Agent Oriented Modelling of Tactical Decision Making

Rick Evertsz, John Thangarajah and Nitin Yadav
School of Computer Science and IT, RMIT University, Melbourne, Australia
{firstname.lastname}@rmit.edu.au

Thanh Chi Ly
DSTO, Perth, Australia
thanh.ly@dsto.defence.gov.au

ABSTRACT

A key requirement in military simulation is to have *executable* models of tactical decision-making. Such models are used to simulate the behaviour of human entities such as submarine commanders, fighter pilots and infantry, with a view to producing realistic predictions about tactical outcomes. Tactics specify the means of achieving mission objectives, and should capture both reactive and deliberative behaviour. The lack of a methodology and supporting tools for designing computer-based models of tactics makes them difficult to create, maintain and reuse, and this is now a significant problem in military simulation domains. To address this, we have developed TDF (Tactics Development Framework), a tactics modelling methodology and tool based on the BDI (Beliefs, Desires, Intentions) paradigm, that supports agent-oriented structural modelling of tactics and related artefacts including missions, storylines, goals and plans. The methodology was initially assessed by analysts in the undersea warfare domain, and was subsequently evaluated using a simple scenario in the autonomous unmanned aerial vehicles domain. The latter evaluation involved a comparison with UML designs, indicating that our methodology provides significant benefits to those building and maintaining models of tactical decision-making.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*methodologies*

Keywords

AOSE methodology; multiagent system; agent design models; cognitive modelling

1. INTRODUCTION

Since the earliest days of the computer simulation of military problems, there has been a need to model tactical decision-making. Tactics are a means of achieving a military objective in an adversarial context. Applications of tactics modelling can be simulation-based, e.g. training [13], or situated in the real world, e.g. UAVs (Unmanned Aerial Vehicles). In such applications, the tactics models are used to drive the behaviour of the entities, for example, in live fire training scenarios [12].

Early on, procedural languages such as Fortran were used to model tactics, for example, in the PACAUS air combat simulation platform. However, frustration with this approach led to a successful early application of the BDI (Beliefs, Desires, Intentions) agent programming language, dMARS, to improve the modelling of air combat tactics [28]. TacAir Soar [24] is another successful AI-based approach to tactics modelling.

Despite the improvement in modelling effectiveness offered by BDI languages and cognitive modelling approaches such as Soar, major problems remain. Models of tactics typically involve a mix of reactive and deliberative decision-making with sudden context switches in response to environmental change. This can make the implementation difficult to understand, debug and maintain, particularly when there is a team maintaining the code base.

Over the last 20 years, our experience, working with groups who model military tactics, has revealed a recurrent theme: model reuse is problematic, particularly when a model is shared across team members. It is not unusual for an analyst to implement a model from scratch rather than try to understand and reuse another's model.

Our current user community of military analysts has over 15 years of experience in agent-based simulation, working closely with military SMEs (Subject Matter Experts). They have been using paper-based workflow diagrams and UML to design tactics models for Undersea Warfare (USW) before implementing them in JACK [38], a Java-based agent programming language. As their code base has grown in size, this approach has not scaled well. The lack of an appropriate design methodology (and tools) to capture the tactics makes this task almost impossible for large systems.

The objective of our research is to develop and evaluate a methodology and tool that supports the demands of modelling tactical decision-making. We have focused on supporting the design of tactics that must work effectively in dynamic domains requiring the application of sophisticated, decision-making, such as that exhibited by human tacticians, e.g. submariners or fighter pilots. Effective performance in such domains requires capabilities such as the balancing of reactivity with proactivity. The entity must be goal directed, but must also be able to switch focus when the environment changes in an important way, or when it discovers that one of the assumptions underlying its current tactical approach is invalid. It may also need to coordinate its activities with peers who are working towards

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

the same goal. It has been argued that these capabilities, namely autonomy, reactivity, proactivity and social ability, are characteristic of agent-based systems [39], and that they are not well supported by most programming paradigms. With this in mind, we have adopted an AOSE (Agent Oriented Software Engineering) approach to the design of tactical decision-making systems.

In developing our application, TDF (Tactics Development Framework), we have extended the Prometheus BDI agent design methodology [31] with explicit support for modelling tactics. This was a natural choice among the more popular BDI agent design methodologies [15] because the supporting Prometheus Design Tool (PDT) [30] generates JACK code directly, a requirement of our user community which has been working with JACK for many years.

We present an innovative tool and methodology for *agent-based design*, rather than an *agent-based implementation* per se. TDF supports the design of agent-based tactical models that ultimately result in a separate, agent-based application that uses those models to generate behaviour.

This paper makes several contributions: (i) the state of the art in AOSE methodologies is advanced by extending, the proven Prometheus methodology with missions, a wider range of goal control structures, plan diagrams and tactics design patterns (Section 3); (ii) these new design artefacts are implemented in a new tool, that builds on the Prometheus Design Tool; and (iii) it presents a preliminary evaluation comprising a pilot study and a separate assessment by USW analysts indicating that the methodology and tool will provide significant benefits to tactics design and development workflow (Section 4).

2. BACKGROUND

In this section we present some background to the methodology. We begin with a brief overview of BDI agents and how tactics relate to them. We also describe the TACSIM tactical simulator and the USW domain, which was the initial target of our methodology and tool. Finally, we present some related work on modelling tactics.

2.1 BDI Agents and AOSE

The BDI agent paradigm [34] is a popular approach to developing multi-agent systems in which agents are modelled using mental constructs such as beliefs, goals and plans.

Briefly, a BDI agent system has a library of programmer-defined plans that are utilised to achieve the goals of the agents. Often, there are multiple plans for achieving a particular goal, and the choice of plan depends on the agent's current situation. For example, a goal to `Return to Base` could be achieved by one of two plans: `Regular Route Plan` and `Concealed Route Plan`, depending on whether the enemy has been detected or not.

