

Specifying Requirements in a Multi-Agent System with Use Cases

Michael Papasimeon and Clinton Heinze
{ michael.papasimeon, clinton.heinze } @dsto.defence.gov.au
Aeronautical and Maritime Research Laboratory
Defence Science and Technology Organisation
PO Box 4331
Melbourne, Victoria, 3001
Australia

Abstract

The software engineering of multi-agent systems demands specification of the required agent behaviours to provide documented requirements for the design and implementation phases. A methodology for the analysis and specification of agent behaviours is proposed, which arises from a lengthy experience in the construction of multi-agent simulations for military operations research. The methodology builds upon the existing use case modelling techniques provided by the Unified Modeling Language (UML) and is in keeping with the agent extensions to the UML proposed elsewhere. A case-study from a specific multi-agent air combat simulation accompanies the elaboration of the methodology.

Keywords

Engineering methodologies, standards, simulation, use cases

Specifying Requirements in a Multi-Agent System with Use Cases

Michael Papasimeon and Clinton Heinze

Abstract

The software engineering of multi-agent systems demands specification of the required agent behaviours to provide documented requirements for the design and implementation phases. A methodology for the analysis and specification of agent behaviours is proposed, which arises from a lengthy experience in the construction of multi-agent simulations for military operations research. The methodology builds upon the existing use case modelling techniques provided by the Unified Modeling Language (UML) and is in keeping with the agent extensions to the UML proposed elsewhere. A case-study from a specific multi-agent air combat simulation accompanies the elaboration of the methodology.

1 Introduction

Air Operations Division (AOD) of the Defence Science and Technology Organisation (DSTO) provides advice to the Australian Defence Force (ADF). Operations research studies form the basis of this advice. Typical studies will compare potential hardware upgrades to an aircraft – perhaps a new missile or a modification to the radar[1, 2]. On occasion the acquisition of a new aircraft will dictate the evaluation of candidates. Comparisons of tactical decision making can also be made resulting in the provision of advice to the ADF about the adoption of standard operating procedures[3].

Studies are conducted using constructive simulations of air combat. The complexity of the domain presents a number of modelling challenges. The physical systems – aircraft, missiles, radars – are sophisticated, highly dynamic, and must be modelled to a sufficient level of fidelity. The human operators of those systems – fighter pilots, fighter controllers, mission commanders – must also be modelled. Intelligent agents are used as the computational models of human decision making within these simulations[4, 5].

The specification of the requirements for the agent component of these simulations is guided by operational scenarios that help to define the research questions of interest. This paper proposes a methodological approach for a use case oriented specification of the agent requirements and details a case-study. By focussing on scenarios to constrain the scope of the development it differs from other modelling methodologies that stem from a domain focussed view [6].

The methodology suggested in this paper makes use of extensions to the Unified Modeling Language (UML) [7]. This is not dissimilar to work by Odell, Parunak, and Bauer, [8, 9] and Burmeister [10] in that it presents a view of agent oriented development that allies closely with the experiences of object oriented development. The development of Agent UML (AUML) [8, 11] proposes to extend the UML for agents but concentrates on the documentation of the interactions between agents. The extensions proposed in this paper extend the UML for use case analysis. Future papers will address specific issues related to UML extensions for the design and implementation of BDI [12] agents.

2 Use Case Analysis for Requirements Specification

A number of analysis techniques are used in software engineering for specifying requirements. These range from natural language software requirements specifications to formal methods using mathematical languages such as Z [13].

Use case analysis is one particular technique that has shown value for specifying requirements for many different types of systems. Use case analysis has gained prominence due to its inclusion in the Unified Modeling Language (UML) and therefore has gained particular prominence in the construction of object oriented systems.

A use case driven approach to developing a system has many benefits. These include:

- A use case is a view of the functionality (or use) of a system from the user's perspective.
- A use case can be considered as a collection of scenarios about system use - developers and users often use scenarios to help them understand requirements.
- Software can be developed using an iterative and incremental approach with one or more use cases defining the subset of functionality for the user which will be implemented in a given iteration.
- Requirements can be specified in non-technical terms that a user can understand without needing to be a software engineer.
- Traceability of requirements through the design, implementation and testing phases of a project is improved.

