The Journal of Systems and Software 110 (2015) 222–238

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A framework for modelling tactical decision-making in autonomous systems

Rick Evertsz [a], John Thangarajah [a,*], Nitin Yadav [a], Thanh Ly [b]

[a] School of Computer Science & IT, RMIT University, Australia
[b] Defence Science and Technology Organisation, Australia

ARTICLE INFO

Article history:
Received 30 January 2015
Revised 25 August 2015
Accepted 27 August 2015
Available online 7 September 2015

Keywords:
Tactics modelling
Behaviour modelling
Multi-agent systems

ABSTRACT

There is an increasing need for autonomous systems that exhibit effective decision-making in unpredictable environments. However, the design of autonomous decision-making systems presents considerable challenges, particularly when they have to achieve their goals within a dynamic context. Tactics designed to handle unexpected environmental change, or attack by an adversary, must balance the need for reactivity with that of remaining focused on the system's overall goal. The lack of a design methodology and supporting tools for representing tactics makes them difficult to understand, maintain and reuse. This is a significant problem in the design of tactical decision-making systems. We describe a methodology and accompanying tool, TDF (Tactics Development Framework), based on the BDI (Beliefs, Desires, Intentions) paradigm. TDF supports structural modelling of missions, goals, scenarios, input/output, messaging and procedures, and generates skeleton code that reflects the overall design. TDF has been evaluated through comparison with UML, indicating that it provides significant benefits to those building autonomous, tactical decision-making systems.

1. Introduction

Fuelled by recent technological advances, there has been an increased interest in autonomous systems, whether for space exploration (Council, 2012), UUVs (Unmanned Underwater Vehicles) (Huntsberger and Woodward, 2011), or UAVs (Unmanned Aerial Vehicles) (Dopping-Hepenstall, 2013). Full autonomy offers many advantages, including allowing a system to venture into environments where teleoperated systems would perform poorly due to high communications latency, for example, deep space.

Despite the recent technological focus on autonomy, in areas such as mining, agriculture and military applications, the general software engineering community may not be aware of the recent progress the field has experienced. Although the Google self-driving car has perhaps been the most high profile autonomy project, many others are underway. Autonomy is now a major focus of the oil and gas industry, where it promises to allow the exploration of previously inaccessible areas and reduce human exposure to danger (NFA Autonomy Working Group, 2012). Fully autonomous robots are now being used by special forces for live fire training (Evertsz et al., 2014). The UK-led ASTRAEA programme has been running for nine years and is working towards the goal of routine flights of unmanned aircraft systems in all classes of airspace (Mills, 2011).

Different applications require, or are best met by, different levels of autonomy. For example, because of the high power requirements and poor levels of transmission (e.g. radio waves) in the underwater environment, UUVs cannot be remotely controlled unless they are close by. Thus the focus of UUV research and development has been on fully autonomous systems. In other domains, if human oversight is necessary, a semi-autonomous approach, where the system submits to the command authority of a human, is preferable.

The objective of this research is to develop and evaluate a methodology that supports the demands of designing autonomous tactical decision-making systems. Our research is focused on supporting the design of autonomous systems that operate in dynamic domains requiring the application of sophisticated, tactical decision-making, such as that exhibited by human experts, e.g. submariners or fighter pilots. Effective performance in such domains requires capabilities such as the balancing of reactivity with proactivity. The autonomous system must not only 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 the same goal. It has been argued that these capabilities, namely autonomy, reactivity, proactivity and

* Corresponding author. Tel.: +61 399259535.
E-mail addresses: rick.evertsz@rmit.edu.au (R. Evertsz), johthan@gmail.com, john.thangarajah@rmit.edu.au, johnt@rmit.edu.au (J. Thangarajah), nitin.yadav@rmit.edu.au (N. Yadav), thanh.ly@dsto.defence.gov.au (T. Ly).

http://dx.doi.org/10.1016/j.jss.2015.08.046
0164-1212/

social ability, are characteristic of agent-based systems (Wooldridge, 2008), 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 autonomous decision-making systems.

AOSE is concerned with how to specify, design, implement, validate and maintain agent-based systems. There are now many AOSE methodologies; Akbar (Akbari, 2010) lists 75, some dating back to the early 1990s. Of these, a number are still in regular use, including ANEMONA (Botti and Giret, 2008), ASPECS (Cossentino et al., 2010), Gaia (Wooldridge et al., 2000), INGENIAS (Pavón and Gómez-Sanz, 2003) O-MaSE (DeLoach and Garcia-Ojeda, 2010), PASSI (Cossentino, 2005), Prometheus (Padgham and Winikoff, 2004), ROADMAP (Juan et al., 2002) and Tropos (Bresciani et al., 2004). Although there are important differences between these methodologies (DeLoach et al., 2009), 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. Although existing AOSE methodologies can support tactics modelling to some degree, the mapping is not straightforward. Our investigation of the requirements for modelling tactical decision-making revealed a number of areas for improvement. These relate to *high-level tasking* (termed *missions*); *what* the autonomous system can achieve (*goals*); and *how* it can achieve its goals (represented as *plan diagrams*).

During our analysis of the requirements of tactics modelling, it became apparent that previous AOSE methodologies do not provide a sufficiently rich representation of goal *control* structures. Tropos (Bresciani et al., 2004) 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. Our objective 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.

To better address the need for autonomous, tactical decision-making, we have taken the Prometheus methodology as a starting point and have developed TDF (Tactics Development Framework) by adding missions, goal extensions, plan diagrams and tactics design patterns. To this end, the Prometheus BDI agent design methodology has been extended with explicit support for modelling tactics. Among the more popular BDI agent design methodologies (Federico et al., 2004), Prometheus was a natural choice given that PDT (Prometheus Design Tool) generates JACK (Winikoff, 2005) code directly, a requirement of our user community which has been working with JACK for many years. A wide range of extensions/simplifications of Prometheus were considered during the development of TDF. The initial set of extensions/simplifications was informed by the 15 year experience we have had in using Prometheus for agent-based modelling and in using BDI languages such as dMARS (d'Inverno et al., 1998) and JACK (Winikoff, 2005) to model decision-making in domains such as autonomous robots, infantry tactics and air combat. This was refined over the course of the current USW (Undersea Warfare) project, based on requirements and feedback from our USW user community, who were also familiar with JACK and Prometheus.

Under the direction given by our user community of USW analysts, a key objective in developing TDF was to offer a method of specifying tactics in a manner that supports reuse and sharing. These *tactics design patterns* specify *what* the tactic does, *how* it does it and what it *needs* to achieve its objective. In software engineering, design patterns foster reuse by providing an abstraction that describes effective solutions to common problems. Similarly, effective tactics, if expressed appropriately, offer the potential for reuse and application to a range of similar problems. For example, the infantry *pincer* tactic, in which both flanks of the enemy are attacked simultaneously, can also be applied to air combat. The abstraction of a tactic into a design pattern offers the potential for sharing and reuse, rather than compelling the analyst to reimplement a tactic from scratch. Such behavioural design patterns are not currently offered by AOSE methodologies.

A mission expresses what the autonomous system needs to achieve as well as other ancillary information such as potential risks (see Section 4.1.1). The mission design artefact was added in response to the requirements of our user community of USW analysts, who run multiple USW tactical scenarios to investigate study questions such as *"Will Sonar X improve tactical outcomes for this set of missions?"*. The tactics that they develop and implement only make sense in the context of one or more missions that are used to investigate the modelling question. Thus, it is important that these missions be specified at the beginning of the tactics design process.

Plan diagrams are a high-level, diagrammatic procedural representations of tactics that provide the added benefit of enabling SMEs (Subject Matter Experts) to critique the model early on. Although the Prometheus methodology includes *process specifications*, which are similar to plan diagrams, the tool (PDT) does not. Furthermore, the Prometheus process specification notation is fairly minimal, and so it was necessary to extend it to include goals, goal conditions, wait-for conditions, failure nodes, and asynchronous merges (termed joins).

This work makes several contributions: (i) the state of the art in AOSE methodologies is extended to facilitate the modelling of tactical decision-making systems by adding missions, a wider range of goal structures, plan diagrams and tactics design patterns (Section 4); (ii) the extended methodology is implemented in a new application, the TDF tool, that builds on PDT (Padgham et al., 2008) (Section 5); and (iii) a preliminary evaluation is presented comprising an assessment by USW analysts, and a UAV pilot study indicating that TDF provides significant benefits to those engaged in building autonomous, tactical decision-making systems (Section 6). Although our work is based on the Prometheus methodology, we believe that these enhancements could be applied to other AOSE methodologies, for example, O-MaSE and Tropos. These methodologies have a number of similarities to Prometheus (DeLoach et al., 2009), and could be extended, for example, to augment their concept of goal to include TDF's goal conditions and goal control structures.

2. Background

This section provides a brief overview of the requirements for autonomous tactical decision-making systems, followed by an introduction to the BDI (Beliefs, Desires, Intentions) paradigm and AOSE.

2.1. Tactics

At their most basic, tactics are the means of achieving a goal. Michon (Michon, 1985) presents a hierarchical control model for driver behaviour. The model comprises (i) a strategic level in which an overall plan is developed, (ii) a tactical level that reacts to the current situation using manoeuvres, and (iii) a control level that handles low-level actions such as accelerating and braking. In Michon's model, tactics handle short-term goals and responses to the immediate requirements of the situation. This view of strategy and tactics mirrors that generally employed by the military; strategy relates to high-level planning and the general approach to a problem, whereas tactics specify the means of achieving more short-term goals.

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.

2.2. Autonomous systems

Autonomous operation is an important requirement for unmanned vehicles, such as UAVs, UUVs and spacecraft. Taking UAVs as an example, although some functions operate autonomously, e.g. following a predetermined flight profile, ground crew perform all high-level decision-making. Fully autonomous UAVs have yet to be operationally deployed. Nevertheless, increasing the level of autonomy has been an active research area for some time, for example, the NASA Remote Agent Architecture for autonomous spacecraft (Muscettola et al., 1998), autonomous UAV's for airborne science missions (Wegener et al., 2004), and NASA's space technology roadmap (Council, 2012).

It is important to distinguish between *automation* and *autonomy*. In an automatic system, actions are independent of context – the system responds to its sensor input in a fixed manner and does not engage in more flexible behaviour, such as seeking further environmental input to delineate the best course of action. In contrast, an autonomous system is designed to acquire knowledge about the context and use it to make more situation-dependent and effective decisions (Doyle, 2003). Having said that the distinction between automation and autonomy is not hard and fast. Indeed, various levels of autonomy can be discerned within the class of autonomous systems itself, for example, the Autonomy Control Levels outlined in Office of the Secretary of Defense (2001).

Under certain circumstances, even a simple UAV reconnaissance scenario may require sophisticated decision-making capabilities. To illustrate, consider a photoreconnaissance mission in which a UAV is tasked to fly across hostile territory and photograph an installation. In the simplest case, all it has to do is fly to the location, photograph the target, and return to base. However, unexpected events may require that it reconsider its current plan on how to achieve the mission objective, or may even mean that the objective must be abandoned. Possible events and relevant variables include:

- *Unexpectedly low fuel level*: Will there be enough fuel left to return to base after the target has been photographed? Is mid-air refuelling an option? Is photographing the target so important that it is permissible to run out of fuel and crash once the images have been transmitted to base?
- *Icing (water freezing on the airframe)*: Icing is an ever present threat that has been responsible for the loss of many aircraft and lives over the years. Is there a danger of icing? Can the mission be completed by flying around the icing threat? Could the changes in aircraft dynamics be explained by icing? If so, enable de-icing measures. Are they proving effective?