When there are multiple plans to achieve the goal, if one plan fails then another is attempted, if applicable in the current situation. This built-in failure recovery is an important aspect of BDI agent systems.

A plan is composed of actions and (sub)goals, where actions are considered as atomic execution statements (e.g. `fire torpedo`) and the (sub)goals in turn are handled by other plans. This naturally gives rise to a goal/plan tree that describes the possible means of achieving the top-level goals. This can also be seen as a description of the possible behaviours of the agent system and indeed in AOSE methodologies these goal/plan trees are a way of describing behaviour models.

AOSE is concerned with how to specify, design, implement, validate and maintain agent-based systems. There are a number of well established AOSE methodologies, including ANEMONA [3], ASPECS [8], Gaia [40], INGENIAS [32] O-MaSE [10], PASSI [7], Prometheus [31], ROADMAP [22] and Tropos [4]. Although there are important differences between these methodologies [9], their common agent-based focus means that there is a large area of overlap, for example, they all facilitate the decomposition of the system into functionally distinct components. Nevertheless, during our analysis of the requirements for modelling tactical decision making, we have identified a number of areas for improvement. These relate to *high-level tasking* (termed 'missions'); *what* the tactics achieve ('goals'); and *how* they achieve their goals (represented as 'plan diagrams').

2.2 Definition of Tactics

In the preceding discussion, we referred to tactics and tactical decision making. The Oxford English Dictionary defines tactics as: *"The art or science of deploying military or naval forces in order of battle, and of performing warlike evolutions and manoeuvres".*

Thus, tactics are adversarial in nature. However, we have adopted a less restrictive definition that focuses on their flexibility:

A tactic is a specification of responsive, goal-directed behaviour.

From this perspective, tactical decision making seeks to achieve an objective in a situation where the system may have to respond to unexpected change. The notion of *responsiveness* is what defines a tactic as something more specific, yet more flexible, than a general behaviour specification. For example, a behaviour specification of how to follow a flight plan to a destination would not be regarded as a tactic. However, a specification of how to navigate to a destination, in a way that can avoid unexpected adverse weather, is indeed tactical. An effective tactical decision-making system must do more than simply blindly execute a procedure; it must be able to respond in a timely manner to events that interfere with the achievement of its goals.

2.3 TACSIM

Developed by DSTO (Defence Science and Technology Organisation), TACSIM supports the modelling of the USW domain and generates quantitative predictions of how proposed capability will impact submarine performance and operational effectiveness. TACSIM generates its predictions by combining JACK-based USW tactics with discrete event simulation to enable Monte Carlo runs. Monte Carlo simulation typically involves many thousands of runs.

2.4 The Undersea Warfare Domain

Although our methodology and tool can be applied to any domain, we present a brief overview of the USW domain because this provides a better understanding of the motivation for the methodology.

The USW domain is distinguished by the paucity of information about the current situation. A submarine commander's knowledge of the tactical situation is time consuming

to acquire, very limited and is sometimes highly uncertain. This severely impedes the decision-making process - much of a commander's tactical repertoire is concerned with building situation awareness whilst not being detected. This places constraints on his use of sensors and the types of manoeuvre available.

On a submarine, sonar is the most important class of sensor. Due to the requirement to remain concealed until it is time to attack, a submarine commander mostly relies on passive rather than active sonar, because the latter could reveal his presence. The undersea environment is noisy and heterogeneous (sound may not travel in straight lines). This makes the data difficult to interpret and the submarine crew must employ sophisticated techniques to improve reliability, for example, Target Motion Analysis [6].

Submarines have a wide range of actions available in support of their tactics, including course, speed and depth changes; deploying countermeasures; firing torpedoes; and using active sonar. However, it is difficult for the uninitiated to appreciate how such seemingly simple actions can contribute to a rich repertoire of USW tactics.

In contrast to other military domains, published studies of human decision-making in USW are few and far between because tactics are kept secret. Studies of decision-making in submarine-related tasks have focused on the biases and limitations of the human cognitive system. The only accessible study of submarine commander decision-making is Project Nemo [19]. Analysis revealed a shallow, adaptive goal structure characteristic of schema instantiation [37]. In other words, the commanders had built up a repertoire of situation descriptions coupled with tactics that worked well in those situations, rather than performing a search through a larger problem space. This approach to decision-making is a trademark of expert problem solving [23] and is well represented by the BDI paradigm which tends to focus on pre-compiled recipes for problem solving, rather than deriving the solution from first principles.

2.5 Related Work

In general, software engineering is an error-prone process. In an effort to reduce errors, software engineers have adopted a wide range of approaches ranging from Domain Specific Languages that offer an appropriate level of abstraction, to software development methodologies that impose a process that reduces the opportunity for error. Developing sound AOSE practices is vital for tactics modelling that is applied to dynamic and complex domains. Nevertheless, to date there are almost no published studies of software engineering approaches to modelling military tactics ([36] is a notable exception). Although there have been a number of studies of human behaviour modelling for military simulation, they have all focused on tools, techniques and research themes, rather than the software engineering aspects, cf. [2]. UML has been suggested as a method for modelling military behaviour [26], but our user community has applied UML to USW tactics modelling for many years, and has found it wanting.

A key feature of our methodology is the provision of design patterns for tactics. In the realm of software engineering, design patterns were first proposed for object-oriented programming [16] to foster software reuse. Subsequently, the notion of design pattern was successfully applied at the agent architecture level, for example, in the early days of the development of the Java mobile agent platform, Aglets [25]. This successful mapping is not surprising, because despite differences such as autonomy, like objects, an agent encapsulates internal state and offers a set of interaction methods, cf. [1].

