An Agent-based Software Architecture to Aid Human Operators during UAS Target Engagement

Derek McColl, Ming Hou
Defence Research & Development Canada
Toronto Research Centre
Toronto, Canada
derek.mccoll@drdc-rddc.gc.ca, ming.hou@drdc-rddc.gc.ca

Kevin Heffner1, Simon Banbury2
1Pegasus Research and Technologies
Montreal, Canada
k.heffner@peretec.com
2C3 Human Factors Consulting Inc.
Montreal, Canada
simon.banbury@c3hf.com

Abstract — Results of simulation experiments with UAS crews have indicated that the crew had difficulties following rules of engagement and the law of armed conflict when engaging a target with a weapon. Defence Research & Development Canada is developing an intelligent adaptive interface, called the Authority Pathway for Weapon Engagement (APWE), which is a decision aid to support UAS crews in following target engagement procedures. This paper identifies the high level requirements and describes the software architecture that meets these requirements. Intelligent software agents are used to adapt APWE interfaces to the needs of the UAS crew in real-time. Software agent requirements are identified in terms of agent framework, agent behavior, and information exchange requirements. A novel client-server architecture with a unified model language state machine using belief-desire-intention agents is proposed using zero-cost open source software.

Keywords—software requirements; software architecture; UAS; software agents; decision aid; target engagement

I. INTRODUCTION

There are several benefits of using software agents for simulation-based training, experimentation and decision-support applications. First, automation agents are well-suited for representing physical entities in a synthetic environment, such as human behaviors, machine and environment models [1]. Also automation agents can assist humans in decision-making and to improve situational awareness [2]. Finally, the choice of agent-based software architectures facilitates the use of additional artificial intelligence (AI) approaches such as machine learning (ML) as a means to generate and/or refine agent behaviors without requiring additional software design and implementation.

The information required to build military simulation models often is based on doctrine, procedures and subject matter expert (SME) experience and expertise available primarily as free-text expressions or as semi-structured message templates. Representing this information in a form that is suitable for machine processing by agents can be a challenge. For example, rules of engagement (ROE) are expressed using natural language that must be translated using a formal representation in order to be consumable and usable by a simulation or other software applications [3]. Another difficulty arises when it becomes necessary to evolve simulation models over time. Indeed, modifying existing simulation behaviors in light of the software linkages and dependencies among the instructor, operator and the simulation modeller interfaces generally translates into costly resources requirements for simulation engineers, experiment, and/or training personnel.

The judicious use of AI technologies including the use of software agents and ML can contribute to reducing these costs by constructing resilient, self-adaptive simulation components and automation aides. For instance, the Belief-Desire-Intention (BDI) agent software model can effectively replace much of the conventional if-then logic and rule-sets with more flexible knowledge repositories, belief sets, and plan libraries that are amenable to integration with ML algorithms [4]. This, in turn, can facilitate the refinement of agents (i.e., simulation behaviors and automation aides) by providing the mechanism for the agents to improve themselves over time without breaking the applications.

This paper presents a novel agent-based software architecture that has been designed by Defence Research and Development Canada (DRDC) for developing and integrating intelligent interfaces and multi-agent systems for use within an unmanned aircraft system (UAS) experimentation test-bed. The core business logic combines a hierarchical state machine using Unified Modeling Language (UML) state charts with BDI software agents that provide specific behaviours for various state machine functions. The solution includes high-performance, scalable web technologies and supports standardized simulation protocols. This architecture is being used to develop the UAS Authority Pathway for Weapon Engagement (APWE) to support human factors experiments on the use of intelligent automation aides as a means to improve UAS crew situation awareness (SA) and facilitate decision-making during target engagements.

II. BACKGROUND

DRDC is supporting the Royal Canadian Air Force (RCAF) Joint Unmanned Surveillance and Target Acquisition
System (JUSTAS) project to acquire a medium altitude long endurance UAS for domestic and international operations. This UAS will have ISR and precision strike capabilities [5]. One of DRDC tasks for the JUSTAS program is to investigate UAS Ground Control Station (GCS) operator interface technologies to make recommendations on UAS GCS design.

In 2014, DRDC, in collaboration with the United States Air Force Research Laboratory, developed an experimental UAS GCS platform called Testbed for Integrated GCS Experimentation and Rehearsal (TIGER) [6,7]. DRDC uses TIGER to investigate GCS interface technologies for JUSTAS. TIGER is comprised of 11 networked workstations including six UAS crew workstations and five workstations for simulation control and data collection. The TIGER crew workstations follow the RCAF squadron level UAS control concept that includes a pilot, a sensor operator, two image analysts and two electronic warfare analysts. Initial experiments with TIGER indicated that the UAS crew had difficulty in following ROE and the Law of Armed Conflict (LOAC) when performing target engagements [8].