Ideally, the UAV's decision-making capability should be comparable to that of a human pilot, and this is mandatory if it will be operating in civilian airspace; see, for example, the development of standards for UAV certification by the UK Civil Aviation Authority (Haddon and Whittaker, 2003), and more recently AMC UAS.1309 (JARUS, 2011).

Effective decision-making in an environment where unexpected events can arise requires certain minimal capabilities.

- The autonomous system should be goal directed, i.e. its activity is focused on the achievement of its goals.
- It should balance goal-directed behaviour with an ability to exploit opportunities or react to dangerous situations. This involves a continual process of environmental monitoring to maintain situation awareness (Endsley, 1988).
- It should be able to drop goals that are no longer achievable and take action to circumvent a situation that is blocking the achievement of one of its goals.

These capabilities are not well served by typical programming languages. Although languages such as Ada are well suited to programming aircraft avionics, for example on the Boeing 777, they are not a good fit for autonomous tactical decision-making.

2.3. The BDI paradigm

Based upon work in philosophy on intentional systems and practical reasoning (Dennett, 1987; Bratman, 1987), the BDI programming paradigm was developed to support the above-mentioned mix between proactive and reactive behaviour. The first implementation of the approach, PRS (Georgeff and Ingrand, 1989), focused on real-time fault diagnosis for Space Shuttle missions. The BDI model of agents has been used for high-level decision-making on UAVs, e.g. (Karim et al., 2004), and on QinetiQ's BAC 1–11 UAV surrogate[1].

A BDI agent is an autonomous computational entity that, unlike a typical object in the object oriented paradigm, is constantly executing a *sense, decide, act* loop. Whereas an object's behaviour is invoked by external messages, a BDI agent is more internally driven, persistently sensing its environment, deciding what to do next and then performing an action if appropriate. A BDI agent continually reassesses its course of action in response to incoming sensory information.

BDI architectures vary in detail, but all share the core attributes illustrated in Fig. 1. A key element is that these systems have a predefined set of plans (Plan Library) that is used to respond to external events from the environment as well as internal events. Internal events are the proactive goals generated within the system.

In the schematic architecture in Fig. 1, sensory information from the environment is mapped to *events* that accumulate in a queue. The BDI Engine takes the next event from the Event Queue and selects all relevant plans from the plan library that could be used to handle the event. The contextual information within each plan is then used to determine the plans that are applicable in the current context and one of these plans is selected to be executed. The chosen plan instance is added to the agent's intentions. In each cycle, the agent chooses one step of an intention (i.e. instantiated plan) to progress, which may produce an action that is passed to an actuator that effects a change in the environment.

Although there are no comparative studies of BDI applied to autonomous systems, a function point analysis over six projects encompassing 10 person-years of development found that, for the complex applications being developed, the benefits of BDI were highly significant:

[1] http://www.theengineer.co.uk/news/autonomy-at-qinetiq/297140.article (last accessed 20 July 2015)

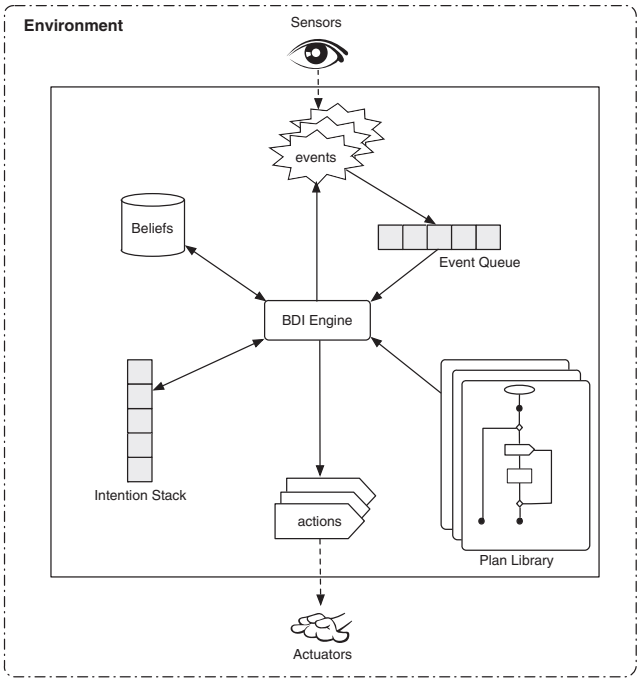

Fig. 1. BDI Architecture.

In a wide range of complex business applications, we show that the use of Belief-Desire-Intention (BDI) technology incorporated within an enterprise-level architecture can improve overall developer productivity by an average 350%. (Benfield et al., 2006)

2.4. Agent oriented software engineering

The development of TDF was motivated by a community of users who have been using paper-based workflow diagrams and UML to design tactics models for undersea warfare. As the library of models has grown in size, this approach has not scaled well. A key requirement in this domain is for SMEs (Subject Matter Experts) to be able to view behaviour models at a design level which can then be systematically propagated into the implementation. The lack of an appropriate design methodology (and tools) to capture the tactics makes this task almost impossible for large systems.

UML (Unified Modeling Language) is the de facto standard notation for software modelling. UML focuses on syntactic rules for defining models, leaving semantics informally defined and open to interpretation. Because of the similarities between agents and objects, early approaches to AOSE are built upon OO (Object Oriented) methodologies, e.g. the AAII Methodology (Kinny et al., 1996) and Agent UML (Bauer et al., 2001). AOSE methodologies and tools continue to be used and enhanced, and although there is considerable overlap, each focuses on different aspects of the problem. For example, Tropos is a BDI-based methodology that focuses on early requirements analysis. O-MaSE uses UML notation to express agent-based design concepts and adds the notion of multi-agent organisations. A comparative review of O-MaSE, Tropos and Prometheus can be found in DeLoach et al. (2009).

Prometheus forms the basis of TDF, and is a BDI methodology based upon over 15 years of development in partnership with commercial designers of agent-based systems and tools. It encourages a top-down approach to the design of multi-agent systems that comprises three main stages: System Specification, Architectural Design and Detailed Design. System Specification defines the relationship between the system and its environment in terms of the overall system goals, typical scenarios that can occur, and inputs and outputs. Architectural Design fleshes out the internal structure of the system, including the data stores, roles to be filled by agents and the communications protocols between them. Detailed Design specifies the internals of the agents in terms of their capabilities and the plans they can use to achieve their goals and respond to the environment.

3. Research method

Before outlining the TDF methodology, we provide some background to the research method used to develop and evaluate TDF. In developing our software engineering approach to modelling tactical decision-making, our main research methods were *Conceptual Analysis* and *Concept Implementation*; according to Glass et al. (2002), these are the dominant approaches adopted by software engineering researchers. The development of our methodology comprised Requirements Elicitation and Analysis (Section 3.1), Conceptual Analysis (Section 3.2), Concept Implementation (Section 5), and Evaluation (Section 6).

3.1. Requirements elicitation and analysis

The original requirements for TDF came from a group of USW analysts who were interested in improving how they model submarine commander tactical decision-making, with a view to generating quantitative predictions of how proposed capability will impact future submarine performance and operational effectiveness in USW scenarios. Requirements elicitation revealed the following key problems to address (in descending order of importance):

R1 – Highlight reactive/deliberative decision-making. In designing and implementing USW tactics, the most problematic aspect for analysts is how to represent the response to unexpected changes in the tactical situation.
R2 – Facilitate reuse of tactics across different scenarios.
R3 – Aid tactics validation. Validation should be supported by providing a means for SMEs to understand and critique the models.
R4 – Capture scenario-specific constraints. The selection of the most appropriate tactic depends on the situation, and these situational constraints need to be represented explicitly so that matching tactics can be developed and/or found.
R5 – Generate skeleton JACK code. Over a period of 10 years, the USW analysts had amassed a substantial library of JACK-based tactics, and an infrastructure that integrated JACK into their USW simulation platform.

3.2. Conceptual analysis

This stage began with the collection and analysis of definitions of the term *tactic* in various domains, including military and management science. The distinction between tactics and strategy was clarified, with the latter taken to denote a higher-level category that groups related tactics in terms of their general approach to achieving the goal. For example, *stealth* is a strategy, whereas in USW, restricting oneself to passive sonar or transiting to a separate thermal layer are particular tactics for maintaining stealth. Our resulting conceptualisation of tactics was discussed in Section 2.1. The key feature of tactics is that they are not only primarily goal-directed but are also responsive to changes in the tactical situation; this observation led us to focus on agent-based modelling and AOSE methodologies in particular.

A systematic review of the literature on hybrid reactive/deliberative reasoning (requirement R1) was undertaken, with a view to determining how changes in course of action have been represented in previous design methodologies. The review revealed that, although there has been a lot of work on algorithms and architectures for balancing reactive/deliberative modes of reasoning, design methodologies have not explicitly incorporated this

aspect. Consequently, we focused on developing a novel method of representing reactive/deliberative tactical decision-making, and in the interests of model validation, looked at how to make the notation accessible to SMEs, by casting it at the goal level and in diagrammatic form.

The requirements for tactics reuse (requirement R2) were initially addressed by reviewing the literature on design patterns, as well as cognitive modelling, to determine the degree of support for reuse in AOSE methodologies and behaviour modelling. We also considered which aspects of object-oriented design patterns could be repurposed to support tactics reuse. In the realm of software engineering, design patterns were first proposed for object-oriented programming (Gamma et al., 1995) 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 (Lange and Mitsuru, 1998). This successful mapping is not surprising, because at the system level, an agent can be viewed as an object, i.e. an encapsulation with internal state and a set of interaction methods, cf. (Aridor and Lange, 1998).

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 (Taylor and Wray, 2004), necessitating the development of a different class of design pattern. Hence, in this work we directed our effort towards developing design patterns that encapsulate goals, tasks and contextual information.

With regard to validation (requirement R3), and based on previous successes in having pilots validate diagrammatic, BDI-based air combat tactics (Murray et al., 1995), we opted to develop an easy-to-understand, diagrammatic representation of tactics. To determine the extent to which validation has been supported in previous behaviour models, we surveyed existing modelling methodologies and tools, as well as diagrammatic procedural behaviour programming languages such as PRS (Georgeff and Lansky, 1985), dMARS (d'Inverno et al., 1998), and JACK (Winikoff, 2005). We have been working with these procedural, diagrammatic BDI languages for over 20 years, and our experience is that SMEs readily understand and can critique behaviour models implemented in this form, and indeed, this has some support in the literature, e.g. Wallis et al. (2002), Murray et al. (1995). However, although this procedural view of tactical behaviour facilitates validation of the steps performed during the execution of a tactic, it does not provide a high-level view of important dimensions of the tactic, including its goal-based decomposition, goal dynamics, scenario-specific constraints (requirement R4), and the types of mission that the tactic relates to. A further review of the literature confirmed that, on the whole, previous AOSE methodologies do not provide a means of capturing scenario-specific constraints (requirement R4) in the early, high-level stages of the design process. Rather, this contextual information gets added later on, and at a procedural, rather than declarative level. For example, in Prometheus this information is represented in the context conditions of plans, during the final Detailed Design stage (Padgham and Winikoff, 2004, p. 113). In Ali et al. (2009), an extension to Tropos is proposed that combines contextual information with early, goal-based requirements in order to model software variability.