However, the internal functioning of agents that embody sophisticated military tactics, is very different to that of typical objects. It has been argued that tactics require flexible, goal-oriented execution that can include concurrent tasks [36], necessitating the development of a different class of design pattern. Hence, in this work we directed our effort towards a methodology that supports the BDI model of agents.

As we describe our framework ahead, we will outline further related work relevant to the different aspects of our framework.

3. OVERVIEW OF TDF

This section presents TDF in the context of a representative USW vignette developed in conjunction with our user community. This vignette is based upon a fictional scenario adapted from the computer game "Dangerous Waters"[1], and has been chosen because it does not embody classified tactical information. The confines of this paper do not allow a full exposition of the design, so we have focused on those aspects that illustrate the novel contributions of TDF.

3.1 Outline of the Approach

Much of the difficulty in agent-oriented design lies in determining how best to decompose the proposed application into functionally distinct components, and consequently this is where Prometheus and other AOSE methodologies have been focused. However, in domains such as USW that involve sophisticated tactical decision-making, a large part of the modelling problem is concerned with individual agent tactics, rather than how groups of agents interrelate to one another and their environment. This necessitates a focus on *how* an agent achieves its objectives, and largely pertains to the decomposition of tactical goals and their mapping to procedures. Our tool augments PDT with a high-level, diagrammatic procedural representation of tactics that is also a help during knowledge acquisition; tactics are goal-oriented and procedural in nature, and it is beneficial for SMEs to be able to critique the diagrams at an early stage, in order to verify that the analyst has correctly interpreted their description of the tactics.

Typically, agent-oriented systems embody a fairly static system architecture; the number of agent instances varies, but the behaviour of those agents is largely defined by the time the application is deployed. In contrast, tactics models are built for a particular training scenario or study, but the models will be frequently augmented and/or changed to handle new scenarios and studies. This requires a dynamic development cycle and an emphasis on the design and maintenance of sets of tactics that can change from study to study. TDF supports reuse through the introduction of 'tactics design patterns' that summarise the attributes of a tactic, including its main objective, sub-goals and methods (plan diagrams). Tactics design patterns offer an idiomatic, BDI method of specifying behaviour at a level of abstraction that makes the intent clear and facilitates communication

[1] `http://en.wikipedia.org/wiki/Dangerous_Waters`

between analysts. Note that the tool does not support the modelling of team structures, however, this is planned for the future.

Although the development of TDF was motivated by the need to support the design and maintenance of USW tactics, we believe that the approach has wider application to multi-agent systems where modification and reuse of behaviour specifications is important. Further, the enhancements to the Prometheus methodology with respect to goal control structures and plan diagrams could be applied to other AOSE methodologies that follow a BDI agent model.

3.2 Main TDF Methodology Stages

In keeping with the Prometheus methodology, TDF partitions tactics modelling into 3 main stages:

- **System Specification.** Identification of system-level artefacts, namely missions, goals, storylines, percepts, actions, data, actors and roles.
- **Architectural Design.** Specification of the internals of the system, including the different agent types (by grouping roles), the interactions between the agents (via protocols), and messages.
- **Detailed Design.** Definition of the internals of the agents, i.e. plan diagrams, internal events, messages sent and received, and data that is used by the agent.

Tactics design patterns combine artefacts that span the three design stages, encapsulating them in a way that yields a reusable template.

3.3 Missions

Military simulation studies are usually motivated by a vignette - a general description of a situation to be used for the purposes of analysis. A mission is more specific than a vignette and is intended to be executed. In TDF, each mission specifies an expected sequence of interactions between the system and its environment, and is therefore a central driver in the tactics design process. Tactics are the means of achieving the mission objective.

A TDF mission comprises the following fields:

- **Objective.** The primary goal, e.g. "Destroy enemy submarine"
- **Secondary Objectives.** Secondary goals to be achieved if an opportunity presents itself, e.g. "Identify other maritime vessels"
- **Mission Statement.** A description of the mission, e.g. "BLUE STG (Surface Task Group) is tasked with destroying the RED base. We expect RED to have dispatched a submarine to intercept. RED's best tactic is to intercept STG in the Strait, a significant choke point. If the RED submarine reaches the Strait, it will be a threat to STG. Make haste to the Strait, locate and destroy the RED submarine."
- **Operational Constraints.** States of the world to be maintained for the duration of the mission, e.g. "Stealth is paramount. Do NOT take advantage of any opportunities to engage surface vessels, regardless of any hostile action on their part."
- **Risks.** Known risks and dangers that can help prompt the design of tactics to handle unexpected situations, e.g. "The number of RED submarines is unknown. A RED submarine has been detected, but there may be other RED submarines en route or already lying in wait."
- **Opportunities.** Situations that could be exploited, e.g. "The RED submarine(s) will have to move fast and so will be more detectable than usual."
- **Storylines.** Alternative ways that the mission might play out, e.g. "Navigate to Area of Operation, Search for Target, Classify Contact, Attack Target, Confirm Target Destroyed."
- **Data.** Mission-specific information, e.g. "mission route, map, danger areas, enemy submarine class, undersea contour map"

3.4 Storylines, Actors, Roles, Percepts and Actions

Scenarios, actors, roles, percepts and actions are a standard part of Prometheus, but the scenario concept has been renamed 'storyline' in TDF because the term 'scenario' has a number of alternate meanings in military vocabulary. Storylines map out key sub-sequences of tactics-related activity that can occur as part of a mission. A storyline can comprise a sequence of goals, actions, percepts and sub-storylines. As each storyline is developed, the need for particular tactical approaches will become apparent together with supporting goals, actions and percepts. Storyline development is complete when there are enough storylines to describe the different ways that each mission can play out.

