Threat Analysis Using Goal-Oriented Action Planning

Philip Bjarnolf
2 PUBLICATIONS **8** CITATIONS
SEE PROFILE

Per M. Gustavsson
George Mason University
78 PUBLICATIONS **405** CITATIONS
SEE PROFILE

Christoffer Brax
SAAB Group
20 PUBLICATIONS **130** CITATIONS
SEE PROFILE

Threat Analysis Using Goal-Oriented Action Planning

Philip Bjarnolf
University of Skövde
Philip.Bjarnolf@student.his.se

Per M. Gustavsson
Christoffer Brax
Mikael Fredin
Saab
Training Systems and Information Fusion
Storgatan 20, 541 30 Skövde, Sweden
<name>@saabgroup.com

Keywords:
Threat Analysis, Goal-Oriented Action Planning, STRIPS, Planning Systems, Decision Support, AI, Coalition Battle Management Language, Commander's Intent, Information Fusion.

ABSTRACT:
Agile Computer Generated Forces (CGF) for the Next Generation Training Systems and Planning Systems need to have a more agile and flexible behavior than the current commonly used CGF systems can deliver. This because of that the human behavior in a real crisis or military operation is adaptive. This paper describes one approach to enable such functionality by combining Threat Analysis and Goal-Oriented Action Planning (GOAP) into an architecture where the concepts and formalism of effects and Actions plays a central role.

Threat assessment is a vital part in everyday life as it enables us to prevent or evade dangerous or otherwise unwanted events. Through threat assessment we evaluate the elements of our world as we perceive and conceive it. These elements can be seen as key building blocks of the world, and depending on the situation, more or less of these keys have a direct impact on the outcome of a certain scenario taking place. An entity capable of assessing its and others action capabilities possess the power to predict how the involved entities may change their world. Through this knowledge and higher level of situation awareness, the assessing entity may choose the actions that have the most suitable effect, resulting in that entity's desired world state.

This paper covers aspects and concepts of an arbitrary planning system and presents a threat analyzer architecture built on the novel planning system Goal-Oriented Action Planning (GOAP). The realized GOAP architecture use two agents that perform action planning to reach their desired world states. One of the agents employs a modified GOAP planner used as a threat analyzer in order to determine what threat level the adversary agent constitutes. This paper does also introduce a conceptual schema of a general planning system that considers orders, doctrine and style and its representation in the Command and Control Lexical Grammar (C2LG); as well as a schema depicting an agent system using a blackboard in conjunction with the OODA-loop.

1. Introduction

Threat assessment is a vital part in everyday life as it enables us to prevent or evade dangerous or otherwise unwanted events. Through threat assessment we evaluate the elements of our world as we perceive and conceive it. These elements can be seen as key building blocks of the world, and depending on the situation, more or less of these keys have a direct impact on the outcome of a certain scenario taking place. In the eyes of the beholder, or even better in the mind of the interpreter, the relevant key parts of the world are registered and in one way or the other given a certain value or reference.

As our world, depending on the level of detail we wish to apply, has what seems to be an almost endless amount of world states, it would be impractical, unnecessary or even impossible to keep track of and model each and every world state. It therefore seems intuitive and wise to only model those world states deemed necessary for the scenario or context at hand.

Once the relevant world state keys have been identified their paired value may be changed by an arbitrary *action* that causes a world state transformation to occur, that is to say that the action has an *effect*. An entity capable of an arbitrary set of actions is by default only capable of changing the world states according to the effects of those available actions. However, this is not to say that the entity is unable to realize world states it has not planned for. Since it is perfectly possible to perform an action without knowing all of its consequences, i.e. effects or world state transformations, an entity may very well experience how things unfold in unseen and sometimes unwanted ways. This also means that in order for a threat assessing entity to comprehend a threat, the entity must be able to picture how a series of actions could realize an unwanted world state. An entity's insight of its and others action capabilities is thus tightly coupled with the entity's threat analysis capabilities. This concept is heavily used in this paper which aims to analyze and develop a threat analysis planning architecture built on Goal-Oriented Action Planning (GOAP). By slightly modifying and thereby extending the GOAP architecture, a higher level of situation awareness may be obtained; enabling entities using a threat analyzer to employ better strategies and decisions. Further the paper provides recommendations to the ongoing standardization of the Coalition Battle Management Language (C-BML) <<REFERENS>> and the development of a Command and Control Lexical Grammar (C2LG) <<REFERENS>> regarding what needs to be able to be expressed to use the suggested methods and technologies described in this paper. A small example is provided where two agents utilize the GOAP technology for their action planning, where one agent also uses a modified planner acting as a threat analyzer, capable of assessing its adversary's threat level in the simulated scenario.