2.1 Use Cases for Non Agent Applications

A use case can be loosely defined as a typical interaction between a user and a software system. For example in a paint program, some example use cases may be "rotate an image" and "convert image to JPEG". Although use cases can vary in size, their most important characteristic is that in some way each use case specifies some functionality which the software provides to the user, allowing the user to achieve some discrete goal. More precisely, a use case can be defined as follows:

A use case is a scenario about system usage, which is described by a sequence of events. Each sequence is initiated by entities known as actors that may be a person, another system or the passage of time. The result of the sequence of events must be of use to the actor initiating the sequence or another actor.

A use case model provides the information that would be expected in a traditional software requirements specification. For example, the actors are commonly the users of the system being specified, and the use cases provide the functional requirements of the users.

The UML supports a standard notation for use cases and actors as shown in Figure 1. Participation in a particular use case by an actor is shown by a line between the actor and the use case. The notation allows for requirements to be abstracted and encapsulated in a structure that is logical and concise.



Figure 1 Notation for Actors and Use Cases

A use case can use or extend the functionality provided by another use case. In the UML stereotypes <<uses>> and <<extends>> define these specific associations. Figure 2 shows an example of <<uses>> and <<extends>> using some functionality from a simple paint program. An artist can “Convert Image to JPEG” which uses the “Save Image” use case and they can “Rotate Image” extends the general use case “Transform Image”. The box surrounding all the use cases defines the system boundary of the paint program, with the artist actor being external to the system. It is possible to have multiple actors in use case diagrams. For example we might have an “Image Analyst” or a hardware device such as a “Printer” as actors interacting with the paint program.

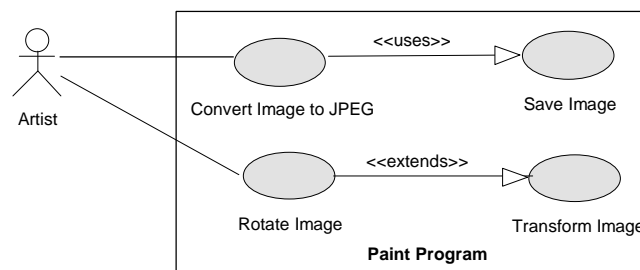


Figure 2 Part of a use case model for a simple paint program

One or more use case diagrams is not enough to fully specify the requirements for a system. Each use case in the diagram needs to be fully documented and specified. The

amount and type of documentation needed for each use case varies depending on the type of system being specified. It can range from a text description to including different types of UML diagrams. Common approaches to documenting uses cases for object oriented systems can be found in [14, 15].

2.2 Use Cases for Agent Applications

Use case analysis has proved to be useful and successful for requirements specification of object oriented systems. The logical question is then, can this type of analysis be used for agent oriented systems? It is proposed that with some modification and extension, use case analysis can become a useful tool for single and multi-agent requirement and behaviour specification.

Wooldridge and Jennings [16], characterise agents as exhibiting autonomy, social ability, reactivity to an environment which they are a part of as well as pro-activeness. The concept of an agent being part of an environment which it is aware of and interacts with distinguishes agents from actors. Actors are external entities (humans or other systems), that interact with a particular software system. Agents, on the other hand can be part of the software system being built. The environment in which they are situated in may be a synthetic virtual environment, a real environment, or a combination of the two, depending on the type of agent. Table 1 gives examples of different types of agents in different types of environments.

Agent	Examples	Environment
Non Player Characters and Simulated Humans	Enemy monsters in video games, artificial pilots in military simulation.	Synthetic/Virtual
Information Gathering	Web Crawlers	Internet
Robot Based	Agents on board planetary space probes.	Real (Planet Surface)

Table 1 Agents and Their Environments

Agents are often ascribed with qualities and capabilities that would normally be associated with human actors. For many types of systems which have entities that can be ascribed with notions of agency (such as non player characters in video games), it often helps in the specification of behaviours if we treat these entities as real humans.

Given that use case analysis has proved successful in specifying requirements of systems for human actors, it is proposed that the same techniques (with some modifications) could be applied to agent oriented systems. Using the UML's extension mechanism of stereotyping, two extensions for use in the requirements specification of agent oriented systems are proposed.