To illustrate, the `Incoming Torpedo` storyline has the following steps: `Torpedo Detected` (percept), `Estimate Bearing And Distance` (goal), `Deploy Countermeasures` (goal), `Evade Torpedo` (goal).

As storylines are mapped out, it soon becomes apparent that the goals initiate particular functions within the overall system. These functional relationships are expressed as roles that encompass one or more goals. Percepts to be handled and actions to be produced by the role are grouped within that role. Example roles in this USW vignette include `Navigation`, `Sonar Interpretation` and `Countermeasures Handling`. Roles ultimately encapsulate agent functionality and are used to guide the decomposition of the system into agents in the Architectural Design stage of the methodology.

Actors are used to model entities that generate percepts and/or are affected by system actions. From a modelling perspective, actors are opaque entities. In contrast to agents, actors are external to the system and their internal structure is not modelled - they are effectively black boxes that produce behaviour and respond to system actions. In the USW domain, actors are used to represent the functional components of the submarine, for example the `Countermeasures` and `Sensor` subsystems.

3.5 Goal Structures

Specifying missions and their storylines prompts the designer to think about the goals that need to be achieved during mission execution. Consideration of top-level goals will lead to the derivation of sub-goals that have to be satisfied on the way to achieving the overall mission objective. These goal/sub-goal relationships are expressed as a hierarchical 'goal structure'. Because tactics are inherently goal oriented, the goal structures form the scaffolding around which the tactics are ultimately built, and offer a high-level description of a tactic's functional decomposition.

During our analysis of the USW domain, it became apparent that previous AOSE methodologies do not provide a sufficiently rich representation of goal *control* structures. Tropos[4] provides a wider range of goal *types* that are well suited to goal-oriented requirements analysis, such as *soft goals*, but these do not address the goal-oriented control structures required to express tactics at a high level of abstraction. Ideally, a goal-based representation of tactical decision making should encode *reactive/deliberative* goal considerations, such as the conditions under which a goal should be suspended or resumed. These goal conditions and inter-goal relationships are important to tactical decision making. Goal-related attributes are usually implicit, hidden deep in the system's implementation. The objective in TDF is to make these hidden dependencies explicit at the design level, so that the designs are a more accurate reflection of the desired system behaviour, thereby promoting user comprehension and the potential for design reuse and sharing. TDF extends the `and/or` connectives of PDT with the following control structures:

- **Goal Ordering.** Goals are implicitly ordered from left to right; this reflects the fact that tactics typically involve a sequence of goals that have to be achieved one after another. This default ordering can be overridden to express cases where serial goal order is unimportant.
- **Conditional.** These have a *guard* that has to be true for the goal to be tried. If the guard is not true, the goal is skipped. Figure 1 shows a conditional goal `Evade Torpedo` with the guard: `no countermeasures remaining`. The `Handle Incoming Torpedo` goal is annotated with a `suspend/resume` condition that handles the situation where the torpedo veers off course but may reacquire the target.
- **Concurrent.** Sibling sub-goals that are to be adopted concurrently and independently. The parent goal will only succeed once the concurrent goals have been successfully achieved. See `Deploy Countermeasures` in Figure 1.
- **Anonymous node.** Used to partition the goal structure into sub-trees that have a different logical relationship to one another. See Figure 1 where `Deploy Countermeasures` and `Evade Torpedo` have an `and` relationship within the overall `or` level of the graph.
- **Asynchronous.** Operationally, the parent goal does not wait for any asynchronous child goals to succeed. Useful in cases where a sequence of tasks needs to be initiated without waiting for them to complete successfully.
- **Maintenance.** Expressed as a guard on a goal that spans all child goals. If the guard becomes untrue, the system will attempt to make it true again. This behaviour is an important component of tactics that must deal with a world that changes unexpectedly. Note that this construct reflects the reactive behaviour of maintenance goals but not the proactive behaviour, as described in [11].
- **Preserve.** Expressed with a guard labelled 'while', the sub-goal is pursued as long as the guard is true. If the guard becomes false, the goal is dropped.

Figure 1 illustrates the goal structure for `Handle Incoming Torpedo`, which derives from the `Incoming Torpedo` storyline outlined in Section 3.4.

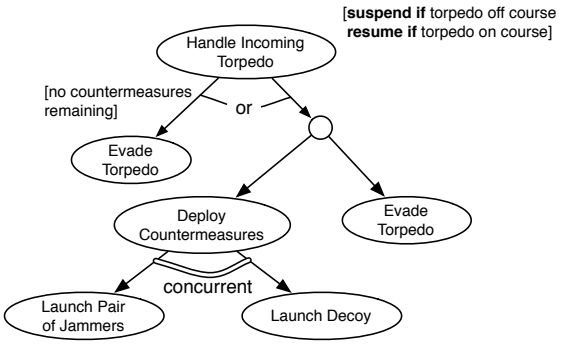

Figure 1: Handle Incoming Torpedo Goal Structure

The first sub-goal to be evaluated is `Evade Torpedo` provided that the guard `no countermeasures remaining` is true (a 'conditional' goal). If the guard is false, then the 'anonymous' node (empty circle) is tried because of the `or` relationship.

The anonymous node has two sub-goals (with an implicit `and` relationship). To `Deploy Countermeasures` the submarine concurrently tries `Launch Pair of Jammers` and `Launch Decoy`, denoted by the 'concurrent' label. Once both sub-goals have been achieved, it tries to achieve `Evade Torpedo`.

Note that to fully utilise the various control structures mentioned above, in practice the agent implementation language should provide the necessary infrastructure to support them. Our target agent system is JACK [38] and we have specified code generation for mapping to JACK language elements, such as the use of the `@parallel` and `@maintain`, to support the `concurrent` and `preserve` control structures, respectively. The details of this mapping are beyond the scope of this paper.