More recently DRDC developed the concept of an APWE to assist the UAS crew in following procedures for target engagement that adheres to ROE and LOAC. These procedures includes sequential and concurrent activities for multiple UAS crew members [8,9]. The main functions of APWE include i) providing the UAS crew with SA on the steps required during the target acquisition and engagement; ii) provide timely and detailed feedback on the reason a target engagement step has been denied; and iii) facilitate the crew’s decision-making processes during target acquisition and target engagement.

Fig. 1, a screen capture of the prototype APWE application at start-up, is a graphical representation of the principal steps and information set that comprise the main interface. This primary display indicates: 1) the progression of steps, as related to the target engagement procedures; 2) information regarding the eyes on target status; and 3) mission time (MT) and time on target (TOT) times. The two sets of vertical bars at the top progression pane provide information on the communication status between the UAS and its external tasking authority.

III. Requirements

APWE has been designed as an Intelligent Adaptive Interface (IAI) multi-agent system. An IAI is defined as a dynamic user interface of a human-machine system that adaptively changes its appearance and/or control characteristics in response to external events in real-time [10]. A full set of formal APWE requirements has been specified in the form of a UML model, but due to space constraints, in this paper we describe only a subset of high level requirements that guided the elaboration of the software architecture.

A. Quality Factors, Design Constraints and Assumptions

This section highlights the principal assumptions and design constraints that dictated the architecture and design decisions.

1. The architecture will utilize solely no-cost open-source software components;
2. Java will be the main programming language;
3. The solution must be deployable to a variety of platforms, including various Linux and Windows OS;
4. It should be easy to maintain and evolve the solution to meet future experimentation needs and perform updates and bug fixes;
5. The solution should be highly scalable to account for potentially large numbers of agents and other software resource requirements; and
6. The solution should use an open architecture approach to facilitate future integration with national and coalition partners.

B. APWE User Interface Requirements

1) Target Engagement Situation Awareness

In order to enhance UAS crew SA, the APWE interface must display: 1) the target engagement procedures; 2) the status of the current authorization step; and 3) any information pertinent to the completion of the current step. A prototype of an APWE interface main panel is shown in Fig. 1.

2) Adaptive Interface

To facilitate decision-making, APWE must adapt its interface by automatically highlighting the current step and displaying pertinent information for the UAS crew. APWE should also facilitate communication between the UAS crew and its external tasking authority as well as analyze and display data from the UAS and simulation systems that are relevant for the crew to perform the target engagement procedure.

3) Intelligent Interfaces

APWE must have different interfaces for different users. UAS crew members have different responsibilities during target engagements and will need different interfaces. For example the pilot, also acting as the crew commander (CC), requires the ability to confirm UAS weapon settings while the image analysts require a function to perform collateral damage estimates. Moreover, the experimenters, simulation control
personnel, and role players (e.g., acting as the UAS external tasking authority) will also need interfaces specific to their roles on TIGER. To support this variety of roles, the APWE software architecture needs to support multiple graphical user interfaces (GUI), each with different information and interaction requirements. These interfaces will also need to change automatically based on input from the UAS crew, the UAS controlling authority, and other TIGER systems. All of these interfaces must share the same target engagement activity and be aware of its current status (being intelligent).

4) UAS Crew-specific and Other IAI

ROE and LOAC can change rapidly during a mission, e.g., friendly troops could be ambushed and quickly need support from a UAS that was already following a high value target. To facilitate such situations, the APWE architecture requires the ability to support multiple concurrent engagement procedures with different ROE and LOAC. Due to the time-sensitive nature of target engagements the APWE architecture must be able to quickly update information in all of its components.

The look and feel of role-player and experiment support staff provided by front-end web technologies is adequate. However, UAS crew user interfaces require specific attention to detail in terms of shapes, colors, animation and user interactions. Therefore, these IAI, referred to as rich clients, will require a full-featured GUI framework.

C. Software Agent Requirements

In order to automatically update its interfaces and communicate with the various components on the TIGER UAS experimentation testbed, APWE will require a set of intelligent agents. The requirements regarding the use of software agents are composed of three main sets: 1) agent framework requirements; 2) functional requirements for agent behaviours; and 3) information exchange requirements for inter-agent communication and extra-agent exchanges (i.e., between the agents and other APWE components, and TIGER components such as simulations and command and control systems).