Following the above reviews and analyses of current practice and its shortcomings, we developed the TDF methodology, using Prometheus as a starting point, because of the requirement that the TDF tool generate JACK code (requirement R5). To heighten the general applicability of our methodology, we surveyed common practice in tactics modelling. Areas examined included forest firefighting, UAVs, USW, air combat, infantry, attack helicopters teamed with UAVs, and tank battles. In general, tactics have been directly implemented either in a platform-specific language or a more platform-independent, AI-based (Artificial Intelligence) language such as Soar (Laird et al., 1994), dMARS or JACK. In the past, UML has been proposed for agent-based modelling (Bauer and Odell, 2005), and indeed all of the groups surveyed, who engaged in design-based modelling of tactics, used UML. Business Process Model and Notation (BPMN) (OMG, Business Process Modeling Notation, 2006) is also a candidate for agent-based modelling of procedural behaviour (Endert et al., 2007), and so we considered its use in TDF. Although UML activity diagrams and BPMN are suitable for modelling the procedural aspects of tactics, interruptibility must be expressed explicitly and it is relatively cumbersome to do so. This reflects the fact that activity diagrams and BPMN were designed for processes where interruption is the exception rather than the rule. This difficulty could perhaps be ameliorated by augmenting BPMN with a declarative, constraint-based, change handling mechanism, such as that discussed in van Der Aalst et al. (2009). In contrast to UML and BPMN, BDI languages are intended to mix reactivity with proactivity, and so are well suited to tactics modelling where responsiveness is the key to effectiveness. Nevertheless, to improve the accessibility of the TDF notation, we adopted the iconography of UML activity diagrams.

4. Tactics development framework

This section outlines the whole design process but focuses on the TDF extensions of the Prometheus methodology. TDF extends Prometheus with missions, a richer goal structure, plan diagrams and tactics design patterns (in TDF, referred to as *tactics*).

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

- *System specification*: Identification of system-level artefacts, namely missions, goals, scenarios, percepts, actions, data, actors and roles.
- *Architectural design*: Specification of the internals of the system, namely 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, namely tactics, plan diagrams, internal events and capabilities.

Fig. 2 illustrates the Prometheus methodology augmented with TDF extensions required to better represent tactical decision-making (shown in *italics*). The diagram updates and augments the original Prometheus methodology diagram (Fig. 1) found in Padgham and Winikoff (2005). It horizontally partitions the design process into the three Prometheus stages: System Specification, Architectural Design and Detailed Design. The first of the three columns highlights design artefacts that relate to system dynamics, namely scenarios, goals, protocols and plan diagrams. The TDF tool provides various editable diagrammatic overviews of the design (second column). As the designer moves from system specification through to detailed design, various structured forms accumulate (third column), for example, missions are defined during system specification, whereas plan diagrams wait until detailed design.

The directed arcs define the propagation of information across the design and the opportunity for crosschecking. For example, *actions* defined during system specification are propagated through to detailed design, where they are available for inclusion in plan diagrams. Crosschecking ensures, for example, that a plan cannot include a percept that is not also defined as being handled by the agent the plan belongs to (in the System Overview). Crosschecking and propagation ensure that the overall design remains internally consistent.

The following overview of TDF is illustrated with the help of an ISR (Intelligence Surveillance Reconnaissance) vignette involving a UAV tasked with locating and photographing a suspected enemy convoy. The UAV has no offensive capability but can employ countermeasures to evade an attack.

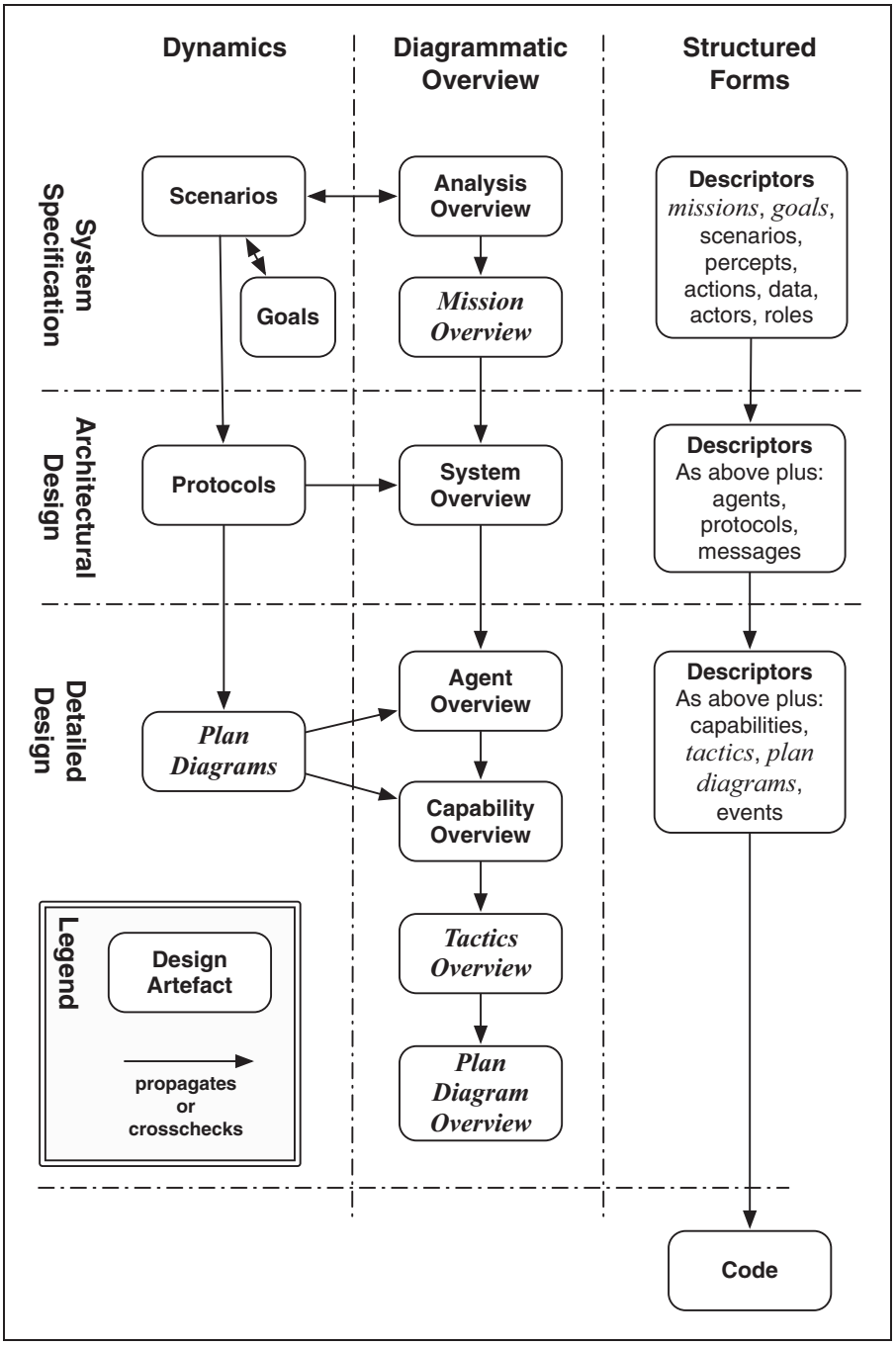

Fig. 2. TDF augmented Prometheus methodology.

4.1. System specification

The System Specification defines the overall interface to the environment, typical interactions and a high-level, goal-oriented view of the system's capabilities. System Specification typically begins with the definition of the missions that the system is being designed to handle. The mission is central to the TDF design process. Each mission defines a *macro* interaction between the system and its environment. Missions are a central driver in the tactics design process.

Definition of the missions is followed by identification of the external human/software entities (actors) that will interact with the system, and how the interactions will unfold (scenarios). This is then fleshed out by specifying the system inputs (percepts) and outputs (actions), data used, goals and roles. Each of these design artefacts can be defined in whatever order best suits the designer's preferred workflow.

Although the methodology refers to artefacts such as actors, scenarios, percepts and actions, these are not instances; rather, they are types/classes, in the general sense that each is a term that denotes a set of instances. For example, a `position` percept would refer to the set of all percepts of type `position`, rather than a particular `position` datum that is received at runtime.

4.1.1. Mission

Missions form an important part of the TDF design process. To some extent, the mission design artefact specifies system requirements. However, it is in one sense narrower, and in another broader

than traditional requirements. It is more narrow to the extent that it specifies one possible interaction between the system and its environment. It is broader to the extent that it embodies particular design attributes (e.g. Operational Constraints, Risks) that define the general properties of the mission. Note that a TDF design typically contains a collection of missions, i.e. a specification of the missions that its tactics can handle. The mission artefact does not substitute for requirements, for example, goal-oriented requirements are represented in TDF using goal structures, rather than missions.

Each mission defines one expected interaction between the system and its environment, but its usage differs from the scenario concept (see Section 4.1.6) in that it always relates to a top-level, macro-objective of the system, whereas scenarios can denote any arbitrary interaction. The structure of a mission is as follows and includes example values for the attributes:

- *Name*: Each mission has a unique identifier, e.g. `Photographic Reconnaissance`.
- *Objective*: Every mission has a primary objective that defines mission success, e.g. `Photograph Target`.
- *Secondary objectives*: Some missions may have secondary goals, to be achieved if an opportunity presents itself and it will not endanger the achievement of the primary objective. For example, `Photograph Potential Enemy Camps`. Although secondary objectives could be included in the Mission Statement, our users tell us that a significant number of the missions they model include some form of secondary objective. Therefore, this property was added to prompt the modeller to think about the secondary objectives of the mission to be modelled.
- *Mission statement*: A description of the mission type, for example: "you are tasked with locating and photographing a convoy. The location of the convoy is roughly known, but you will need to search the area to locate it."
- *Operational constraints*: A state of the world to be maintained for the duration of the mission, for example: "Do not enter any no-fly zones."
- *Risks*: These help prompt the design of tactics to handle unexpected but dangerous situations, for example: "the convoy is affiliated to a hostile group, so there is a possibility that you will be attacked."
- *Opportunities*: Aspects of the situation that could be exploited, for example: "the convoy is expected to be moving fast on dirt roads. A dust cloud could indicate its location."
- *Scenarios*: A scenario is a sequence of episodes that can occur during the mission. Example scenarios for this mission might include: `Navigate to Area of Operation, Search for Target, Classify Target, Avoid Incoming Missile, Photograph Target`.
- *Data*: General mission-related data, for example: `map, no-fly zones`.

4.1.2. Actors

Although the term *actor* can be taken to imply wilful intent, there is no need for this to be the case. Essentially, from the system's perspective, actors are entities that generate percepts, whether wilfully or otherwise. They can also be affected by system actions. From a modelling perspective, actors are opaque entities. In contrast to agents (Section 4.2), 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. Actors are used to represent the UAV's sensory and effector subsystems (see Sensor, Countermeasures and Navigation Subsystems in Fig. 3).