<<agent>>

An <<agent>> is an <<actor>> that is part of a software system, and has behaviours defined by agent use cases in the context of the system. In a use case diagram an agent

is confined inside the system boundary. An agent can interact with other agents or other actors through agent use cases. The notation for an <<agent>> is the same as for an <<actor>>.

<<agent use case>>

An <<agent use case>> is a collection of scenarios about the agent's interaction with a system. Each scenario describes a sequence of events which the agent initiates. The result of the sequence of events has to be something of use to the initiating agent, to another agent, or to any other actor. The diagrammatic notation for an <<agent use case>> is the same as for a standard UML use case.

The two stereotypes defined above complement the UML's existing use case notation and methodology. Many multi-agent software systems are hybrid systems where multiple agents can interact with multiple actors. For example, consider many types of video games where human players (often in teams) compete against enemy opponents which may be considered as agents in a synthetic environment. Such a system is a hybrid multi-actor and multi-agent system where standard use case analysis, and the agent oriented extensions defined above can be combined to specify behaviours of all the entities both human and artificial.

3 Methodology for Specifying Agent Systems

A methodology for specifying agent systems using use cases is required. We suggest a three step process which involves the following steps:

1. Identify agents and actors
2. Identify use cases for agents and actors
3. Document each use case

3.1 Identify Agents and Actors

The analyst first decides what actors and what agents are going to be involved in the software system being specified. To identify the actors in a system, the analyst must determine who will be the users of the system, and with what external systems the software will be interacting. Identifying the agents in a system is a bit more difficult. In a military simulation this involves deciding what human operators to model. In a video game the agents are likely to be all non-player characters and in an information retrieval system the agent may be a web crawler.

3.2 Identify Use Cases for Agents and Actors

For each actor in the system the use cases are specified by drawing a use case diagram with a number of ellipses joined by lines to the actor. Each ellipse signifies a use case or some functionality provided by the system to the actor. The same approach can be applied to determining agent use cases for agents. However, instead of considering

functionality for an agent use case, the types of behaviours an agent exhibits in the context of an environment should be considered. Therefore for each agent:

1. Decide on the behaviours that need to be exhibited by each agent – each type of behaviour becomes an agent use case.
2. Decide on the types of actor-agent and multi-agent interactions there will be in the system. Each interaction becomes an agent use case.
3. Refine the use case model by adding lower level functionality for actors and lower level behaviours for agents, making use of the <<uses>> and <<extends>> associations to show the relationships between use cases.

3.3 Document Each Use Case

Once an overall picture emerges of the functionality provided to actors and the general behaviours of agents, it is time to document the use cases. Use case documentation for actors in object oriented systems is described elsewhere [14]. For agent oriented specification of use cases, the following structure has proved to adequately capture the required behaviours for the projects on which use case analysis has been used.

1. **Use Case Name** – The name of the use case.
2. **Descriptive Text** – One or two paragraph description of what the use case involves, describing the agent behaviour in the context of the agent's environment (ie interaction with other objects), and interaction with other agents and actors.
3. **Agents** – A list of agents involved in this use case.
4. **Actors** – A list of actors involved in this use case.
5. **Use Case Associations** – A list of other use cases associated with this use case. This includes use cases which this use case <<uses>> and <<extends>>.
6. **Environment** – A description of the environment and the objects in the environment that the agent interacts with in this use case. A UML class diagram is often useful in this section to show how the agent fits into the environment.
7. **Pre-Conditions** – A list of all conditions that must hold to be true before the agent can initiate this use case.
8. **Post-Conditions** – A list of all conditions that must hold to be true after the agent has completed a behaviour defined by this use case.
9. **Flow of Events** – A numbered list of activities or events which the agent initiates which defines the behaviour of the agent in this use case. This section should also include a UML activity diagram which depicts the flow of events. The use of *swim lanes* shows interactions in the flow events with other agents. The use of UML sequence and collaboration diagrams can also be used to visualise the interaction of the agent with the entities or objects in the environment in a fashion similar to that suggested by Odell et al. [9].