2. Artifacts of Planning

How decision making agents perceive and conceive their environment and the "knowing of what is going on" (Endsley, 1995, pp. 37) can be referred to as situation awareness (SA), which is highly involved with how well an agent is able to cope with its environmental challenges. For example, a blind and deaf agent that participates in a game of hide and seek, has far lower SA than an unimpaired counterpart; who on the other hand, has lower SA than an agent that can communicate with an airborne teammate who shares its bird eye view of the situation. This plain example shows how an agent's SA can be increased by sharing the information of the situation at hand to other agents. The definition of SA and its build-up can be theoretically explained or thought of in numerous ways, as done by Lagervik & Gustavsson (2006), Endsley (1995), Ackoff (1999), and Bedney & Meister (1999). However, the main parts involved are without doubt the agent's sensory system, its memory, and its ability to interpret, understand, and mediate to others its subjective impressions and gathered knowledge.
By centralizing this SA or knowledge through a blackboard, as suggested by Orkin (2006), all participating agents become an entity with the same basic world comprehension, even though individual styles may interpret the information in different ways.

A conceptual schema of an arbitrary planning system is presented in Figure 1. The schema shows how an agent's style influences the inputs from the world, broken up into sensed orders and world states. As the sensed data is passing through the personality layer, it gets styled according to the agent's personality, after which the brain, i.e. reasoning and planning system, is able to draw a conclusion and formulate the most promising plan; which in turn changes the world into or towards a desired state.

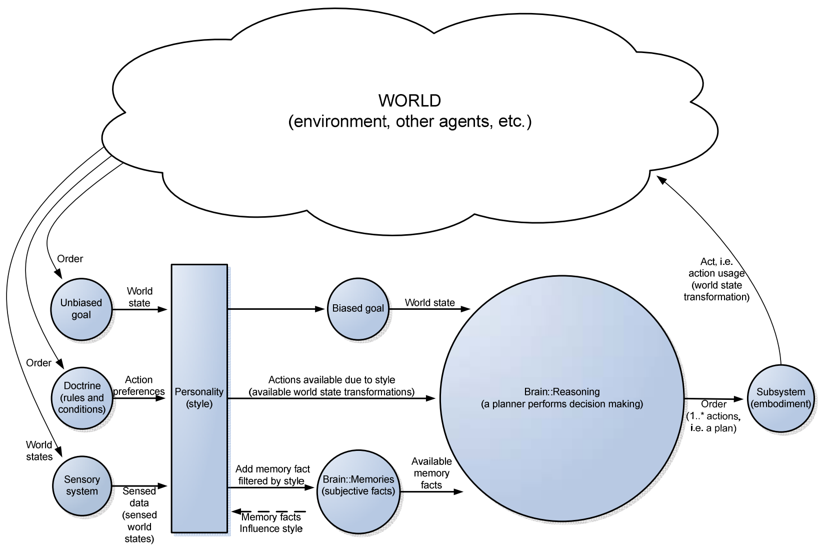

WORLD
(environment, other agents, etc.)
Order
Order
World states
Unbiased goal
World state
Doctrine (rules and conditions)
Action preferences
Sensory system
Sensed data (sensed world states)
Personality (style)
Biased goal
World state
Actions available due to style (available world state transformations)
Add memory fact filtered by style
Memory facts Influence style
Brain::Memories (subjective facts)
Available memory facts
Brain::Reasoning (a planner performs decision making)
Order (1..* actions, i.e. a plan)
Subsystem (embodiment)
Act, i.e. action usage (world state transformation)

Figure 1 – Conceptual Schema of a Planning System

All actions undertaken by agents are consequences of a desire to achieve certain goals. As these goals or tasks might have been proposed by another commanding agent, the medium or technique mediating the task has a great responsibility to ensure that it is not altered or perceived in an inaccurate way by the agent supposed to execute the task (i.e. the taskee). By having the commander only telling *what* he wants to be done instead of *how*, the taskee is given the authority to execute (and formulate) a plan of its own (Kleiner, Carey & Beach, 1998). The concept of a correct picture of the task or desired end-state, as first formulated by the commander, is referred to as Commander's Intent (CI).