4.1.3. Percepts and actions

The system interacts with the actors in its environment by way of inputs (percepts) and outputs (actions). As the design is fleshed

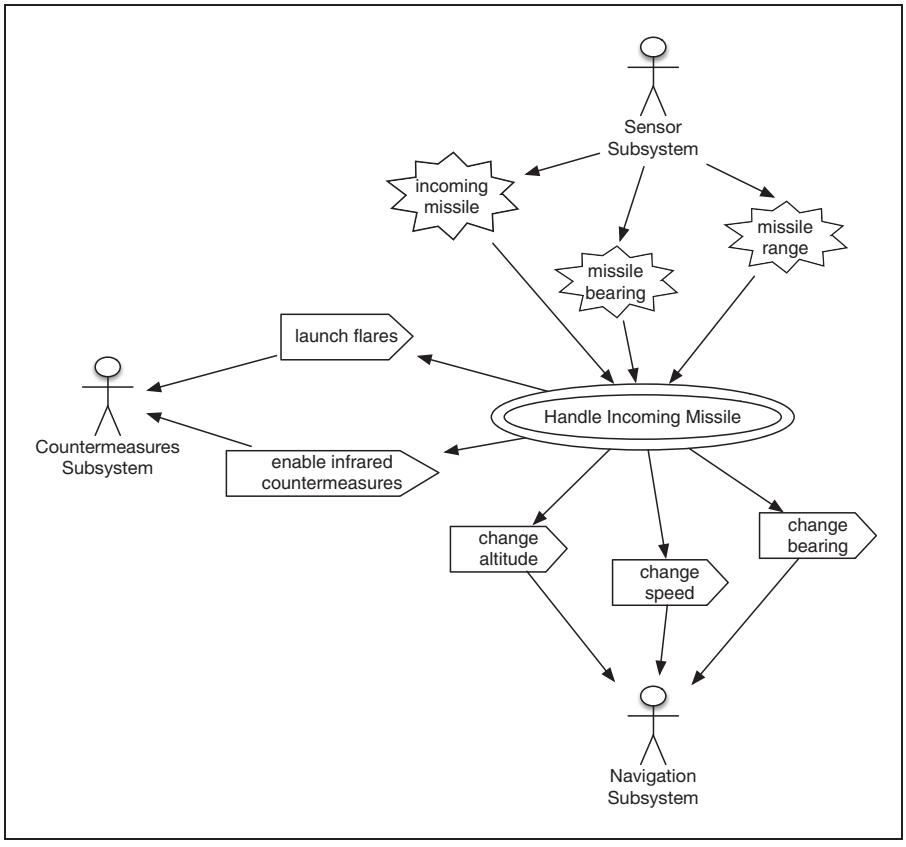

Fig. 3. Design artefacts: scenario, actors, percepts and actions.

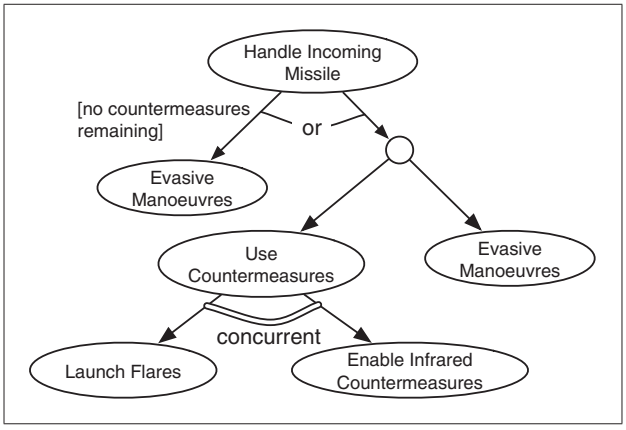

Fig. 4. Goal structure to handle incoming missile.

out, questions will arise regarding the information required from the environment, and how the system must act on the environment to achieve its goals. This will lead to the addition of further percepts (e.g. `incoming missile`) and actions (e.g. `launch flares`), as illustrated in Fig. 3.

4.1.4. Goals

The system is designed to meet its mission objectives, and those objectives lead to the derivation of sub-goals that have to be achieved. These goal/sub-goal relationships are expressed as a goal structure that defines the hierarchy using `and` , `or` and `concurrent` connectives (see Fig. 4). 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.

The goals in the goal structure diagrams are ordered from left to right and the graph implicitly comprises `and` nodes, i.e. all goals must be achieved in left-to-right order. Tactics typically involve a sequence of goals that have to be achieved one after another, thus, a left-to-right conjunction makes the best default interpretation of a goal structure.

TDF extends this with control structures that better capture inter-goal relationships that are important to tactical decision-making. These goal-related attributes are usually implicit, hidden deep in the autonomous 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 provides the following goal control structures:

- *Conditional*:A conditional goal is used to denote a goal that should only be adopted if certain conditions are true. If the conditions are not met, the arc is skipped. (see `[no countermeasures remaining]` example, Fig. 4). Effective tactical decision-making is very context-dependent, but contextual information is usually embedded inside the procedures (plans) that implement the tactic. Conditional goals allow such implicit contextual information to be expressed at the goal level during the early stages of the design process.
- *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 Fig. 4, `Launch Flares` and `Enable Infrared Countermeasures`.
- *Unordered*: In TDF, sub-goals that are not concurrent are implicitly attempted from left to right. This default ordering can be circumvented by specifying that particular goals can be attempted in any order.
- *Anonymous node*: Used to partition the goal structure into sub-trees that have a different logical relationship to one another. For example, in Fig. 4 the anonymous node (empty circle) partitions the goal structure into a disjunctive (first `Evasive Manoeuvres` goal and anonymous node) and a conjunctive (`Use Countermeasures` and the second `Evasive Manoeuvres` goal) sub-tree. Its diagrammatic role is the same as that of parentheses in composite, textual logical expressions.
- *Asynchronous*: Operationally, an asynchronous goal's parent does not wait for it to succeed. This is useful in cases where a sequence of tasks needs to be performed asynchronously, 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 Duff et al. (2014).
- *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 should be dropped.

To illustrate some of these goal control structures, consider the objective of evading an incoming missile (Fig. 4). This example shows an application of conditional and concurrent goals, and the combination of `and` and `or` sub-trees via an anonymous node. If there are no countermeasures remaining, the `Evasive Manoeuvres` goal is tried (a conditional goal). Otherwise, countermeasures are used, which involves launching flares and using infrared countermeasures (concurrent goals) to interfere with the missile's ability to maintain a lock on the UAV. As soon as countermeasures have been deployed, the second branch emanating from the empty circle (an anonymous node) is tried, i.e. evasive manoeuvres begin.

Without the inclusion of a conditional goal, it would not be apparent that the UAV immediately adopts the `Evasive Manoeuvres` goal if there are no countermeasures available. Without the `concurrent` inter-goal relation, one could erroneously surmise that the UAV launches flares and then enables its infrared countermeasures. In fact, when there is an incoming missile, it is vital that these two actions are performed as soon as possible, i.e. concurrently.

4.1.5. Roles

As the goal structures are developed, it soon becomes apparent that they perform particular functions within the overall system. Each function can be expressed as a *role*. Percepts and actions required by the goals in a role are grouped within that role. Roles ultimately encapsulate agent functionality and are used to guide the decomposition of the system into agents. Fig. 5 shows the `Situation Awareness` role that is responsible for the `Maintain Situation Awareness` goal. The role involves actions such as `enable radar` and percepts such as `missile bearing`.

4.1.6. Scenarios

Scenarios are used to map out key sequences of task-related activity in the system (akin to use cases). A given scenario will comprise a sequence whose members include goals, actions, percepts and sub-scenarios. As each scenario is mapped out, new goals, actions and percepts will crop up, leading to the development of variations as well as completely new scenarios. This iterative process completes when the designer feels that there are enough scenarios to indicate the overall behaviour of the system. It is better to develop a few important scenarios rather than try to cover all cases, which can lead to a

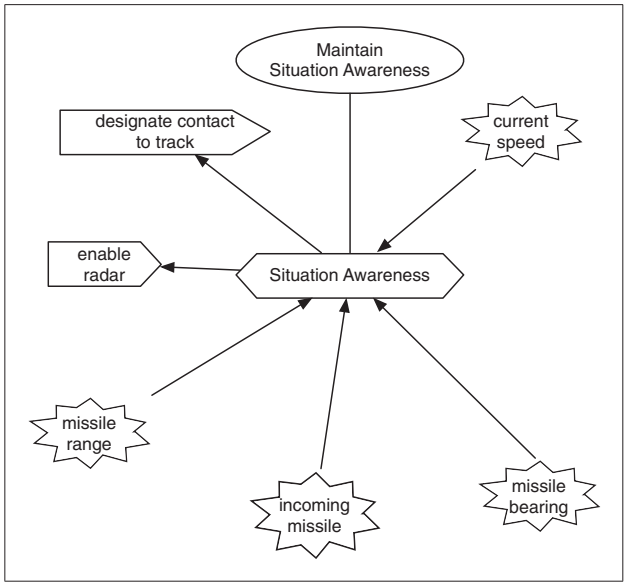

Fig. 5. Situation Awareness role.

cluttered design. In any event, variations can be documented in the design. Ideally, all of the percepts, actions and goals should be involved in at least one scenario (see Fig. 3 for a diagrammatic representation of the `Handle Incoming Missile` scenario's relationship to actors, percepts and actions).

4.2. Architectural design

In the Architectural Design stage, the main tasks are to define agent types (in terms of roles) and the inter-agent communication that needs to occur (protocols and messages). The allocation of functionality (roles) to agents can be based on a number of factors, including preferring high cohesion and low coupling. Cohesion refers to the extent to which the components are related, either by function or in terms of time. Coupling is the degree to which components depend upon one another for data.

4.3. Detailed design

The main purpose of the Detailed Design stage is to specify the different ways that the system can achieve the functionality defined in the System Specification stage. One of the powerful features of BDI is its support for the representation of multiple ways of responding to a percept or satisfying a goal. Each method typically applies in a different context and is represented as a separate procedure (plan).

The System Specification defines *what* the system does, but not how; the latter is the purpose of the Detailed Design. In TDF, the *how* is specified using a diagrammatic plan representation that is based on UML activity diagrams. TDF plan diagrams are a level of abstraction above the implementation, and should be viewed as diagrammatic pseudo-code rather than an executable implementation. A TDF plan diagram specifies the general steps of a plan without getting bogged down in implementation detail.

4.3.1. Capabilities

In the Detailed Design, *capabilities* are defined that encapsulate the tactics, goals, plans, messages, percepts, actions and data required to perform a particular function within the system. Capabilities are reusable pieces of functionality that can be assigned to different agents in the system, and are analogous to Abstract Classes in object oriented programming. They may correspond to roles, combine a number of roles, or implement part of a role. A capability can also be decomposed into a hierarchy of sub-capabilities, allowing the reuse of more fine grained functions within the Detailed Design. Fig. 6 shows the goal/plan tree of the `MissileEscaping` capability. The goal/plan tree shows what goals the capability can achieve and the various ways (plans) that it can achieve those goals. The `incoming missile detected` message triggers the `MissileDetected` plan, and the `Handle Incoming Missile` goal, in turn, triggers the `IncomingMissile` plan, and so on. The `HowLongToArrive` plan computes the `missile ETA` and updates the data store (top right, Fig. 6).