10. **Alternative Flows** – A numbered list of activities or events which the agents performs in anomalous, or exceptional circumstances. An alternative flow describes agent behaviour in this use case not handled by the main flow of events. It is possible to have more than one alternative flow in a use case. This section should also include relevant activity, sequence and collaboration diagrams.

4 Case Study

This section describes a case study of the application of the proposed methodology. The system under consideration is a military simulation of air combat.

4.1 Domain Description

The BattleModel is one of the large simulation environments in use by AOD. It provides an object oriented framework for simulation and is equipped with models of the required physical systems. Intelligent agents are used to provide the models of the human operators. Analysing, designing, testing and implementing the agents for a particular study is a common and ongoing task. There is the capacity to reuse existing agents, if not in their entirety then at least in part, but it is common to require new agent functionality. In either case, it is necessary to specify requirements so that decisions about reuse or development can be made.

4.1.1 Air Combat

The complexity of the air combat domain necessitates a certain amount of background information to clarify the roles and capabilities of the parties involved. The following definitions may assist in understanding the case study.

DCA: Defensive Counter Air; an aircraft with the responsibility for defence of an area or an asset. This achieved by flying a Combat Air Patrol (CAP). Fighter aircraft operate in pairs, as a leader and a wingman. Typically the leader is the more experienced pilot and is generally responsible for the tactical control of the mission.

Strike: A role that involves attacking a surface target. Strike aircraft are less capable of defending themselves than fighter aircraft sometimes necessitating that they be accompanied by a fighter escort.

Escort: A fighter aircraft that flies close to the strike aircraft in order to protect it from hostile threats.

AEW&C: Airborne Early Warning and Control; an aircraft equipped with highly capable radars and other sensors. These aircraft have a surveillance role and, because of their significant ability to detect and track targets, also provide advice to fighter aircraft concerning tactics to adopt. A crew of more than six is normal. They will fill a variety of roles: pilot, mission commander, fighter controller, and sensor manager.

4.1.2 Operational Scenario

The case study presented in this paper will focus on the specification of behaviours of the agents for the studying of tactics for defending a ground target against an air attack. The details of the study have been fabricated to avoid security related concerns but the methodological approach presented is that used in the construction of current agent systems.

The scenario is presented below in Figure 3 and the description in Table 2. This type of layout is an abbreviated form of the scenarios provided by air-force personnel to AOD for operations research purposes. In addition, analysts have access to standard operating procedures manuals, tactical manuals, their own domain knowledge and experience, and clarifying interviews with pilots and controllers.

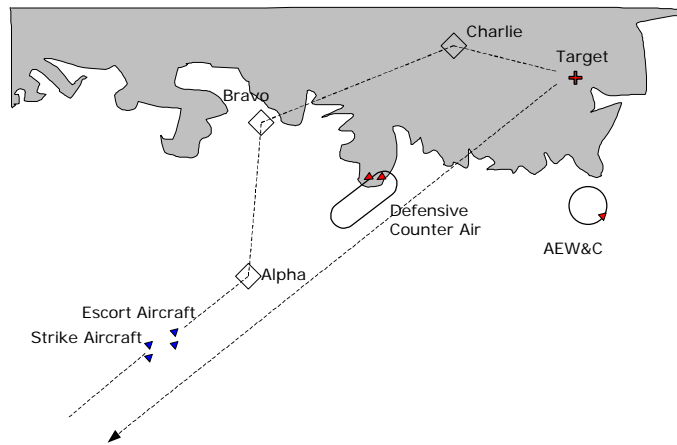


Figure 3 Schematic of Operational Scenario

STUDY PURPOSE : Examine the impact of the presence of a fighter control capability in opposing a strike mission.	
Red Side	Blue Side
<p>The pair of escort aircraft will accompany the pair of strike aircraft. At waypoint alpha the aircraft will descend to 500 feet and increase speed to 560 knots. At waypoint Charlie the strikers will split from the escort, reduce altitude to 250 feet and commence a bombing run into the target.</p> <p>If the escorts encounter opposition fighters they will intercept and kill or divert them.</p> <p>The escorts should attempt, where possible to remain in contact with the strikers to provide as much protection as possible.</p>	<p>An AEW&C aircraft is airborne to provide surveillance coverage. In the event of an attack, the AEW&C will vector fighters from their airborne combat air patrol (CAP) toward the attackers providing tactical control.</p> <p>A pair of fighters in a defensive counter air role is on a CAP covering the expected route of the attackers. Upon a radar detection or a command from the AEW&C these fighters will attempt to intercept the attackers. The fighter controller on-board the AEW&C aircraft will provide varying levels of control ranging from: no advice; to advice about the position of any hostile aircraft; to advice about the tactics to adopt.</p>