1) Software Agent Framework Requirements

A survey of agent frameworks is out of the scope of this paper. However, previous related work has indicated that the BDI (agent) approach was well suited for the development of intelligent adaptive systems (IAS) since BDI agent design method supports agents acting rationally in pursuit of goals, has been proven in real-world systems, and can be implemented relatively easily [2]. Furthermore, the BDI agent approach is consistent with the event-based application paradigm (i.e., TIGER) since events are the primary triggers and inputs to the agent behaviour models. In addition, based on the assumptions from the section III.A, the agent framework should be zero-cost, open-source and Java-based.

In terms of usability, the agent framework should allow for the definition of a family of agents in a hierarchical structure. A hierarchy of agents allows for simpler individual agents and promotes traceability of the actions taken by the multi-agent system. To ease development and testing of future APWE capabilities, the agent framework should provide a simple mechanism for creating new agents.

2) Agent Behaviour Requirements

Agent behaviours will support interface automation functionality based on the specific target engagement step currently being executed. For example, in the case of collateral damage estimation (CDE), agents will be tasked with the identification of friendlies and hostiles within the vicinity of a target. Depending on weapon availability, the other agents will automatically calculate and update the blast zone display on the map view. Another set of agents will constantly identify and review the applicable ROE and in the case of a conflict or inconsistency, notify the operators, as required.

The ROE verification agent will collaborate with one of the most important agent types: the checklist verification agent. Many of the target engagement steps have dynamic checklists that indicate to the UAS crew the status of any given step.

Another class of agents are responsible for collecting and processing information related to operator state monitoring. This information can result in the generation of triggers that may spawn new agent activities. In addition to supporting IAI functions, agents may also execute tasks for the benefit of a more senior agent. In other words, the agents will be used to collaborate in an orchestrated manner, including as part of a hierarchal network of multi-level agents, consistent with the approach outlined in [2,10].

3) Information Exchange Requirements

The agents will share a common set of messages. Having a common set of messages will facilitate integrating a hierarchical family of agents and will support a distributed, event-based, messaging architecture. The agent message set will include inter-agent messages and extra-agent messages, such as simulation events (e.g., Distributed Interactive Simulation (DIS) events), APWE events, etc. Since the agents will need to communicate with UAS systems and APWE interfaces the agent message set will need components that allow the agents to exchange information with these other systems in addition to other agents. Inter-agent information exchange should support delegation of tasks by senior agents to junior agents.

IV. Architecture

The APWE system and agent requirements were used to drive the development of the APWE architecture.

A. Client-Server Web Socket Architecture

A client-server asynchronous, event-driven, messaging architecture was chosen to meet these APWE and agent requirements. Specifically, the client-server approach allows for the server to store information concerning the target engagement for all the client applications while the various client applications are updated with only the information each client requires. The clients can also send requests to the server, as required. The server maintains a model of the target engagement procedure in the form of a state machine. The server also manages client application requests and generates
events and updates that are propagated to client applications. The asynchronous event-based nature of the architecture allows the server to update each of the client applications to target engagement procedure updates and changes as they occur, which is critical for applications such as time sensitive targeting (TST). This architecture design also allows for the various users to have unique client applications that permit control over only the APWE systems or information they need to execute their tasks in the target engagement procedure.

The Web Socket protocol, RFC 6455 [11] does not require polling and is well-suited for low-latency, efficient, two-way, persistent event-driven communication. Although typically utilized for client-server web applications, it was decided to use web sockets for all communications between the server and all clients, including rich or desktop clients (i.e., UAS crew APWE clients). Therefore, the Apache Software Foundation (ASF) Tomcat Java Servlet implementation was selected. The APWE server was implemented as a web socket-enabled Tomcat Servlet. This corresponds to the third level of the software stack, depicted in Fig. 2. The first and second layers correspond to zookeeper and curator, respectively, and allow for distributing the server across multiple nodes in order to satisfy the scalability requirement.

B. APWE Server Core: AP State Machine

The business logic is derived primarily from operational procedures and input from military SME. However, this logic is subject to change since the RCAF UAS target engagement procedures currently are being defined. Also, the business logic depends on the specific mission type, e.g. pattern of life (POL) or troops in contact (TIC). It is therefore important to be able to: 1) communicate the business logic to stakeholders; 2) easily update the logic based on SME feedback; and 3) dynamically change mission types during mission execution. For these reasons, the core business logic was implemented as a UML state machine combined with the Model-View-Controller (MVC) design pattern.