4.3.2. Agents

Agents are autonomous, computational entities, and are assigned the capabilities needed to perform their function. Although they have the same structure as capabilities (i.e. they contain tactics, goals, plans, etc.), unlike capabilities they can be instantiated in the running system to perform computation.

4.3.3. Plan diagrams

There is a long tradition of using diagrams to represent procedures, whether as flow charts (Gilbreth and Gilbreth, 1921), Petri Nets (Petri, 1966), recursive transition networks (e.g. PRS (Georgeff and Lansky, 1985)), UML activity diagrams (Heaton, 2005), BPMN (OMG, Business Process Modeling Notation, 2006) 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. Another motivation for the adoption of UML diagrammatic notation was the earlier related work on unifying existing AOSE notations through the use of UML iconography (Padgham et al., 2009). 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, 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.

Fig. 7 lists the node icons used in TDF plan diagrams. The node types are:

- *Initial*: Plan diagrams have a unique initial node, preceded by either a goal, percept or message that acts as a trigger for the plan. Optionally, the initial node can have a guard that specifies the context in which the plan is applicable.
- *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.
- *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.

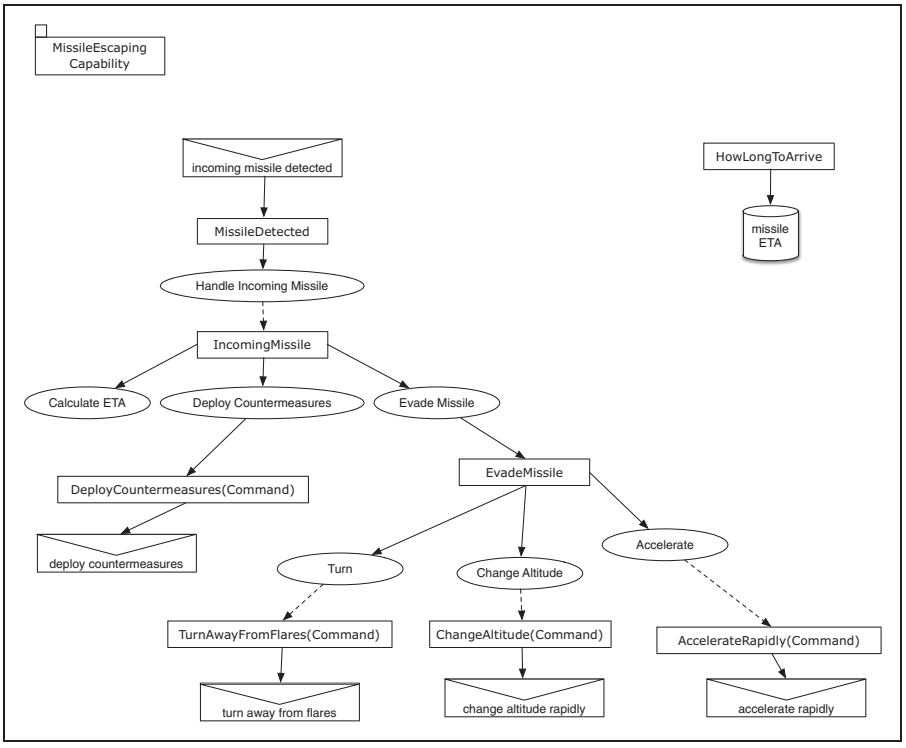

Fig. 6. MissileEscaping capability.

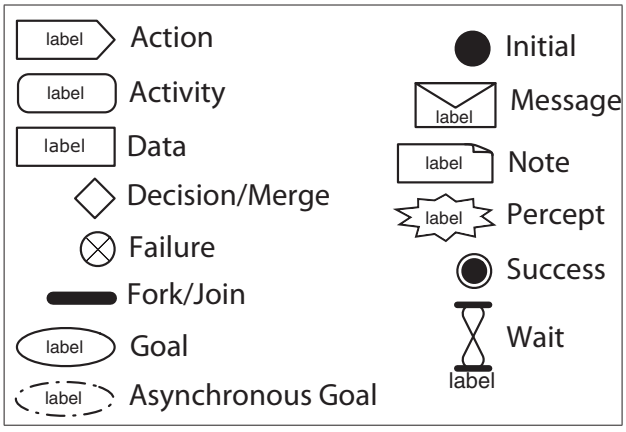

Fig. 7. Plan diagram node icons.

- *Percept*: Precedes initial node. Denotes a reactive plan.
- *Success*: Terminates the plan successfully.
- *Wait*: Waits for a condition to become true.

The plan diagram in Fig. 8 incorporates a percept, initial node, action, fork, three asynchronous goals, join, a goal and a success node. The plan is triggered by a `low oil pressure` percept, and immediately reduces the load on the engine. It then sets up three asynchronous goals to monitor related engine indicators; these goals are pursued in their own separate threads, and the plan does not wait for them to succeed. Finally, it adopts the goal to reconsider the viability of the current mission in the context of the engine problem.

Note that, because of the underlying BDI model, each plan step is interruptible to the extent that the execution of other (higher priority) plans can be interleaved with this one. So, for example, if an incoming missile is detected and there is a higher priority plan for dealing with that event, this plan will effectively be suspended until the crisis is averted.

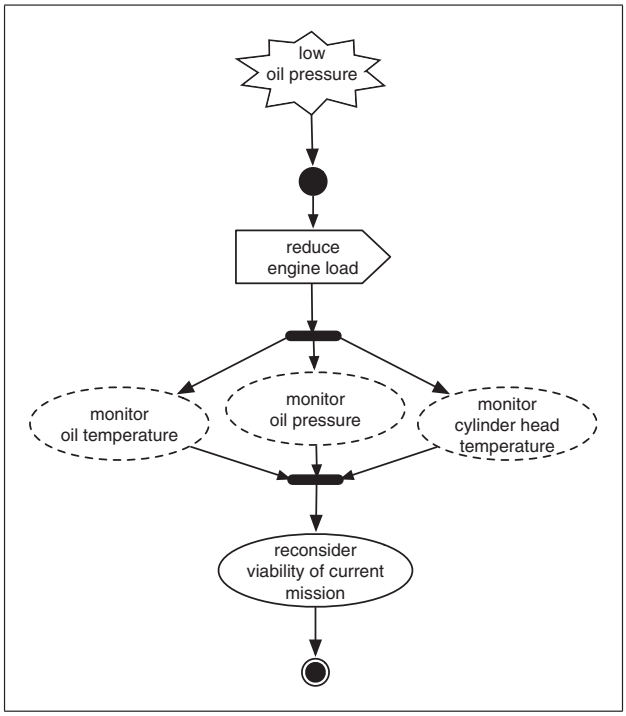

Fig. 8. Plan diagram to handle potential engine problem.

4.3.4. 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 (Chandrasekaran, 1986) 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) (McDermott, 1988) 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*.

This section outlines the properties of tactics design patterns. In developing TDF, an effort was made to use property names that are meaningful to analysts and SMEs; for example *objective* rather than *top-level goal*, and *goal structure* rather than *goal graph*.

- *Objective*: The objective is the top-level goal that the tactic is designed to achieve. An objective is usually named either with reference to what it does or what it achieves, for example, `land` (what it does) or `landed` (what it achieves). An example tactical objective in the UAV domain is `Escape from Incoming Missile`.
- *Trigger*: A percept that triggers the use of the tactic, e.g. `low oil pressure`. This is used to model a reactive tactic, i.e. one that is triggered by an environmental event rather than a goal.
- *Problem description*: A description of the types of problem the tactic applies to. For example "an incoming missile has been detected".
- *Solution description*: A description of how the tactic achieves its objective. For example, "this tactic uses countermeasures to distract the missile and then evades by turning, climbing/descending and accelerating".
- *Context*: A tactic is only applicable in a particular context, i.e. if its precondition is true. A precondition is a boolean test of the state of the world, e.g. "have sufficient countermeasures remaining".
- *Outcomes*: A description of how the world has changed after the tactic has achieved its goal, e.g. "have fewer countermeasures".
- *Restriction*: A situation that must be maintained for the duration of the tactic's execution, e.g. "do not enter no-fly zones".
- *Information required*: Information that the tactic requires to perform its function. These include the products of perception (percepts), e.g. `missile detected, missile bearing, missile range`.
- *Information updated*: Agent beliefs (data) updated by the tactic, e.g. `location, speed, altitude, number of countermeasures remaining`.
- *Goal structure*: A reference to the top-level goals underlying the tactic, e.g. `Handle Incoming Missile`.
- *Plans*: A collection of references to plan diagrams, specifying procedural methods of achieving the goals in the goal structure. For example: `EvadeMissile, DeployCountermeasures, TurnAwayFromFlares`.
- *Source*: References to source material used to create the tactic, e.g. "Drafted from Doctrine Manual UAV-2-10".

In TDF, plans are grouped in terms of the tactics design patterns they contribute to and are represented at an abstract level independent of the particular implementation language. This encourages the designer to think in terms of high-level tactics and promotes procedural abstraction. Thinking in terms of tactics, rather than low-level procedures, also facilitates merging, reuse and maintenance of tactics sets. A tactics design pattern makes the important attributes explicit. For example, if a tactic requires that the UAV remain above a certain cruise altitude, this is defined as an explicit restriction. When considering merging this tactic with one that involves dropping under low cloud, it is obvious that there is a potential conflict. In our experience, this type of conflict is usually not apparent at the implementation level; for example, a `descend` action might be embedded deep down in the plan/goal graph and might not be noticed when inspecting at the top-level plans.

5. The TDF tool

The TDF plugin extends PDT with graphical tool support for the TDF methodology. The main purpose of the tool is to empower analysts with a mechanism to capture behaviour specifications such as tactics.

5.1. Prometheus design tool (PDT)

PDT is an Eclipse-based plugin used for designing BDI software systems. The PDT plugin supports all three Prometheus design phases, namely, System Specification, Architectural Design, and Detailed Design. PDT utilises the multi-editor architecture of Eclipse to provide multiple tabbed editors, such as Analysis Overview, Scenario Overview, Goal Overview, etc., in a single plugin. All of these editors, except for Capability Overview, Agent Overview and Plan Diagram, are accessed by clicking their respective tabs (present at the bottom of the content pane). The outline panel provides a tree view of the design hierarchy consisting of both static elements (common to all design instances), such as Goal and Analysis Overview, and dynamic elements (that depend on the particular design instance) such as capabilities and agents. Generally, a non-trivial design will contain multiple capabilities, agents and plans, and therefore the editors associated with these entities, that is, Capability Overview, Agent Overview, and Plan Diagram, respectively, get activated by clicking the relevant entities in the Detailed Design portion of the outline panel. Finally, the palette consists of entities such as goals and plans that can be added to relevant diagrams.

5.2. TDF tool

Core elements of the PDT plugin are shown in Fig. 9. The TDF plugin provides the following extensions to the PDT tool:

- missions;
- additional goal types and control structures;
- plan diagrams; and
- tactics.

5.2.1. Tactics and missions

The Tactics Overview editor (Fig. 10) allows the design of new tactics by relating them to goals and capabilities. The graphical depiction of tactics in the plugin is through a red, head-shaped icon. To maintain consistency with PDT we retain the icons for entities that are shared between PDT and TDF such as capabilities, goals, etc. The Mission Overview editor (Fig. 11) is used for linking missions with tactics and scenarios.