Table 2 Description of Operational Scenario

4.2 Identifying Actors

A simple assumption in identifying the agents in this system is that an agent will be substituted for every human in the real scenario. This leads to the identification of the actors and agents (see Figure 4). This assumption can be revisited as the analysis proceeds.

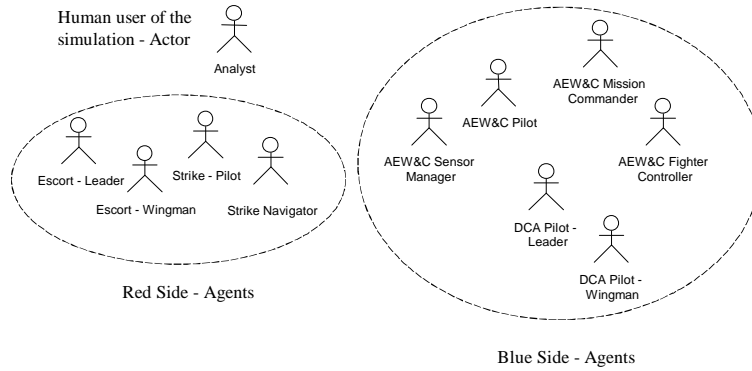


Figure 4 Actors and Agents - First Iteration

The roles, responsibilities, and a brief description of the command control and communication relationships are included for each of these actors. Five of these brief descriptions are shown below in Table 3.

Analyst	Actor: The person who will use the finished system for operations research studies. Specifies the starting conditions, the data to be recorded, and other parameters associated with the conduct of the study.
Fighter Controller	Agent: The crew member(s) on board the AEW&C aircraft responsible for scrambling, vectoring, and controlling fighter aircraft.
DCA Leader	Agent: The fighter pilot responsible for patrolling a region. Conducts mission in conjunction with the wingman. Commands the pair of aircraft and makes all high level decisions. Under certain circumstances will be controlled by advice from the AEW&C.
Strike Pilot	Agent: The strike aircraft crew member (this particular aircraft has a crew of two) responsible for flying the aircraft and ensuring the safe conduct of the mission.
Strike Navigator	Agent: The strike aircraft crew member responsible for navigation, monitoring many of the on-board systems, and for weapons. The navigator and pilot have a close working relationship coordinating and communicating activities.

Table 3 Actors and Agents - Description

4.3 Identifying Use Cases

The first step in identifying the use cases to be included in the use case diagrams is the preparation of primary scenarios. Primary scenarios can now be assembled from an inspection of the operational scenario. A primary scenario is prepared for each actor. Some examples of these scenarios are provided below.

4.3.1 Primary Scenarios

Primary Scenarios:

Fighter controller:

1. Monitors radar for detection of tracks
2. Identifies tracks
3. Vectors fighters toward enemy aircraft
4. Provides advice about the position and action of the enemy aircraft
5. Vectors the fighters home after enemy aircraft have been successfully intercepted

DCA Leader

1. Flies a combat air patrol (CAP) searching for enemy aircraft
2. Upon detecting an enemy aircraft (or upon advice from the AEW&C) will vector toward the enemy aircraft
3. Conducts an intercept of the aircraft
4. Returns to base

Strike Pilot

1. Flies the aircraft along a predetermined set of waypoints.
2. At waypoint Alpha descends to 500 feet
3. Increases speed to 560 knots
4. At waypoint Charlie altitude is decreased to 250 feet and the bombing run is commenced
5. After the target is successfully bombed waypoints are flow to return to base

Strike Navigator

1. Determines when waypoints are reached
2. Launches the weapon

A review of the actors was made and a decision to aggregate the Strike Pilot and Strike Navigator into a single agent (Striker) was taken. This change creates an agent that subsumes the functionality of the pilot and the navigator. This decision was taken because of the close linking between the two agents and because of the relative inactivity of the navigator. For functional specification it is reasonable to treat them as a single agent. The initial assumption of one actor/agent for each human has been revisited and modified. Further modifications to the numbers and types of agents will be made during the design of the system. Decisions taken at analysis time are to provide for a useful set of agents for specifying the required functionality. The implemented system need not implement this.