The term 'maintenance goal' has a long history of application in BDI systems. Unfortunately, the term can be misleading. In JACK, a maintenance statement defines a sub-goal to be pursued as long as some condition is true, e.g. "search for target as long as you have sufficient fuel to return to base". However, there is no sense in which anything is maintained, except in the very loose sense of maintaining the pursuit of a goal while some condition holds. A JACK maintenance statement is really a form of guarded action, as expressed by PRS's 'preserve goal' statement [21]. PRS also has a maintain goal that attempts to achieve a state that should have been maintained but was violated. TDF supports the expression of a preserve goal and a maintain goal through the use of the `while` and `maintain` statements, respectively. A goal will be pursued as long as its `while` condition is true; this corresponds to JACK's `@maintain` statement.

3.6 Plan Diagrams

The goal structures prompt the specification of the various ways that the goals can be achieved, expressed as plan diagrams. A plan diagram is a high-level procedural representation that shows the sequence of steps executed by some part of the tactic. Taken together, the collection of plan diagrams procedurally specifies how the whole tactic does what it does.

TDF plans are a level of abstraction above the implementation, and should be viewed as diagrammatic pseudo-code

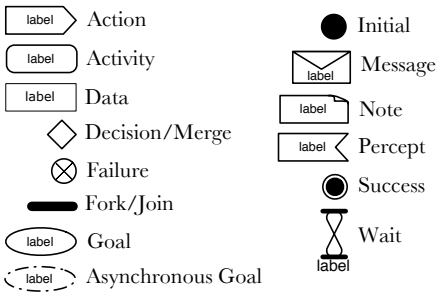

Figure 2: Plan Diagram Node Icons

rather than an executable implementation. A TDF plan specifies the general steps of a procedure without getting bogged down in implementation detail. A plan diagram is an extended implementation of the Prometheus methodology's unimplemented notion of 'process diagram' [31].

There is a long tradition of using diagrams to represent procedures, whether as flow charts [18], Petri Nets [33], recursive transition networks (e.g. PRS [17]), UML activity diagrams [20], Business Process Modeling and Notation (BPMN) [29] or countless variations. Because UML is widely used for diagrammatic software specification, we adopted it as the basis for the representation of plan diagrams in TDF, modifying where necessary to reflect BDI semantics and PDT notational convention. The overriding goal is to foster tactics designs that are easy to understand and modify. The plan diagram representation encourages this by reducing some of the complexity that can be expressed in UML activity diagrams, excluding elements that are not relevant to BDI models, e.g. parameter passing via input/output pins (which is handled in BDI models via event parameters, or via the beliefs database), and exception handler nodes.

Despite the similarities in notation between plan diagrams and UML activity diagrams, the underlying semantics have very little in common due to the fundamental differences between the BDI and objected oriented paradigms. For example, in BDI, each node either succeeds or fails. If it fails, the invocation of the plan fails and the execution engine will try an alternative way of achieving the goal if one is available. Another important difference is that a plan may be suspended if a higher priority task has to be performed; a plan may also be abandoned if certain conditions no longer hold true.

Figure 2 shows the TDF plan diagram node icons. The node types are:

- **Initial.** Plan diagrams have a unique start node, preceded by either a goal, percept or message, as well as an optional guard.
- **Action.** Performed on the agent's environment.
- **Activity.** Denotes a sequence of computational steps.
- **Data.** Data access and update.
- **Decision/Merge.** A decision node represents a conditional choice between options. A merge node transitions to its outgoing arc as soon as one of its incoming arcs completes.
- **Failure.** Terminates the plan with failure.
- **Fork/Join.** A fork node denotes concurrent threads. A join node synchronises its incoming threads.

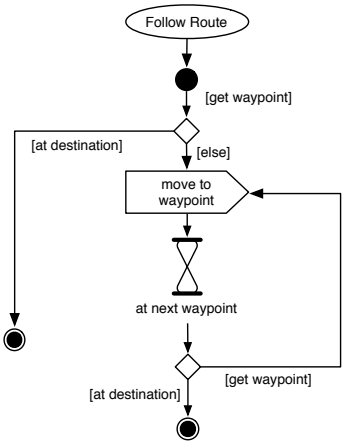

Figure 3: Waypoint Following Plan Diagram

- **Goal.** A goal to be achieved.
- **Asynchronous Goal.** A goal to achieve without waiting for it to succeed.
- **Message.** A message sent to another agent.
- **Note.** For documentation.
- **Percept.** Precedes initial node. Denotes a reactive plan.
- **Success.** Terminates the plan successfully.
- **Wait.** Waits for a condition to become true. Used to express temporal information, such as waiting until a particular time, for an elapsed amount of time, or for a more general condition to become true.

Iteration can be expressed as shown in Figure 3. The plan performs a `move to waypoint` action and waits until the next waypoint has been reached. It loops until the destination has been reached. The label `[get waypoint]` denotes a data lookup.

3.7 Tactics Design Patterns

A key objective for TDF is to offer a high-level representation that supports reuse and sharing by providing the developer with an extensible library of tactics design patterns. Tactics design patterns, termed 'tactics' in TDF, encode general-purpose tactical solutions that can be customised for more specialised applications. A number of approaches to the provision of re-usable design templates have been investigated, particularly in the field of Knowledge-Based Systems. Generic Tasks [5] are a means of representing high-level problem solving in a way that captures the strategy using a vocabulary that is relevant to the task. In a similar vein, Problem-Solving Methods (PSMs) [27] have been proposed as a way of expressing domain-independent, reusable strategies for solving certain classes of problem. A PSM comprises a definition of *what* it achieves, *how* to achieve it and what it *needs* to perform its function. These are termed respectively its 'competence', 'operational specification' and 'requirements/assumptions'. In our approach, the *what* is expressed as the 'objective' and 'outcomes', the *how* as the 'goal structure' and 'plans', and the *needs* as the 'information required'.