5.2.2. Additional goal types

The TDF plugin supports the modelling of additional goal types and control structures beyond standard PDT's achievement goals and `and/or` control structures. For their graphical representation, the TDF plugin provides variations of PDT's oval look for goals to show different goal types. In the goal overview palette (see Fig. 12), maintenance goals are depicted by the character "M" inside the oval, asynchronous goals have a dashed line, and anonymous goals are circular.

Maintenance goals are associated with *maintenance conditions* that are shown within square brackets. In terms of graphical representation, ordered goals are linked with solid lines whereas unordered goals are linked with dashed lines. The goal type, that is, `AND`, `OR`, `CON`, is shown just below the parent goal.

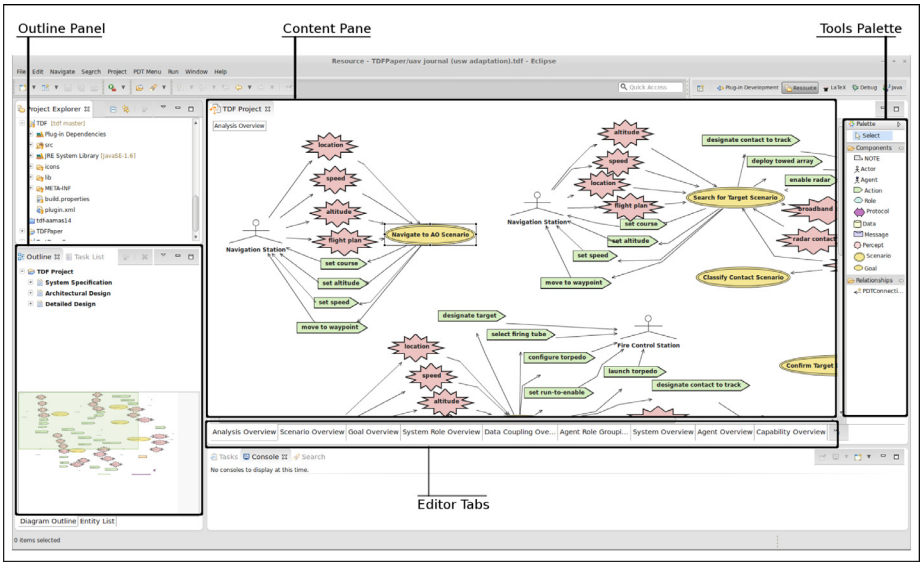

Fig. 9. TDF Eclipse plugin.

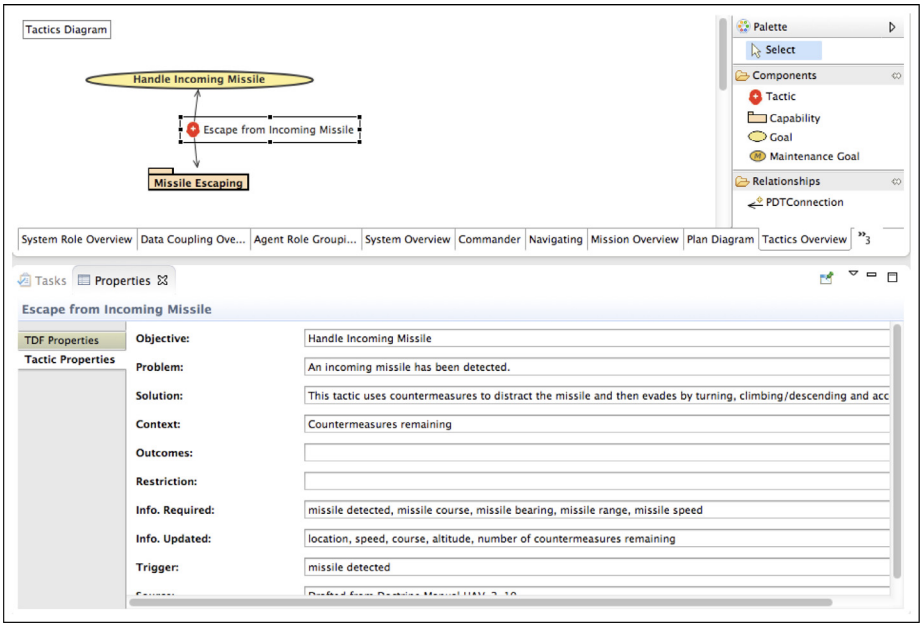

Fig. 10. Tactics overview in TDF plugin.

5.2.3. *Plan diagrams*

In PDT the internals of a plan are specified using informal text. Plan procedures are described as pseudocode but this is optional for designers. The TDF tool presents a more structured approach for specifying the internals of the plan in the form of detailed plan diagrams as described in Section 4.3.3.

Since a plan can be shared across different agents, plans exist at the same level as agents in the detailed design. Clicking a plan icon in the outline panel opens the plan diagram editor. By dragging and dropping various entities from the palette one can define the steps of the procedure, including:

- the plan's triggering goal;
- entry and exit points, including success and failure outcomes, for a plan;
- internal logic of the plan body;
- data dependencies of the plan;
- plan effects, such as actions and messages.

Fig. 13 shows an example of a plan from the UAV domain. As outlined earlier, the TDF methodology provides a conceptual framework to capture procedural knowledge such as plans in a principled way. The TDF plugin supports the accompanying methodology by providing an effective visualisation of the procedural and contextual know how associated with plans.

5.3. *Features of the tool*

The TDF plugin has desirable features such as *type safety, automatic propagation, batch image export*, 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.

1. *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. For example, when a user adds a maintenance goal to a diagram, she is

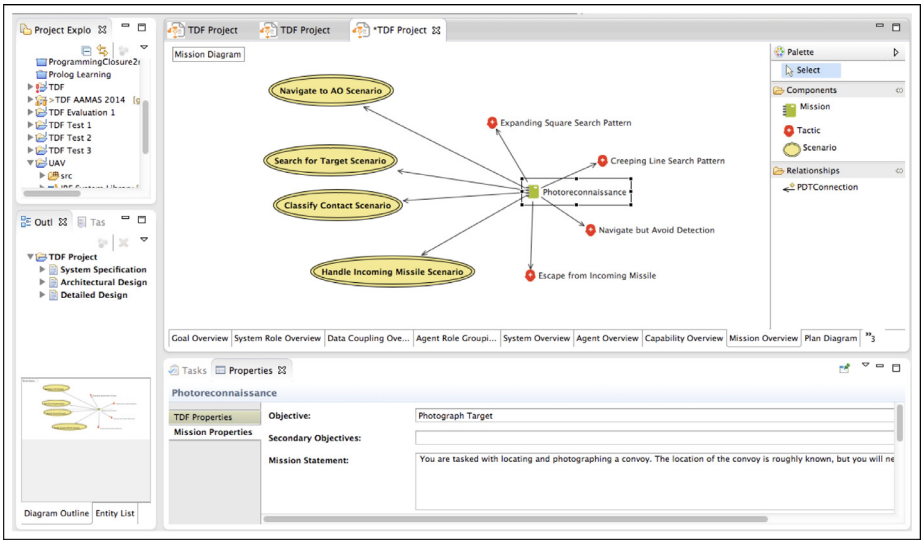

Fig. 11. Mission overview in TDF plugin.

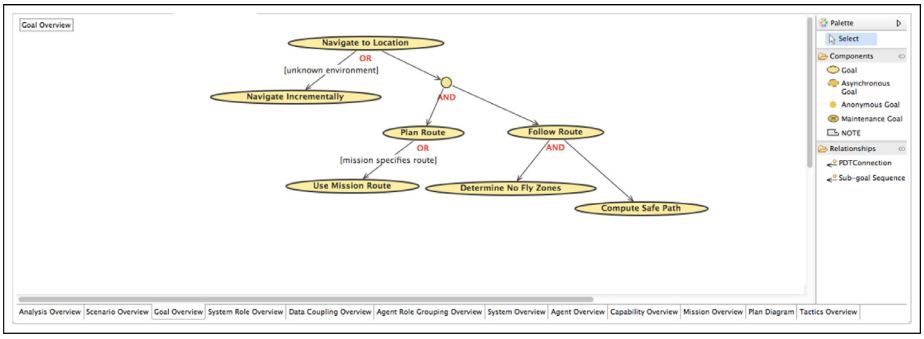

Fig. 12. Goal overview in TDF plugin.

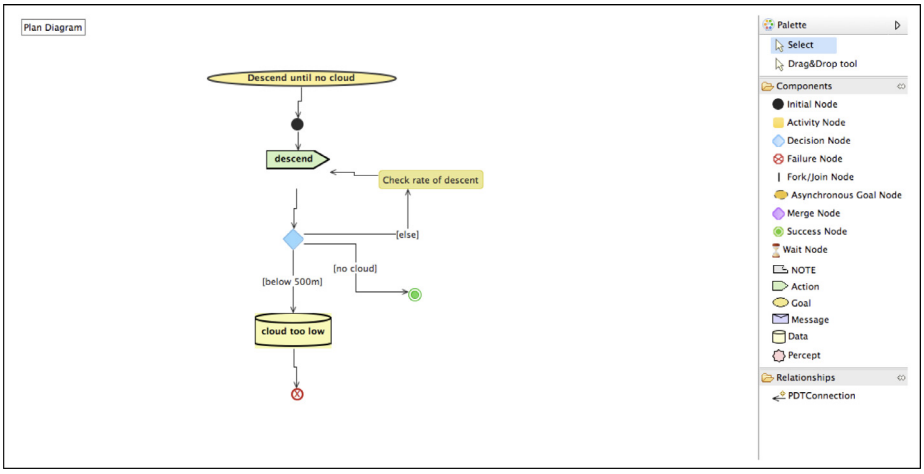

Fig. 13. Plan diagram in TDF plugin.

presented with a dialog box showing a list of all maintenance goals. She can then select an existing maintenance goal from the list, or create a new goal (by typing a unique name). This approach not only provides filtered entity-specific lists but also avoids typing errors.

2. *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.
3. *Code generation*: The plugin comes with an extension capable of generating skeleton code for JACK (Winikoff, 2005) and GORITE (Rönnquist, 2008). 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.

4. *Batch image export*: The plugin allows batch exporting of images of all the editors. These images can then be used to either create reports or share with other team members.

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.

6. Evaluation

To evaluate TDF, we selected UML as a baseline for comparison because it is regularly employed by our user community to design tactics, and our experience in various tactical domains suggests that UML is a natural choice for those with a standard software engineering background who are faced with building a tactical decision-making system. The objective of the evaluation is to test the prediction that, relative to UML, TDF improves the level of comprehension of a tactical decision-making system's design. The underlying assumption is that a tactics design that is easier to understand will facilitate reuse and sharing between developers.

6.1. Initial assessment by domain experts

An initial assessment of TDF was undertaken by a group of analysts who have been modelling USW tactics for many years, using a combination of UML and informal diagrams. The USW domain is very interesting from a tactical decision-making perspective – the submarine commander spends much time developing an understanding of the tactical situation. This has to be done stealthily, mostly relying on passive sonar to avoid detection by an adversary. Consequently, tactical decisions may be based upon limited and uncertain information.