From these primary scenarios an initial iteration of the use case diagrams is conducted. Use case diagrams are developed in a rather adhoc fashion. Actors are placed on the diagrams and functionality associated with them is added. Where agents cooperate, communicate, or in some way interact to provide a particular functionality two or more agents share a single diagram. The use case diagram shown in Figure 5 is for the fighter controller and the DCA leader that results from an analysis of the primary scenarios. This is may not be the only diagram that features these agents but it a useful one for indicating a certain set of functionality. As many diagrams as are necessary to specify the high level behaviours in a useful way should be developed.

This initial use case diagram is only the first iteration and will be extended and detailed greatly when the secondary scenarios are considered.

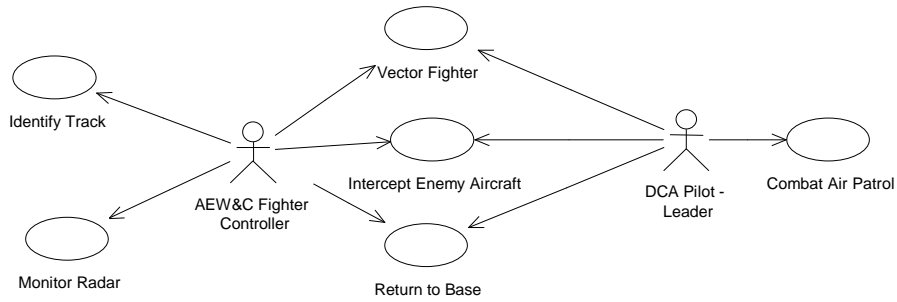


Figure 5 Fighter Controller and DCA Pilot Use Case Diagram

4.3.2 Secondary Scenarios

The secondary scenarios allow for the consideration of possible eventualities that exist outside of the “everything goes right” scenario. If everything goes “as planned” in an air combat scenario there may be very little cooperation or interaction between agents. The mission unfolds with pre-briefed tactics being followed to achieve the mission goal. When unexpected events necessitate dynamic replanning and coordinated action the scenarios will tend to involve more agents. The secondary scenarios typically capture more of these multi-agent interactions. The scenario below illustrates this with an example that necessitates close cooperation between the fighter controller and the lead fighter pilot.

Secondary Scenario: During the intercept of the enemy aircraft by the DCA the enemy escorts obtain the advantage and acquire a radar lock upon the DCA and launch missiles. This eventuality requires a radar-lock/missile evasion manoeuvre that can be assisted by the fighter controller.

1. DCA leader reports that s/he is locked by radar “DCA LEADER SPIKED”
2. Fighter controller advises a heading to fly to break the radar lock “Break left, heading 230”.
3. The DCA leader commences a high *g* turn to the left to come around to heading 230.
4. The DCA leader loses contact with the wingman and the enemy aircraft.
5. The DCA leader successfully defeats the missile.
6. The fighter controller advises about the position of the wingman, the position of the enemy fighters and the vector to fly to rejoin the combat.