In order to achieve a goal, a *plan* consisting of actions has to be formulated and executed by the agent. The plan is thus nothing more than a set of actions that has to be executed in a correct order with certain timing, consequently causing one or more *effects* which results in an alteration of a world state. In Figure 1 the plan formulating system, i.e. the reasoning brain of an agent, receives styled data input from the world. Depending on the current goal, actions at hand, and past experiences, the plan formulating system outputs an action or a series of actions (i.e. a plan) that are realized by a subsystem of the agent; consequently changing a world state towards or into the desired end-state.

The plan formulating system is called a *planner*, and is in other words allowing agents to *formulate their own action sequence* in order to accomplish a certain goal. Orkin (2004c, pp. 222) states that "*in order to formulate a plan, the planner must be able to represent the state of the world in a compact and concise form*", which incorporates identifying the most important variables of the world where the current scenario is taking place. This means that even though the actual world state consists of a myriad of variables, only the most significant variables should be used in order to keep the planning as simple and fast as possible.

2.1. Effect Based Planning

Effect based planning (EBP) aims to clarify the effects in different areas regarding the actions incorporated into a plan. By utilizing the EBP concept, negative consequences are to be foreseen and avoided or worked against in order to achieve a desired goal with as few side-effects, i.e. unintended effects, as possible. EBP has a strong connection to Commander's Intent since the planning entities must fully understand (or share) the CI in order to formulate a plan with actions causing the desired effects according to the CI. As stated by Farrell (2004), CI is central to EBP. FOI, the Swedish Defense Research Agency, has through Schubert, Wallén & Walter (2007) presented an EBP technique using a cross impact matrix (CIM) which "…is to find inconsistencies in plans developed within the effect based planning process." (Schubert et al., 2007, pp. 1) using Dempster-Shafer theory and sensitive analysis. The CIM consists of all *activities* (A), *supporting effects* (SE), *decisive conditions* (DC) and *military end state* (MES) of the plan, and should be seen as a dynamic entity built and continually managed by a broad working group with a strong knowhow of the various matrix components and their interaction (Schubert et al., 2007). By employing this CIM architecture, any weaknesses and all strengths of the plan can be found prior to the effect based execution phase, giving the involved participants a more similar understanding of the situation leading to better decisions being made. The actual plan formulated through EBP is described by Schubert et al. (2007) as a tree structure having the MES as root. An example of such plan is presented in Figure 2.

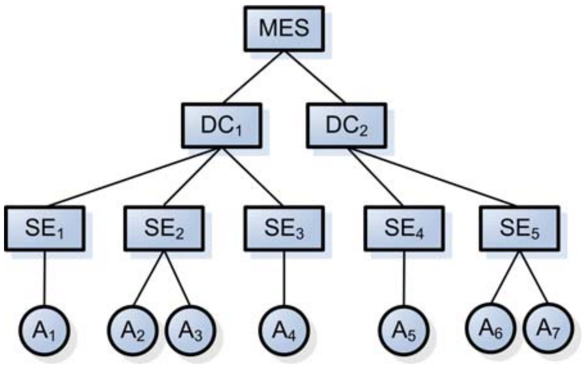


Figure 1. The plan formed according to EBP (after Schubert et al., 2007, pp. 2).

Comparing this structure in this context to others, one can imagine the similarities of a composite/atomic goal breakdown by Buckland (2005), or composite/simple task and action described by Dybsand (2004). In that sense DC could be seen as a composite goal or task, while SE maps to a composite goal or simple task[1], and A maps to an atomic goal or action. Regardless of the naming convention, MES is without doubt reached by the occurrence of A, SE and DC.

As effects from actions are modeled in GOAP, the planning taking place could somewhat be seen as EBP since the known effects may judge what actions get incorporated into the plan.

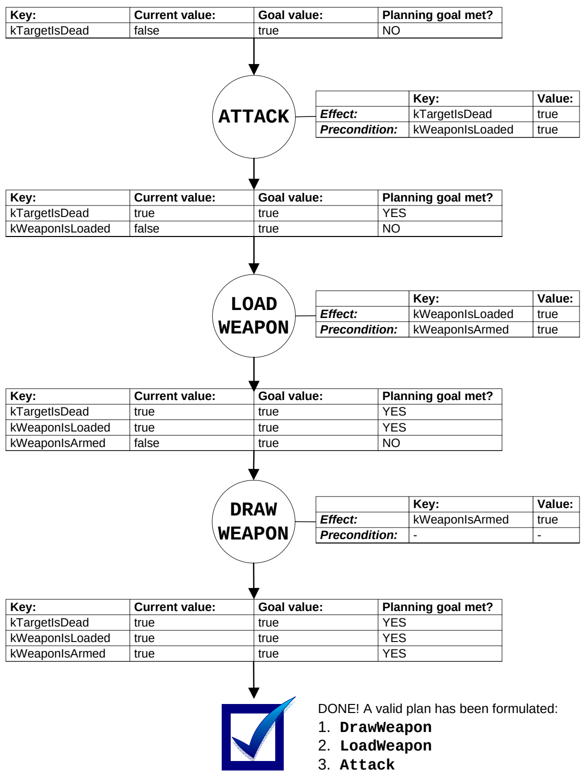
[1] *Composite* and *simple* tasks are sometimes also referred to as *compound* and *primitive* tasks (Erol, Hendler & Nau, 1994; Hoang, Lee-Urban & Muñoz-Avila, 2005).