Because the USW analysts' tactics library is classified, we were not given access to any information about it. However, they gave a positive assessment of TDF, indicating that an experimental evaluation would be worthwhile. Their feedback was that TDF will be beneficial in the context of their Monte Carlo constructive tactical simulations, because these generate a very large number of outcomes, making validation by visual inspection of the implementation impractical. The USW analysts believe that TDF's high-level design view of tactics facilitates analysis of the aggregated result of the many thousands of simulations they routinely run. Although the feedback they can give us is tightly constrained by the prevalence of classified information in the data set, the TDF project has been extended twice by our stakeholder, and this is a strong indication that TDF is working well for them. These are real military users, working on real, classified tactical problems.

6.2. A participant-based TDF/UML comparison study

The method, results and analysis for this study are as follows:

6.2.1. Method

A photoreconnaissance scenario was used to evaluate how well the designs were understood by the participants.

Participants

The participants comprised 10 computer science students, all very familiar with UML. They were not compensated for their involvement in the evaluation.

Experimental design

The participants were randomly assigned to the two experimental conditions (TDF and UML) in equal numbers using a between-subjects design.

Procedure

In order to minimise the need to learn the mechanics of the TDF and UML tools, the tactics 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 min 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 tactics and could ask questions related to syntax but not the behavioural implications of the tactics.

The participants completed a 15-item multiple choice questionnaire (see Appendix) immediately after studying the tactics. The questionnaire tested the subjects' understanding of the UAV's behaviour in specific situations addressed by the tactics design. The subjects could consult the design at any time while completing the questionnaire. Time taken to complete the questionnaire was recorded.

6.2.2. Results

As predicted (H_1: $\mu_1 > \mu_2$), the mean number of questions answered correctly was higher for the TDF group: TDF$\mu_1 = 82.4\%$, UML$\mu_2 = 66.4\%$. A one-tailed, independent *t*-test rejects the null hypothesis, H_0: $\mu_1 = \mu_2$, $p < 0.025$.

There was no significant difference in the time taken to complete the questionnaire between the two groups: $\mu_1 = 13.4$, $\mu_2 = 13.6$ (minutes).

6.2.3. Analysis

The experimental evaluation indicates that TDF's approach to tactics representation is more readily understood than equivalent UML versions. 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 interruptible behaviour, 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. A maintenance goal comprises a maintenance condition that specifies a particular state of the world that the agent must maintain, for example, ensuring that there is enough fuel to return to base. If the maintenance condition becomes untrue, then a goal is adopted to bring the world back into compliance, for example, by refuelling. Maintenance goals can be succinctly expressed in TDF, but are not straightforward to specify in UML because it does not provide a specific goal maintenance design artefact.

7. Discussion

TDF supports the design of autonomous, tactical decision-making systems that operate in dynamic domains. Such systems must be able to balance goal-directed behaviour with an ability to respond to important environmental change. It has been argued that AOSE maps well to such problems; however with regard to tactical decision-making, previous approaches to AOSE fall short in terms of high-level

tasking (missions), goal structures and specifying how to achieve goals.

The original motivation for the development of TDF came from a team of analysts who have been using UML to model USW tactics over a period of many years. This approach did not scale to larger tactics libraries, and sharing and reuse across analysts was problematic. Indeed, our experience of modelling military tactics over the last 20 years suggests that model reuse causes difficulties, particularly when sharing across team members. It is not unusual for a developer to prefer to implement a new tactical model from scratch, rather than try to understand and reuse another's model.

TDF extends Prometheus with missions, a wider range of goals structures, plan diagrams and tactics design patterns. These extensions are intended to make autonomous decision-making designs easier to understand and share. Ease of comprehension was evaluated in the context of a UAV photoreconnaissance scenario. The results of the evaluation indicate that TDF designs are easier to understand than corresponding UML versions. This effect was significant, despite the fact that the designs were deliberately simplified to reduce the inherent advantage that TDF has with regard to activities that can be interrupted by environmental events.

We believe that TDF will also be useful in applications that rely upon behaviour models in the form of "virtual actors" that must be embedded in simulation environments incorporating human trainees, as well as computer games and so-called serious games (applications other than entertainment, e.g. training). With humans involved, there is increased scope for variation which means that the behaviour models must provide broad coverage to handle less frequent use cases. This makes attention to tactics at a design level all the more important. There are plenty of such applications, for example, disaster management or search-and-rescue. Tactics design patterns will help with reuse, for example, if models of a doctor and a nurse have already been built, their design patterns can form the starting point for specifying a paramedic virtual actor. To this end, we have recently integrated the TDF tool with VBS (Virtual Battlespace), the leading military 3D, photo-realistic virtual environment. As is common with virtual environments that focus on visual realism, VBS's inbuilt models of human behaviour are fairly rudimentary, and its scripting language is not well suited to the task of implementing dynamic and flexible tactics. Our TDF/SIM infrastructure uses TDF's JACK code generation capability to interface to VBS, and allows the design and implementation of dynamic tactics that drive the behavioural entities in VBS. Further details can be found in Evertsz et al. (2015).

A major contribution of this research has been the investigation of the requirements for modelling tactical decision-making, and what is lacking in existing AOSE methodologies. Tactics have to mix reactive and proactive modes of computation, and an effective design methodology should provide a means of highlighting this aspect in the early stages of the system's specification. TDF achieves this through its provision of conditional goals, that allow reactive/proactive behaviour to be expressed at the goal level rather than at the procedural level; the latter occurring during the later detailed design, rather than system specification. TDF's application of the concept of *design pattern* to tactics is another novel contribution to the field that promises to foster the creation of large, reusable and platform-independent tactics libraries. This is a pressing need in the field of autonomous decision-making systems, and in the more general area of behaviour modelling. TDF's mission concept provides an important context for the design and development of tactics, as well as a potential means of categorising the tactics library in terms of the missions that can be handled. Finally, the provision of plan diagrams allows procedural aspects to be expressed at a high level of abstraction, one that has the potential to be understood and critiqued by SMEs.

7.1. The way forward

There is considerable scope for extending the TDF methodology and tool to handle a wider range of modelling requirements. Areas for further work include:

- *Formalisation of descriptors*:Currently, TDF is mainly a descriptive design tool, although it can also generate skeleton JACK and GORITE code. In the future, a number of TDF properties could be formalised. For example, the Outcomes property of tactics design patterns could be changed from a natural language description to a formal, logical expression that defines how the world changes during and after the execution of the tactic. This would provide an opportunity for automated detection of conflicts, for example, a tactic with the outcome of surfacing would conflict with a mission that has the restriction that the submarine remain submerged at all times. However, caution should be exercised when moving in the direction of formalisation. Our experience with users suggests that, for many developers, the inclusion of such formal elements in the design is a burden. There is a delicate trade-off between using natural language descriptions of property values vs. formal expressions. This could be overcome by allowing the expression of both formal and informal property values. The user could use formal expressions if automated design analysis is required, falling back on natural language descriptions if automated analysis is not required or if formalisation would significantly slow down the design process.
- *Team modelling*:Because TDF was originally developed to model individual submarine commander tactics, it does not explicitly model team structures. Nevertheless, teams can be modelled through the use of the role and message design artefacts. We are planning to extend TDF to handle team-based modelling of explicit team structures.
- *Completeness checking*:In BDI systems, the goal structure is implicitly represented by plans. The mapping of goals to plans in TDF could be automatically checked to ensure that the goal structures are reflected by the set of plan diagrams, highlighting goals that have no procedural embodiment as plans. This could be implemented via some form of abstract interpretation (Cousot and Cousot, 1977) of the plan graphs to derive the implicit goal structure. Related work on the automated verification of plan diagrams against scenarios and goal structures (Abushark et al., 2015) could be applied here.
- *Tactics ontology*:Tactics reuse would be enhanced by providing an editable ontology for tactics. Reasoners in modern ontology editors allow set membership to be automatically determined via class properties. With an appropriate ontology, tactics design patterns could be automatically classified based on their properties. For example, automatically identifying tactics that involve "lethal force".
- *Environment modelling*:Following Prometheus, TDF models the environment in terms of actors, percepts and actions. However, this could be extended to encompass more recent AOSE-based research on *artefacts*, which treats environmental entities as first class abstractions, to be modelled in similar detail to agents, e.g. Belardinelli et al. (2013). Furthermore, the development of an ontology for the environment would facilitate the design of tactics at a level that abstracts away the low-level details of the environment, facilitating integration with different platforms.
- *Broader code generation targets*:Currently, the TDF tool generates JACK and GORITE codes, which through the use of appropriate plugins, can interface to sensor and actuator APIs, as well as simulation platform APIs (e.g. the VBS integration referred to earlier). The automated link from design to implementation is an important one, and extending TDF's code generation capabilities will

broaden its applicability, allowing it to be used, for example, for safety-critical applications that must conform to RTCA/DO-178A (currently, the most stringent software certification requirements for airborne systems).

– *Knowledge elicitation*:Most tactical decision-making systems are designed to emulate the reasoning of human experts, for example, pilots or infantry commanders. This entails a period of knowledge elicitation with SMEs before the required tactics can be designed. We are currently extending the TDF methodology to encompass knowledge elicitation in a way that provides traceability from elicitation, through requirements, to design and implementation.

Acknowledgements

This work was supported by the Defence Science and Technology Organisation, and the Defence Science Institute.

Appendix

Multiple choice questionnaire

1. **If the mission specifies the route, will the UAV determine the no fly zones?**
   – Yes, No, Only if the Use Mission Route goal succeeds, Undefined
2. **What happens if the UAV is not told that it has reached the next waypoint?**
   – Returns to base if fuel is low, Keeps flying in the same direction, Requests position update, Undefined
3. **The UAV would choose a Creeping Line search pattern when:**
   – Target is far away, Never, Target is travelling in a straight line, Target direction is known, Undefined
4. **Why would the UAV suddenly start to climb?**
   – Avoid turbulence, Convoy found, Incoming missile, Undefined
5. **Incoming missile. What does the UAV do if there are no countermeasures remaining?**
   – Launch flares, Enable infrared countermeasures, Evasive manoeuvres, Handles incoming missile, Uses countermeasures, Undefined
6. **If there is low cloud and a missile is detected, the UAV:**
   – Climbs, Descends, Neither, Undefined
7. **When can the UAV enter a No Fly Zone?**
   – When attacked from the No Fly Zone, When low on fuel, Never, Undefined
8. **If attacked from above, the UAV turns before it descends:**
   – True, False, Sometimes
9. **If attacked from the ground, the UAV completes evasive manoeuvres when the following actions have been completed:**
   – Turn, Climb, Accelerate, Any of these, All three, Undefined
10. **When there is an incoming missile, which is done first?**
    – Launch flares, Enable infrared countermeasures, Simultaneous, Undefined
11. **While following the route, what does the UAV do if low on fuel?**
    – Descends, Climbs, Stops following route, Returns to base, Undefined
12. **If descending below low cloud, what happens if the UAV is below 500m?**
    – Levels off, Fails to descend below low cloud, Descends until below low cloud, Undefined
13. **If the UAV does not determine a route, which route does it follow?**
    – Mission route, Navigates in general direction of target, Does not follow route, Undefined
14. **What is the outcome of photo reconnaissance?**
    – UAV returns to base, Photograph taken, UAV has less fuel, Undefined
15. **If an incoming missile is detected, does the UAV abandon the mission objective?**
    – Yes, No, Only until the missile has been evaded, Undefined

References

Abushark, Y., Thangarajah, J., Miller, T., Harland, J., Winikoff, M., 2015. Early detection of design faults relative to requirement specifications in agent-based models. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1071–1079.