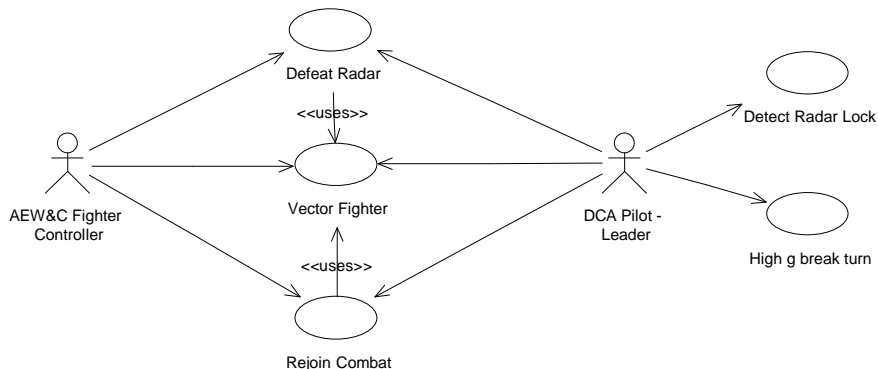


Figure 6 Use Case diagram from secondary scenario

4.4 Documenting Use Cases

Once the use case diagrams have been completed with the input from the secondary scenarios the individual use cases can be documented. Section 3.3 described the range of techniques available for documenting use cases. The documentation for one of the use cases, *defeat radar*, is given below.

Use Case Name: Defeat Radar
Description: When an aircraft launches a radar guided missile the launch aircraft's radar must lock to the target. The pilot of the target aircraft – in conjunction with advice from a fighter controller – can perform a series of manoeuvres designed to defeat the radar and the missile.
Agents: Pilot, Fighter Controller
Actors: None
Use Case Associations: uses Vector Fighter
Environment: Fighter aircraft, radios, missile, radar, radar-warning-receiver, AEW&C
Pre-condition: An enemy has a radar lock on an aircraft
Flow of events:

1. The aircraft pilot informs the AEW&C fighter controller that a radar is locked on.
2. The fighter pilot commences a high g turn. During this turn the fighter will lose contact with the enemy aircraft and will have no radar track information about the enemy aircraft action.
3. The fighter controller calculates a heading for the fighter to fly and provides updates to the fighter pilot about the location of the enemy aircraft.
4. The fighter pilot makes heading adjustments based on advice from the fighter controller
5. The radar lock is lost

Alternative Flow 1.

1. If a missile is known to have been launched and the missile has had time to get close to its target missile evasion manoeuvre is commenced

Post condition: The enemy radar has lost lock and the missile is defeated

Additional information can be provided in the form of an activity diagram (see Figure 7). In the activity diagram below agent interactions are represented across the swim-lane in a manner similar to that proposed by Odell et al. [9].

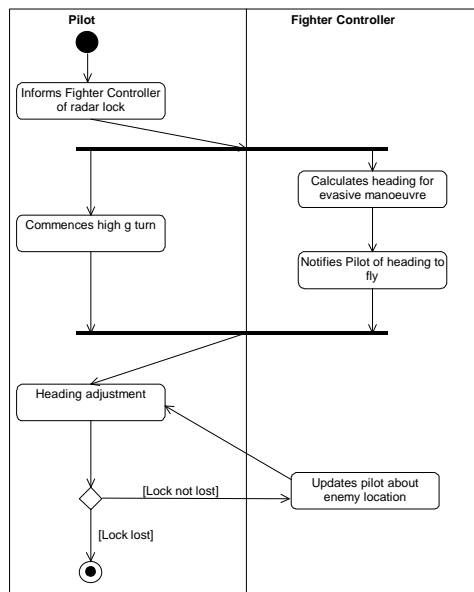


Figure 7 Activity Diagram with “Swim Lanes” for Defeat Radar Use Case

5 Discussion and Future Work

5.1 Note about the BDI agent model

The beliefs-desires-intentions (BDI) rational agent is the model of choice for all of our agent systems development. This model is computationally implemented by languages such as dMARS [17] and JACK [18]. A detailed account of the syntax of these languages is beyond the scope of this paper but both have language constructs that closely map to three items that detail a use case.

	Pre-Condition	Flow of events	Post condition
DMARS	Context Condition	Plan body	! (achieved the goal)
JACK	Context()	Body()	Post event

Table 4 BDI Agent Language properties

Furthermore, JACK has another language feature – capabilities – that map closely to the very notion of a use case. Details of the utility of these language properties in relation to UML and use case analysis will be addressed in a future paper.

5.2 Human-in-the-loop Variant

Research into the interaction of humans and agents as team-members required the construction a human-in-the-loop variant of the above system. This version required an interface to allow a human operator to play the part of a fighter controller. This particular variant of the BattleModel required no further use case analysis. Conceptually the new system was simply the changing of the fighter controller from an agent to an actor. The underlying use case analysis and documentation remained unchanged and was used to specify the requirements for the GUI for the human fighter controller.