3. GOAP

Goal-Oriented Action Planning stems from discussions from the GOAP working group of the A.I. Interface Standards Committee (Orkin, 2006), and has so far only described the high-level concepts of GOAP. It is based on STRIPS (STanford Research Institute Problem Solver) which is an automated planner consisting of an initial state and goal states, which the planner is trying to reach by using actions that in turn have pre- and postconditions (Fikes & Nilsson, 1971). Long (2007) explains that the actions in GOAP stems from operators in STRIPS, and the primarily differences between the two techniques are the adaption of action costs in GOAP along with the use of the A* algorithm[2]. Much like the re-planning capabilities of the Goal-Driven Agent Behavior described by Buckland (2005, pp. 385), GOAP also incorporates a kind of re-planning feature, enabling the planner to construct a new plan consisting of other, but still valid, actions leading to the pursued goal world state (i.e. the end-state). The notion of using remembered (old) planning data and avoid planning from scratch, is called Case-Based Planning (Ontañón, Mishra, Sugandh & Ram, 2007), and could be incorporated in GOAP. The goals and actions available in a scenario using GOAP are static, i.e. no other goals and actions than those predetermined by the developer are ever used. **Error! Reference source not found.** depicts a planning example where the goal state `(kTargetIsDead, true)` can be reached by executing the actions `DrawWeapon`, `LoadWeapon` and `Attack`.

Orkin (2004c) suggests that the relevant world states are represented as a list of structures containing an enumerated attribute key, a value, and a handle to a subject. A specific action's or *operator's* preconditions and effects therefore has the same structure; both a precondition and an effect has a key and a value[3], e.g. one action called `GoToSleep` might have a precondition with the key `kBedlampIsOn` with the associated value `false`, and the effect might have the key `kIsTired` with the value `false`. An action can have multiple preconditions and effects, i.e. the action `GoToSleep` in the previous example may have additional preconditions and effects other than `(kBedlampIsOn, true)` and `(kIsTired, false)`. The actions are thus sequenced by satisfying each others preconditions and effects, and by doing so a plan pursuing the current goal (i.e. the desired end-state) is formed. Orkin (2004c, pp. 218) sees it as "each action knows when it is valid to run, and what it will do to the game world". It should be noted that an action only has the preconditions of interest to it.

Besides having a symbolic precondition or effect, an action may have a context precondition or effect, where the resulting value is dynamically obtained by an arbitrary function implemented by a certain subsystem. For example, the value of whether a threat is present could be directly retrieved from the world state `(kThreatIsPresent, true)` or from a context function `EvaluateThreatPresence` that returns a suitable value or handle. Long (2007, pp. 9) mentions two benefits of using context preconditions: "It improves efficiency by reducing the number of



Key:	Current value:	Goal value:	Planning goal met?
kTargetIsDead	false	true	NO

	Key:	Value:
Effect:	kTargetIsDead	true
Precondition:	kWeaponIsLoaded	true

Key:	Current value:	Goal value:	Planning goal met?
kTargetIsDead	true	true	YES
kWeaponIsLoaded	false	true	NO

	Key:	Value:
Effect:	kWeaponIsLoaded	true
Precondition:	kWeaponIsArmed	true

Key:	Current value:	Goal value:	Planning goal met?
kTargetIsDead	true	true	YES
kWeaponIsLoaded	true	true	YES
kWeaponIsArmed	false	true	NO

	Key:	Value:
Effect:	kWeaponIsArmed	true
Precondition:	-	-

Key:	Current value:	Goal value:	Planning goal met?
kTargetIsDead	true	true	YES
kWeaponIsLoaded	true	true	YES
kWeaponIsArmed	true	true	YES

DONE! A valid plan has been formulated:
1. **`DrawWeapon`**
2. **`LoadWeapon`**
3. **`Attack`**

Figure 3. The planner finds a valid plan by a regressive search (after Orkin 2004c,pp 225).

potential candidates when searching for viable actions during planning and it extends the functionality of the actions beyond simple Boolean world state symbols.".
In GOAP, each action has a static cost value which helps the planner to determine what actions are more favorable than others, i.e. the planner considers more specified actions before more general ones (Orkin, 2005). A* search is used to obtain the best path of valid actions which results in a plan. However, the use of static values has been criticized as they never change

[2] The A* algorithm is described in detail by Buckland (2005, pp. 241).