Akbari, O.Z., 2010. A survey of agent-oriented software engineering paradigm: towards its industrial acceptance. J. Comput. Eng. Res. 1 (2), 14–28.

Ali, R., Chitchyan, R., Giorgini, P., 2009. Context for goal-level product line derivation. In: 3rd International Workshop on Dynamic Software Product Lines (DSPL). Carnegie Mellon University, Pittsburgh, PA, pp. 8–17.

Aridor, Y., Lange, D.B., 1998. Agent design patterns: elements of agent application design. In: Proceedings of the Second International Conference on Autonomous Agents. ACM, pp. 108–115.

Bauer, B., Müller, J.P., Odell, J., 2001. Agent UML: a formalism for specifying multiagent interaction. In: Agent-oriented software engineering, 1957. Springer, Berlin, pp. 91–103.

Bauer, B., Odell, J., 2005. Uml 2.0 and agents: how to build agent-based systems with the new uml standard. Eng. Appl. Artif. Intell. 18 (2), 141–157.

Belardinelli, F., Lomuscio, A., Patrizi, F., 2013. Verification of agent-based artifact systems. J. Artif. Intell. Res. 51, 333–377.

Benfield, S.S., Hendrickson, J., Galanti, D., 2006. Making a strong business case for multiagent technology. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. ACM, pp. 10–15.

Botti, V., Giret, A., 2008. ANEMONA: A Multi-agent Methodology for Holonic Manufacturing Systems, 1st edition Springer Publishing Company, Incorporated.

Bratman, M.E., 1987. Intention, Plans, and Practical Reasoning. Harvard University Press, Cambridge, MA (USA).

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

Chandrasekaran, B., 1986. Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. IEEE Expert 1 (3), 23–30.

Cossentino, M., 2005. From requirements to code with the PASSI methodology. In: Agent-Oriented Methodologies, 3690. Springer, pp. 79–106.

Cossentino, M., Gaud, N., Hilaire, V., Galland, S., Koukam, A., 2010. ASPECS: an agent-oriented software process for engineering complex systems. Auton. Agents Multi-Agent Syst. 20 (2), 260–304.

Council, N.R., 2012. NASA Space Technology Roadmaps and Priorities: Restoring NASA's Technological Edge and Paving the Way for a New Era in Space. Technical Report. National Research Council.

Cousot, P., Cousot, R., 1977. Abstract interpretation: a unified framework for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages. Los Angeles, California, pp. 238–252.

DeLoach, S., Padgham, L., Perini, A., Susi, A., 2009. Using three AOSE toolkits to develop a sample design. Int. J.Agent-Oriented Softw. Eng. 3 (4), 416–476.

DeLoach, S.A., Garcia-Ojeda, J.C., 2010. O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. Int. J. Agent-Oriented Softw. Eng. 4 (3), 244–280.

Dennett, D.C., 1987. The Intentional Stance. MIT press.

van Der Aalst, W.M., Pesic, M., Schonenberg, H., 2009. Declarative workflows: balancing between flexibility and support. Comput. Sci.-Res. Dev. 23 (2), 99–113.

d'Inverno, M., Kinny, D., Luck, M., Wooldridge, M., 1998. A formal specification of dMARS. Intelligent Agents IV Agent Theor., Archit. Lang. 1365, 155–176.

Dopping-Hepenstall, L., 2013. Integrating UAS into airspace 2014 – the view from ASTRAEA. In: IET Seminar on UAVs in the Civilian Airspace. The Institution of Engineering and Technology, London, UK, pp. 1–29. doi:10.1049/ic.2013.0064.URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6616086

Doyle, R.J., 2003. Autonomy Needs and Trends in Deep Space Exploration. Technical Report. DTIC Document.

Duff, S., Thangarajah, J., Harland, J., 2014. Maintenance goals in intelligent agents. Comput. Intell 30 (1), 71–114.

Endert, H., Küster, T., Hirsch, B., Albayrak, S., 2007. Mapping BPMN to agents: an analysis. Agents, Web-Services, and Ontologies Integrated Methodologies. MALLOW, pp. 43–58.

Endsley, M.R., 1988. Design and evaluation for situational awareness enhancement. In: Proceedings of the Human Factors Society 32nd Annual Meeting. Human Factors Society, Santa Monica, CA, pp. 97–101.

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

Evertsz, R., Thangarajah, J., Ambukovski, N., 2015. Using agent-based tactics models to control virtual actors in VBS3 (demonstration). In: Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems, pp. 1929–1930.

Federico, B., Marie-Pierre, G., Franco, Z. (Eds.), 2004. Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Springer.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design patterns: Elements of reusable object-oriented design. Addison-Wesley Reading.

Georgeff, M.P., Ingrand, F.F., 1989. Monitoring and control of spacecraft systems using procedural reasoning. In: Proceedings of the Space Operations Automation and Robotics Workshop.
Georgeff, M.P., Lansky, A.L., 1985. 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.
Gilbreth, F.B., Gilbreth, L.M., 1921. Process Charts - First Steps in Finding the One Best Way to do Work. American Society of Mechanical Engineers, New York.
Glass, R.L., Vessey, I., Ramesh, V., 2002. Research in software engineering: an analysis of the literature. Informat. Softw. Technol. 44 (8), 491–506.
Haddon, D.R., Whittaker, C.J., 2003. Aircraft airworthiness certification standards for civil uavs. Aeronaut. J. 107 (1068), 79–86.
Heaton, L., 2005. Unified Modeling Language (UML): Superstructure Specification, v2. 0. Tech. Rep. Object Management Group.
Huntsberger, T., Woodward, G., 2011. Intelligent autonomy for unmanned surface and underwater vehicles. In: OCEANS 2011, pp. 1–10.
JARUS, 2011. JARUS II: AMC UAS.1309, Draft, Issue B. Technical Report. UAS Systems Safety Analysis 1309 Group.
Juan, T., Pearce, A., Sterling, L., 2002. 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. ACM, pp. 3–10.
Karim, S., Heinze, C., Dunn, S., 2004. Agent-based mission management for a UAV. In: Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004. IEEE, pp. 481–486.
Kinny, D., Georgeff, M., Rao, A., 1996. A methodology and modelling technique for systems of BDI agents. In: Proceedings of the European workshop on Modelling autonomous agents in a multi-agent world: Agents Breaking Away, pp. 56–71.
Laird, J.E., Jones, R.M., Jones, O.M., Nielsen, P.E., 1994. Coordinated behavior of computer generated forces in tacair-soar. In: In Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation. Citeseer.
Lange, D.B., Mitsuru, O., 1998. Programming and Deploying Java Mobile Agents Aglets. Addison-Wesley Longman Publishing Co., Inc.
McDermott, J., 1988. Preliminary steps toward a taxonomy of problem-solving methods. In: Automating knowledge acquisition for expert systems. Springer, pp. 225–256.
Michon, J.A., 1985. A critical view of driver behavior models: What do we know, what should we do? Human Behavior and Traffic Safety. Plenum Press, New York, pp. 485–520.
Mills, N., 2011. Opening the airspace for UAVs – ASTRAEA progress report. In: Knörzer, D., Szodruch, J. (Eds.), Innovation for Sustainable Aviation in a Global Environment: Proceedings of the Sixth European Aeronautics Days. IOS Press, Madrid, pp. 394–398.
Murray, G., Steuart, D., Appla, D., McIlroy, D., Heinze, C., Cross, M., Chandran, A., Raszka, R., Tidhar, G., Rao, A., Pegler, A., Morley, D., Busetta, P., 1995. The challenge of whole air mission modelling. In: Proceedings of the Australian Joint Conference on Artificial Intelligence.Melbourne, Australia
Muscettola, N., Nayak, P.P., Pell, B., Williams, B.C., 1998. Remote agent: to boldly go where no AI system has gone before. Artif. Intell. 103 (1), 5–47.
NFA Autonomy Working Group, 2012. Autonomous Systems: Opportunities and Challenges for the Oil & Gas Industry. Technical Report.
Office of the Secretary of Defense, 2001. Unmanned Aerial Vehicles Roadmap 2000-2025. Technical Report. Washington DC.
OMG Business Process Modeling Notation, 2006. Version 1.0, OMG Final Adopted Specification. Object Management Group.
Padgham, L., Thangarajah, J., Winikoff, M., 2008. Prometheus design tool. In: Proceedings of the 23rd AAAI Conference on AI, pp. 1882–1883.
Padgham, L., Winikoff, M., 2004. Developing Intelligent Agent Systems: A Practical Guide. Wiley.
Padgham, L., Winikoff, M., 2005. Prometheus: A practical agent-oriented methodology. In: Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-Oriented Methodologies. Idea Group, Hershey, PA, pp. 107–135.
Padgham, L., Winikoff, M., DeLoach, S., Cossentino, M., 2009. A unified graphical notation for AOSE. In: Agent-Oriented Software Engineering IX. Springer, pp. 116–130.
Pavón, J., Gómez-Sanz, J., 2003. Agent oriented software engineering with INGENIAS. In: Multi-Agent Systems and Applications III. Springer, pp. 394–403.
Petri, C.A., 1966. Communication with automata: Volume 1 supplement 1. Technical Report. DTIC Document.
Rönnquist, R., 2008. The goal oriented teams (GORITE) framework. In: Programming Multi-Agent Systems. Springer, pp. 27–41.
Taylor, G., Wray, R.E., 2004. Behavior design patterns: Engineering human behavior models. In: Proceedings of the Behavior Representation in Modeling and Simulation Conference.
Wallis, P., Rönnquist, R., Jarvis, D., Lucas, A., 2002. The automated wingman-using JACK intelligent agents for unmanned autonomous vehicles. In: Aerospace Conference Proceedings, 2002. IEEE, 5. IEEE, pp. 5–2615.
Wegener, S.S., Schoenung, S.M., Totah, J., Sullivan, D., Frank, J., Enomoto, F., Frost, C., Theodore, C., 2004. UAV autonomous operations for airborne science missions. In: Proceedings of the American Institute for Aeronautics and Astronautics 3rd Unmanned Unlimited Technical Conference.
Winikoff, M., 2005. JACK intelligent agents: an industrial strength platform. Multi-Agent Programming. Springer, pp. 175–193.
Wooldridge, M., 2008. An Introduction to Multiagent Systems. Wiley.
Wooldridge, M., Jennings, N.R., Kinny, D., 2000. The Gaia methodology for agent-oriented analysis and design. Auton. Agents Multi-Agent Syst. 3 (3), 285–312.

Rick Evertsz is currently at RMIT University, and has over 20 year experience in agent-oriented analysis, design and development in areas including real-time optimisation of air traffic flow, network fault diagnosis, and military behaviour modelling. He is also interested in cognitive modelling and the development of cognitive architectures.

John Thangarajah is an Associate Professor in Artificial Intelligence at RMIT University, Australia. His main research interests are in Agent Oriented Software Development, Agent Reasoning, Intelligent Conversation Systems, Agent Testing, and Intelligent Games.

Nitin Yadav received his PhD in Computer Science from RMIT University in 2014. His current research interests include: behaviour composition, temporal logics, model checking, agent-oriented programming languages, and BDI-style logics.

Thanh Ly is with the Defence Science and Technology Organisation's Joint and Operations Analysis Division. His research interests include agent-based modelling, tactical simulation, and evolutionary computing.