADE are described in further detail below.



### BWAPI

BWAPI is an open-source API that enables AI creators code injection into Starcraft: Broodwar. It was developed in C++, but many extensions have allowed access to other languages such as Python, C#, and Java. For the purposes of KhasBot, Java has been used. Java also allows us to use ProxyBot, the Proxy AI mentioned earlier, easily.

BWAPI breaks down the StarCraft game into 5 basic objects:

- *Game*: Holds all information about the current game being played, including known unit locations, resource nodes, etc.
- *Player*: Holds all information a player would have, for instance current resources, buildings, and controllable units.
- *Unit*: Represents a piece in the game, be it a mineral, building, or army unit.
- *Bullet*: Represents a projectile from an attack from units that are capable of ranged attacks.
- *Force*: Represents a collection of players attempting a singular objective. This is similar to a co-operative description, where some players may be on the same team.

With these objects, another object *AIModule.cpp* is made as a *dll* which is used as the injected code.

The basic flow of BWAPI then works as follows, when the actual game begins BWAPI will call upon the *onStart()* method from the *AIModule* which will setup all the above objects with the begining of the game information. After this, an *onFrame()* method is called every frame of the game. This is the looping method which holds all the reasoning and computational information of the actual AI. The *AIModule* returns orders to the StarCraft game via *UnitCommand*s. By giving various Units in control of the AI player, such as build, attack, research, and upgrade , the game can progress towards a conflict where either player can obtain victory. The victory condition occurs when the AI player has eliminated the enemy force from the map.

### Agent-Oriented Programming

Agent-Orient Programming is a Programming Paradigm started back in the 1990s. The basis is to on the concept of 'agents' as opposed objects. By doing this, one can focus more readily on the reasoning processes and communication paths of a system, rather than method calls. Aside from communication, Agents can have a focused goal or set of goals to accomplish.

The basic flow of Agent-Oriented Programming, which our KhasBot is designed upon, is that Agents hold a set of beliefs and then based off changes (or additions) to them, the agents belief set will change, causing a possible reaction or change in the behaviour, or attempted actions, of said Agent.

For this particular framework, we used JADE (Java Agent DEvelopment Framework) to model our agents. JADE is a Java middleware framework designed to handle and run agents developed from a package given by JADE. JADE also handles running the agents platform along with all the communication.

## Design

Initially, our first design was based on RTS AI framework, BWSAL. However, this is developed in C++ and is not Agent-Based. The idea behind using an Agent-Oriented design was to offload reasoning of playing the RTS to individual agents that only focused about one facet of the overall problem. We then split the problem into the following 7 agents:

- *Commander Agent* : Concerned with keeping the information up to date and controlling current active build orders. The Commander Agent is the heart of KhasBot, it is also the communcation gateway to and from StarCraft. All game information passes through the Commander Agent and the Commander Agent processes all the commands to pass back to StarCraft.
- *Building Manager Agent* : Concerned with placement of future buildings and builds structures based on the issued build order. The building manager will analyze the current map data to determine where buildings can be placed and can communicate with the Unit Manager Agent to obtain a worker to build the planned building.
- *Structure Manager Agent* : Concerned with training units and upgrading technolgy. Any command that would be normally issued by a player to a building is managed by this agent.
- *Battle Manager Agent* : Concerned with all combat matters including scouting, the act of sending a unit out into the fog-of-war to obtain valuable information. The Battle Manager Agent controls all attack units, battle tactics, and makes decisions with those units based on scouting information, enemy movements and build orders.
- *Resource Manager Agent* : Concerned with all resource gathering, including "rate of" gathering, worker management and needed materials for segments of the build order.
- *Map Manager Agent* : Concerned with maintaining and updating map information and determines efficient path finding for any units in AI control.
- *Unit Manager Agent* : Concerned with mainting unity between all the pieces used by the other Manager Agents and merging together unit commands for the Commander Agent.

In addition to the above agents, we also developed the *Proxy Bot Agent*, that was in charge of interfacing with the BWAPI and socket information from the StarCraft game and converting that information into messages the Agent System could understand. It also converted messages from the Agent System to something the BWAPI and socket information understand.

Most of our Manager Agents follow a similar behavior to each other. Due to this, we were able to create a superclass KhasBotAgent that took care of most of the repetitive tasks. With many agents working on seperate areas, the key here is organizing their communication with each other. Luckily, all of our protocols exhibit a request/inform type flow, which is a core design structure of the JADE Framework. All manager agents are informed of game updates by the *Commander Agent* and all Manager Agents request units and inform new commands to units to *Unit Manager Agent*.

The actual StarCraft Game is mapped to a *GameObject* that the agents use to keep track of what they are doing. As the game progresses, the Multi Agent System(MAS) continues to get game updates via a *GameUpdateObject*, which is essentially a reduced game object. The *GameObject* is broken down into two different things, Player Information and Map Information. Map Information includes items like where is it safe to build or walk and what the starting locations are for the map in the game. Player Information handles all units and ways the units can do research or upgrades. These units include structures, mineral patches, and all other pieces in the game that can change.

## Implementation

The basic idea that each Manager Agent follows is that they are each an agent inside of JADE that have multiple threaded behaviours that focus on different protocols between the various Manager Agents.

Due to the multiple behaviours at once, each agent heavily uses the DataStore ability from JADE, a central data location on the platform, to keep its information consistent with the other behaviours.

### Build Orders

## Results

Initial tests have been positive. The message passing between agents have resulted in simple build orders being executed.

## Conclusion and Future Work

We have found that building StarCraft AI is viable using an Agent-Oriented Programming Framework. The comunication structure established by JADE allowed us to create agents capable of specific goals and relaying imformation based on those goals to other agents and to StarCraft. Splitting the computation needed in an AI among agents

provides another avenue for creating StarCraft AI and ultimately video-game, military, or management AI.

In the future, we will continue to refine KhasBot with more build orders and potential decisions for those build orders. We will also look into generalizing KhasBot for all three races, as our Build Order parser will allow for that.