The properties of tactics design patterns are listed below. An effort was made to use property names that are intuitive

to analysts and SMEs; for example 'objective' rather than 'top-level goal', and 'goal structure' rather than 'goal tree'.

- **Name.** Every tactic has a unique identifier.
- **Objective.** This is the top-level goal the tactic achieves and captures the proactive nature of tactics, e.g. "Escape Torpedo"
- **Trigger.** A percept that triggers the use of the tactic. This is used to model the reactive nature of a tactic, i.e. one that is triggered by an environmental event rather than by adopting a goal to be achieved. Typically with tactics, the trigger will lead to goal-based decision-making, but this field allows for reactive behaviour without goal deliberation, e.g. "Incoming torpedo detected"
- **Problem Description.** A description of the types of problem the tactic applies to, e.g. "An incoming torpedo has been detected on passive sonar, but our countermeasures are depleted"
- **Solution Description.** A description of how the tactic achieves its objective, e.g. "Having no countermeasures, the only way to evade is to flee. The tactic can also involve firing a torpedo along the bearing of the initial detection before fleeing. This tactic can flee in open water, but can also take advantage of undersea terrain, using it for concealment."
- **Context.** A tactic is only applicable if its context is true, where a context is a boolean test of the state of the world, e.g. "No countermeasures remaining"
- **Outcomes.** A description of how the world will change after the tactic has been successfully executed.
- **Restriction.** Properties of the world that must be maintained for the duration of the tactic's execution.
- **Information Required.** Information that the tactic requires to perform its function, e.g. "undersea contour map, torpedo bearing, torpedo origin, ownship state (speed, position etc.)"
- **Information Updated.** Updates to the agent's beliefs.
- **Goal Structure.** A hierarchical representation of the goals underlying the tactic.
- **Plans.** A collection of references to plan diagrams, e.g. "Track Torpedo, Evade Torpedo, Hide Behind Terrain"
- **Source.** References to source material used to create the tactic, e.g. "Drafted from Doctrine Manual USW-2-10."

In TDF, plans are grouped in terms of the tactics they contribute to, and are represented at an abstract level independent of the particular implementation language. This encourages the analyst to think in terms of high-level 'tactics', and promotes procedural abstraction. Thinking in terms of tactics, rather than more low-level plans, also facilitates merging, reuse and maintenance of tactics sets. The tactics design pattern makes the important attributes of the tactic explicit. For example, if a tactic requires that the submarine remain submerged at all times, this is defined as an explicit `restriction`. When considering merging this tactic with one that involves coming to the surface (an `outcome` of the latter tactic), it is obvious that there is a conflict that needs to be resolved. In our experience, this type of conflict is usually not apparent at the plan level; the surfacing action might be embedded deep down in the goal/plan tree and is likely to be missed when looking at the top-level plans.

3.8 The TDF Tool

The TDF tool has been demonstrated previously [14], and is implemented as an Eclipse plug-in that extends PDT. The tool has desirable features such as type safety, automatic propagation, and code generation. These features play a significant role in ensuring that design artefacts created by the TDF tool are sound, and that the design maps to executable code.

- **Type safety.** To add an entity to a diagram, a user may either select an existing entity from a list or create a new one. This approach not only provides filtered entity-specific lists but also avoids typing errors.
- **Automatic propagation.** Entities that span across multiple editors are automatically propagated. For example, when a new goal is added to the Analysis Overview, it is automatically propagated to the Goal Overview editor.
- **Code generation.** The plugin comes with an extension capable of generating skeleton code for JACK [38] and GORITE [35]. The code generation feature provides a predefined structuring, in the form of JAVA packages, to the system. This facilitates and enforces good software engineering practices. In addition, the code generation is built to allow an iterative design and coding approach. The generated files are marked with areas that are affected by automatic code generation. Any changes outside of these marked areas are preserved when generating code after updating an existing TDF design.

Overall, providing the TDF tool as an Eclipse plugin integrates the design and development of tactics within a unified environment. One of the future directions for the tool is to extend the code generation abilities to cover agent-oriented programming languages other than JACK and GORITE. This will encourage different communities to adopt the TDF methodology for designing tactical decision-making systems. In parallel, we are also working on developing an automatic report generator that will combine the export image functionality with additional information that can be processed from properties defined for various entities.

4. EVALUATION

We have begun an initial evaluation of TDF, comprising informal feedback from our user community and a separate pilot study comparing user comprehension of a TDF design with an equivalent UML one.

4.1 User Community Feedback

As a first step towards evaluating our approach, our user community of USW analysts, applied the proposed methodology to some of their USW tactical simulations which they conduct using TACSIM. Whilst we were not able to observe the resulting models, due to the sensitive nature of the simulations, we were provided with written feedback.

The initial feedback from USW analysts is that TDF is likely to be very useful in the context of Monte Carlo constructive tactical simulations because its high-level, design view of tactics facilitates reasoning about how the agents

will behave at runtime. This is essential in Monte Carlo simulations because they generate a very large number of outcomes, making validation by visual inspection of the implementation impractical. The analyst must clearly understand the scope of the tactics models in order to draw the correct conclusions from the aggregated result of thousands of simulation runs.

Our users commented that TDF's use of tactics design patterns to maintain the link between missions, goals and implementation will facilitate reuse, and that this is vital to their workflow. The tactics design patterns should also help solve search and retrieval problems found in large tactics repositories. As TACSIM is a long-term endeavour, the collection of TDF tactics design patterns is expected to grow over time. Without this software engineering support, retrieval of relevant content will become increasingly difficult.