![Fig. 2. Authority Pathway Software Stack.](image)

![Fig. 3. APWE High-level State Machine: Pattern of Life (POL)Mission](image)
The Spring Java software framework by Pivotal [12] offers several packages, including an example of a Web-MVC state machine with an embedded Tomcat server. The Spring State Machine (SSM) package also provides support for the Eclipse Papyrus plugin that allows for defining a hierarchical state machine as a set of UML state charts. Fig. 3 is the UML state chart for the actual implementation of the first-level business logic for a POL mission. This state chart is loaded dynamically by the APWE server. The SSM package provides support for all UML state machine functionality, including: hierarchical state machines, orthogonal regions, states, extended states, guard conditions, actions & transitions, internal transitions, events and entry & exit conditions [13]. The APWE prototype application utilizes all of these features. In Fig. 3, all rectangular forms represent states. The EYES_ON_TARGET_REQUIRED state is a second-level sub-state machine and represents the main area of concern for the target engagement activity. In fact, the CDE and weapon engagement planning (CDE_WEP) state is also a (third-level) sub state machine. Every line with an arrow represents a transition, a subset of which involves external events that are shown in capitals prefixed with “EVENT”. Black dots represent initial states, required for all state machines. Guard conditions are shown between square brackets and resolve to logic statements, generally concerning extended state variables.

The SSM package also provides state machine listener and controller classes for handling internal (state machine) events and state changes as well as communication between the server and clients. Fig. 4 is the sub state machine depicted in the previous figure as the “CDE_WEP” state. This sub state machine illustrates the orthogonal regions feature along with the fork and join functionality. This is required, in this case to allow for concurrent CDE_WEP. Both CDE and WEP must be completed by crew and confirmed by the CC before the subsequent target engagement steps can be started. All of the states shown in yellow in Fig. 3 are represented as sub state machines.

C. Client Applications

The client application shown in Fig. 1 is an example of a UAS crew rich client that was implemented using the JavaFx GUI framework. The rich clients implement web socket communications to subscribe to server events and updates and also to make requests. UAS client applications are configurable and use profiles to define their functionality based on user credentials. Another UAS client application is used to provide a common SA view to a large shared display.

Fig. 4. Weapon engagement planning and collateral damage estimation sub-state machine.

Fig. 5. APWE Client Server Application.
Another type of client application is intended for use by the TIGER simulation control personnel and/or role players that will be acting the role of the UAS external tasking authority. This application is a web application that utilizes Bootstrap and Angular JS to provide the front-end user interface and presentation logic, respectively. The prototype APWE web client is shown in Fig. 5. JavaFX and Bootstrap/Angular JS are the top-level of the APWE software stack shown in Fig. 2.

D. BDI Agent Integration

The Jadex framework [14] is an open source Java BDI agent framework based on the Foundation of Intelligent Physical Agents (FIPA) IEEE specification [15]. Jadex has built-in support for domain knowledge in the form of beliefs and utilizes the RETE pattern-matching rule engine [16]. Jadex agents are highly integratable multi-agent software components known as active components. The BDI agents implement the architecture shown in Fig. 6. Jadex BDI agents include inter-agent communication mechanisms.

Jadex agents will be spawned as new processes by the APWE state machine when state machine actions are invoked. Jadex agents will also be able send and receive state machine events and subscribe to state machine data.

As the number of agents grows, resource constraints on the APWE server host machine could have posed a problem. However the use of the ASF zookeeper/curator packages allows for a virtually unlimited number of agents running on a fault-tolerant distributed server.

V. CONCLUSIONS

APWE has been designed as an IAI acting as a decision aid for UAS crew members. APWE includes dynamically adaptive user interfaces, driven by intelligent agents, to assist UAS operators to follow target operational procedures and relevant ROE and LOAC for target engagements. This paper outlines the requirements and software architecture for APWE and its intelligent agents to effectively assist UAS operators. APWE has been designed as a client-server asynchronous event-based messaging architecture with the core business logic implemented as a hierarchal UML state machine. This state machine spawns BDI software agents as required to execute actions. APWE has been developed entirely with zero-cost open source software packages. Future work will include implementing BDI agents, persistence capabilities, and integrating APWE on TIGER and evaluating the software with UAS crews.

ACKNOWLEDGMENT

We would like to thank Major Mario Charon and Serge Pelletier for their help developing the concept and user interfaces for APWE.

References


Fig. 6. Jadex Agent Abstract Architecture [17]