5.3 Conclusions

The utilisation of a use-case analysis for the specification of functional requirements for a multi-agent system, has been discussed. Future papers will document further extensions to the UML for the design and implementation of a BDI multi-agent system.

The case-study presented here is not a complex one by AOD's standards. There are fairly few agents and the scenarios require minimal cooperation. Furthermore, the agents themselves do not exhibit the complexity of reasoning of other AOD systems. Not all agent development within AOD makes use of a use case analysis documented with UML. Several techniques have been tried and tested. The use case and UML

approach presented here has been successful on small projects. Planned operational use on large sophisticated developments will test this further.

Acknowledgments

The authors would like to thank Dr. Simon Goss, Mario Selvestrel, Martin Cross, Graeme Murray, Dr. Gary Kemister, Ian Lloyd, Dr Adrian Pearce, Dr. Sam Waugh, and Dr. Peter Wallis for their advice, testing of ideas, and support of the research and development.

References

1. McIlroy, D., *et al.* *Air Defence Operational Analysis Using the SWARM Model*. in *Asia Pacific Operations Research Symposium*. 1997.
2. McIlroy, D. and C. Heinze. *Advanced Operational Reasoning for Hornet Tactics Analysis*. in *The Air Warfare Studies Coordination Group Conference (AWSCG 96)*. 1996.
3. Heinze, C., B. Smith, and M. Cross. *Thinking Quickly: Agents for Modeling Air Warfare*. in *Proceedings of the Australian Joint Conference on Artificial Intelligence AI '98*. 1998. Brisbane, Australia.
4. Tidhar, G., C. Heinze, and M. Selvestrel, *Flying Together: Modelling Air Mission Teams*. *Applied Intelligence*, 1998. **8**(3): p. 195-218.
5. Tidhar, G., *et al.* *Using Intelligent Agents in Military Simulation or "Using Agents Intelligently"*. in *Eleventh Innovative Applications of Artificial Intelligence Conference. Deployed Application Case Study Paper*. 1999. Orlando, Florida.
6. Kinny, D., M. Georgeff, and A. Rao, *A methodology and modelling technique for systems of BDI agents*, . 1996, Australian Artificial Intelligence Institute: Melbourne, Australia.
7. Fowler, M., *UML Distilled - Applying the Standard Object Modeling Language*. Object Technology Series, ed. J. Booch, Rumbaugh. 1998: Addison Wesley.
8. Odell, J., *Engineering Artifacts for Multi-Agent Systems*, . 1999: ERIM CEC.
9. Odell, J. *Representing Agent Interaction Protocols in UML*. in *AAAI Agents Conference*. 2000.
10. Burmeister, B., *Models and Methodology for Agent Oriented Analysis and Design*, . 1996.
11. Bauer, B., *Extending UML for the Specification of Interaction Protocols*, . 1999: Submitted to the 6th call for Proposal of FIPA.
12. Rao, A.S. and M.P. Georgeff, *Modeling rational agents within a bdi-architecture*, . 1991, Australian Artificial Intelligence Institute: Melbourne, Australia.
13. Woodcock, J. and J. Davies, *Using Z*: Prentice Hall.
14. Schneider, G. and J. Winters, *Applying Use Cases - A Practical Guide*. Object Technology Series, ed. J. Booch, Rumbaugh. 1998: Addison Wesley.
15. Rosenberg, D. and K. Scott, *Use Case Driven Object Modeling with UML*. Object Technology Series, ed. J. Booch, Rumbaugh. 1999: Addison Wesley.
16. Wooldridge, M. and N. Jennings, *Agent Theories, Architectures, and Languages: a Survey*, in *Intelligent Agents*, W.a. Jennings, Editor. 1995, Springer Verlag: Berlin. p. 1-22.
17. d'Inverno, M., *et al.* *A formal specification of dMARS*. in *Fourth International Workshop on Agent Theories, Architectures and Languages (ATAL '98)*. 1998: Springer Verlag.
18. Busetta, P., *et al.*, *JACK Intelligent Agents - Components for Intelligent Agents in JAVA*, in *Agent Link Newsletter*. 1999.