[3] The value is either a integer, float, bool, handle, enum, or a reference to another symbol.

during the execution of the application (Long, 2007), making it harder to formalize a more expressive plan.

For example, the action `SupportSuppressedSquadMember` probably should have a lower cost, i.e. be more favorable, if the supporting member is not under fire. These cost values resemble those mentioned by Reynolds (2002), who suggests using a priority system where different actions and circumstances are given different priorities. He furthermore states that "These [priority] values allow soldiers to be chosen by their current status, and for soldiers of a certain rank or health to be omitted unless their status is of a lower priority.".

Thanks to the general approach of GOAP it can be applied to virtually any problem scenario, while still having the benefits of a reasoning system that has an understanding of how goals and decisions relate, and what the effects might be (Lagervik & Gustavsson, 2006). Three arbitrary examples are given where GOAP could be used as means of solving the problem at hand:

- **Baking a pie**. If the overall goal is to bake a pie the end-state could be `(PieIsBaked, true)`. The plan is then formulated based on the actions available and their costs, which for instance might include `TurnOnOwen`, `PutBowlOnTable`, `AddEggToBowl`, `AddButterToBowl`, `AddFlourToBowl`, `MixBowlIngredients`, `PutBowlInOwen`, `Wait`, and `PutBowlOnTable`. In this case the first action would be `TurnOnOwen` which has no preconditions, just as `PutBowlOnTable`. However the actions `AddEggToBowl`, `AddButterToBowl` and `AddFlourToBowl` might have the precondition `(BowlIsOnTable, true)` which is the very effect of the action `PutBowlOnTable`. If the mixture is supposed to fare better if the egg is used before that butter, which in turn is used before the flour; then `AddEggToBowl` is given the lowest cost value of the three actions whilst `AddButter` is given a lower cost value than `AddFlourToBowl`. The baking is then allowed to continue by utilizing the remaining actions of the plan, resulting in a baked pie once the final action `PutBowlOnTable` has been executed.
- **Telling a story**. If the overall goal is to reach the end-state `(StoryHasBeenTold, true)`, various subparts of the story may be linked by preconditions and effects which enables a dynamic story based on the variables at hand. For example, the story could start by reading a light sensor which acts as a precondition for one of the introduction subparts. After the introduction has been experienced, the user's pulse is checked which may act as a context precondition of some other subpart of the story, which may lead to less creepy story parts being told until the user is sufficiently calm.
- **Using a tutorial**. Much like the story telling example, a tutorial might be modeled using GOAP, allowing it to display hints according to the user's contextual preferences or other aspects of the scenario.

As seen from the examples above, there are many areas of application but GOAP has so far, what is known, not been particularly used in these novel areas. The reason for this is probably that the technique is new and has not yet been fully accepted, realized or understood by the market. Few official implementations of GOAP exists, and while there are some high-level guidelines, the very founders of GOAP have not yet created an interface standard. The GOAP architecture should thus be seen more as a guideline than a de facto standard. Because of this it seems that components incorporated in GOAP could be excluded as well as others being included if it serves the overall implementation.
An agent using GOAP has a *working memory* in which the agent's SA is updated and stored for later use in conjunction with a blackboard. The role of a blackboard in GOAP is to act as an intermediate information repository between various subsystems, which enable them to share data without having direct references to each other. Because certain agent sensors may have expensive computational costs, the computations are made over many frames and results are cached in the working memory (Orkin, 2005). The working memory's cached data is stored in the form of `WorkingMemoryFacts` which are divided into different types depending on the scenario at hand and coupled with the experienced confidence of the fact. A baking scenario might have a `Cupcake`, `DanishPastry`, `Event` and `Smell` as facts, whilst the combat scenario created by Long (2007) included facts like `Enemy`, `Friendly`, `PickupInfo` and `Event`. By employing these kinds of facts the planner is able to query the working memory in the best manner it may seem fit, e.g. by querying what `Cupcake` smell *most*. A strong advantage of GOAP identified by Orkin (2004a, pp. 1) is that "atomic goals and actions of a GOAP system are easy to read and maintain, and can be sequenced and layered to create complex behaviors"; the gain of simplicity and modularization makes it easier for multiple developers to collaboratively implement complex behaviors.