4.2 Pilot Study

The objective of the pilot study was to investigate whether TDF designs are easier to understand than equivalent UML versions. To this end, a UAV photo reconnaissance scenario was presented to the participants, because the USW domain is complex and generally unfamiliar. In this vignette, the objective of the UAV is to locate a convoy and photograph it. The design presents various search patterns, avoids no-fly zones, and includes the ability to handle unexpected situations such as low cloud, insufficient fuel, and an incoming missile.

Experimental Conditions. Two groups: TDF vs. UML.

Subjects. 10 postgraduate and senior computer science students, all very familiar with UML, were randomly assigned to the two groups, in equal numbers.

Method. In order to minimise the need to learn the mechanics of the TDF and UML tools, the designs were presented as a collection of static diagrams. All sessions were run by the same experimenter. Because the TDF group were unfamiliar with TDF, they were given a 15 minute overview of the notation. Both groups were encouraged to ask questions about any notation they were unsure off. All subjects were allowed unlimited time to peruse the design and could ask questions related to syntax but not the behavioural implications of the design. A 15-item multiple choice questionnaire was completed immediately after examining the design. The questionnaire tested the subjects' understanding of the UAV's behaviour in specific situations addressed by the design. The subjects could consult the design at any time while completing the questionnaire. Time taken to complete the questionnaire was recorded.

Results. Mean questions answered correctly: TDFμ_1 = 82.4%, UMLμ_2 = 66.4%. H_0: $\mu_1 = \mu_2$. H_1: $\mu_1 > \mu_2$ (null hypothesis rejected, $p < 0.025$; one-tailed, independent t-test). There was no significant difference in the time taken to complete the questionnaire between the two groups: μ_1 = 13.4, μ_2 = 13.6 (minutes).

Analysis. This pilot study indicates that TDF's approach to tactics representation is more readily understood than the equivalent UML version. The effect is surprisingly large, given that the designs were deliberately simplified so as not to disadvantage UML. The BDI paradigm allows for a simpler representation of behaviour that can be interrupted, because an executing plan can be interrupted at any time by another, higher priority one. In contrast, potential interruptions must be explicitly represented in UML activity diagrams. If there are more than a few such interruptions, the activity diagram can quickly become very complex. This TDF/BDI advantage was ameliorated by keeping the potential interruptions to just two, and this only occurred in one activity diagram. We suspect that the experimental effect would have been even larger if the design had included a more sophisticated reactive/deliberative mix of behaviour, such as that represented in TDF by maintenance goals.

5. DISCUSSION

This paper introduced TDF, a novel methodology and tool for modelling military tactics, which extends a state of the art agent design methodology with artefacts required for tactics modelling. There is a real need for software engineering support during the development of military tactics models. Current approaches, such as UML or direct implementation, are effective for small sets of tactics, but do not scale well. Of particular concern is the need to reuse and change the mix of tactics from study to study. This leads to a requirement for designs that, at the level of tactics at least, can be easily modified. Current AOSE methodologies have focused on modelling the decomposition of a system into agents and how those agents interact with one another and their environment. Though this approach is effective for building agent-based systems, it falls short when it comes to tactics modelling.

TDF provides support for tactics representation in the form of design patterns that include links to high-level plan diagrams. The provision of conditional goals is key to the design and maintenance of tactics because tactics typically mix deliberative and reactive modes of reasoning. Annotating goals with the conditions under which they are adopted, maintained, dropped or resumed is a key feature of TDF that makes the tactics easier to understand, reuse and modify. This advantage was highlighted by the comparison of TDF designs with UML versions.

The pilot study indicates a significant tactics modelling advantage for TDF vs. UML. It will be worth performing a more extensive evaluation once our user community has used TDF to build larger USW tactics libraries. We also plan to evaluate TDF in other military as well as non-military domains to assess the general applicability of this approach to tactics design.

TDF's high-level diagrammatic view provides an opportunity for SMEs to critique the tactics models; an important part of validation. We are currently extending TDF to focus on knowledge elicitation and have found that the early generation of storylines combined with TDF's conditional goal structures helps with eliciting the more subtle aspects of when goals should be adopted/dropped and how a tactic should respond to changes in the situation.

TDF could also move further in the direction of formalisation of its various design artefacts. However, our experience with PDT users suggests that this can hinder usability. This could be overcome by allowing the expression of a user-preferred mix of formal and informal property values.

ACKNOWLEDGEMENTS

We are grateful to the Defence Science and Technology Organisation (Australia), and the Defence Science Institute (Melbourne, Australia) for the invaluable support given to this project.

REFERENCES

[1] Yariv Aridor and Danny B. Lange. Agent design patterns: elements of agent application design. In *Proceedings of the second international conference on autonomous agents*, pages 108–115. ACM, 1998.

[2] J. Blascovich and C. R. Hartel. *Human behavior in military contexts*. National Academies Press, Washington DC, 2008.

[3] Vicente Botti and Adriana Giret. *ANEMONA: A multi-agent methodology for holonic manufacturing systems*. Springer Publishing Company, 1st edition, 2008.

[4] P. Bresciani, A. Perini, p. Giorgini, F. Giunchiglia, and Mylopoulos.J. Tropos: An agent oriented software development methodology. *AAMAS*, 8(3):203–236, 2004.

[5] Balakrishnan Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE expert*, 1(3):23–30, 1986.

[6] Pedro F. Coll. Target motion analysis from a diesel submarine's perspective. Technical report, DTIC Document, 1994.

[7] Massimo Cossentino. From requirements to code with the passi methodology. *Agent-oriented methodologies*, 3690:79–106, 2005.

[8] Massimo Cossentino, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abderrafiâa Koukam. ASPECS: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 20(2):260–304, 2010.

[9] S.A. DeLoach, L. Padgham, A. Perini, and A. Susi. Using three AOSE toolkits to develop a sample design. *IJAOSE*, 3(4):416–476, 2009.

[10] Scott A. DeLoach and Juan Carlos Garcia-Ojeda. O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. *International Journal of Agent-Oriented Software Engineering*, 4(3):244–280, 2010.

[11] Simon Duff, John Thangarajah, and James Harland. Maintenance goals in intelligent agents. *Computational Intelligence*, 2012.

[12] R. Evertsz, A. Lucas, C. Smith, M. Pedrotti, F.E. Ritter, R. Baker, and P. Burns. Enhanced behavioral realism for live fire targets. In R. S. St. Amant, D. Reitter, and E. W. Stacy, editors, *Proceedings of the 23rd Annual Conference on Behavior Representation in Modeling and Simulation*, Washington DC, 2014. BRIMS Society.

[13] R. Evertsz, M. Pedrotti, and W. Glover. Realistic virtual actors for training in counterterrorism. In *Proceedings of SimTecT*, 2010.

[14] R. Evertsz, J. Thangarajah, N. Yadav, and T. Ly. Tactics development framework (demonstration). In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems*, pages 1639–1640. International Foundation for AAMAS, 2014.

[15] Marie-Pierre Gleizes Federico Bergenti and Franco Zambonelli, editors. *Methodologies and Software Engineering for Agent Systems : The Agent-Oriented Software Engineering Handbook*. Springer, 2004.

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: Elements of reusable object-oriented design*. Addison-Wesley Reading, 1995.

[17] M. P. Georgeff and A. L. Lansky. Development of an expert system for representing procedural knowledge. final report, for nasa ames research center, moffet field, california, usa. *Artificial Intelligence Center, SRI International, Menlo Park, California, USA*, 1985.

[18] Frank Bunker Gilbreth and Lillian Moller Gilbreth. Process charts - first steps in finding the one best way to do work, 1921.

[19] Wayne D. Gray, Susan S. Kirschenbaum, and Brian D. Ehret. The précis of project nemo, phase 1: Subgoaling and subschemas for submariners. In *Nineteenth Annual Conference of the Cognitive Science Society*, pages 283–288, 1997.

[20] Linda Heaton. Unified modeling language (UML): Superstructure specification, v2.0. *Object Management Group, Tech. Rep*, 2005.

[21] François Félix Ingrand, Raja Chatila, Rachid Alami, and Frédérick Robert. PRS: A high level supervision and control language for autonomous mobile robots. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 1, pages 43–49. IEEE, 1996.

[22] Thomas Juan, Adrian Pearce, and Leon Sterling. Roadmap: extending the gaia methodology for complex open systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 3–10. ACM, 2002.

[23] Gary A. Klein. *Sources of power: How people make decisions*. MIT press, 1999.

[24] J. E. Laird, R. M. Jones, O. M. Jones, and P. E. Nielsen. Coordinated behavior of computer generated forces in tacair-soar. In *In Proceedings of the fourth conference on computer generated forces and behavioral representation*. Citeseer, 1994.

[25] Danny B. Lange and Oshima Mitsuru. *Programming and Deploying Java Mobile Agents Aglets*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[26] Nilesh Maltare, Aditya Parasrampuria, and Sachin Patel. Exploiting UML to model military organisation and military behaviour. In *3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, volume 9, pages 461–465. IEEE, 2010.

[27] John McDermott. Preliminary steps toward a taxonomy of problem-solving methods. In *Automating knowledge acquisition for expert systems*, pages 225–256. Springer, 1988.

[28] G. Murray, D. Steuart, D. Appla, D. McIlroy, C. Heinze, M. Cross, A. Chandran, R. Raszka, G. Tidhar, A. Rao, A. Pegler, D. Morley, and P. Busetta. The challenge of whole air mission modelling. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*, Melbourne, Australia, 1995.

[29] OMG, Business Process Modeling Notation. Version 1.0. *OMG Final Adopted Specification, Object Management Group*, 2006.

[30] L. Padgham, J. Thangarajah, and M. Winikoff. Prometheus design tool. In *Proceedings of the 23rd AAAI Conference on AI*, pages 1882–1883, Chicago, USA, 2008. AAAI Press.

[31] L. Padgham and M. Winikoff. *Developing intelligent agent systems: a practical guide*, volume 1. Wiley, 2004.

[32] Juan Pavón and Jorge Gómez-Sanz. Agent oriented software engineering with INGENIAS. In *Multi-Agent Systems and Applications III*, pages 394–403. Springer, 2003.

[33] Carl A. Petri. Communication with automata: Volume 1 supplement 1. Technical report, DTIC Document, 1966.

[34] A.S. Rao and M.P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the first ICMAS (95)*, pages 312–319. San Francisco, 1995.

[35] Ralph Rönnquist. The goal oriented teams (GORITE) framework. In *Programming Multi-Agent Systems*, pages 27–41. Springer, 2008.

[36] Glenn Taylor and Robert E. Wray. Behavior design patterns: Engineering human behavior models. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference*, 2004.

[37] Kurt VanLehn. Problem solving and cognitive skill acquisition. In M. Posner, editor, *Foundations of cognitive science*, pages 526–79. MIT Press, MA, 1989.

[38] M. Winikoff. JACK intelligent agents: An industrial strength platform. *Multi-Agent Programming*, pages 175–193, 2005.

[39] Michael Wooldridge. *An introduction to multiagent systems*. Wiley. com, 2008.

[40] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.