Layered behavior is accomplished by inheritance where subsequent behaviors or actions implement the same basic behavior as their parents. For example may an `Attack` action serve as an abstract parent to more specified children attack actions like `SneakAttack` and `ThrustAttack`.

By using GOAP, agents may make decisions not anticipated by the developer as the chain of actions (the plan) is made up at runtime (to the developer's joy or dismay). GOAP incorporates the use of finite state machines, but they are said not be as hardly coupled to the transition logic as an original Final State Machine (FSM) model. Instead of having the states themselves decide the transition, the various actions handle the transition logic (Orkin, 2004c). As stated by the Working Group on Goal-Oriented Action Planning (2004), the overall design of GOAP can be divided into three separate systems:

1) A goal selection interface that weighs up relevance, probabilities, and so on, and decides which goal to plan towards.
2) A planner that takes a goal and plans a series of actions to achieve it.
3) An action interface for carrying out, interrupting, and timing actions that characters can carry out.

Figure 4 depicts these subsystems as the goal selection interface is incorporated into the style layer that selects an appropriate goal world state depending on the personality of the agent. The planner that takes a goal and plans a series of actions to achieve it is depicted as a reasoning brain. The action interface is comprised in the embodiment of the agent or agents performing the suggested action, leading to the desired world state transition.

The Working Group on Goal-Oriented Action Planning (2003) recognize GOAP to have the ability to provide more flexible and diverse behaviors than FSMs and rule based systems, since plans are constructed "online" from atomic actions and adapted specifically to the current goal. Instead of a simple finite state machine reactively directing an agent in every situation, GOAP is employed to formulate a plan that strives to realize a certain goal world state, that in turn has been found by a goal selector. This goal selector may be an internal implementation as a set of functions, each representing a certain goal which depending on the scenario returns a goal relevancy value. The goal that returns the highest relevance value is pursued; however a cut-off implementation may stop the most promising goal from being pursued if its relevance value is too low, leaving the agent idle until a sufficient relevance value is obtained. Another implementation aspect is to externalize the goal selector completely, having the GOAP planner receiving an order containing the desired goal world state. Choosing goals by a relevancy value is mentioned by O'Brien (2002, pp. 379):

GOAP employs real-time planning by using regressive searches in order to find appropriate actions that satisfy the pursued goal. The plan formulating algorithm starts its search at the goal and then works its way towards the current state, by finding actions with the appropriate preconditions and effects. The search does not find all possible plans, but instead finds a valid plan that, thanks to the A* algorithm, is the most promising, i.e. cheapest plan. Because of the regressive search, GOAP can not deliver a valid incomplete plan as it does not contain the initial action step leading from the current state to the next, hence it is not an anytime algorithm4. For example, if the planning system was intercepted at stage 2, as illustrated by **Error! Reference source not found.**, the plan would only contain the action sequence leading from a certain state (F or G) to the goal state (H); it would not contain the first necessary action step leading from the current state to the next (i.e. from A to B, C, D or E).

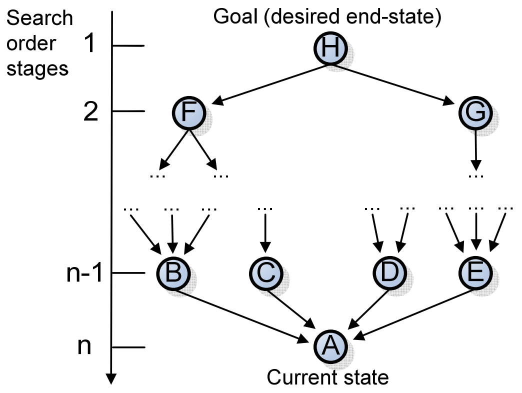


Figure 4 – The plan formulation start at the goal state H, working towards the current state A by finding appropriate actions.

3.1. Effect Based and GOAP

As effects from actions are modeled in GOAP, the planning taking place could somewhat be seen as EBP since the known effects may judge what actions get incorporated into the plan. Hierarchical Task Network
A Hierarchical Task Network (HTN) is a set of tasks ordered in a hierarchical fashion where the root task corresponds to the overall completion of *the* task, i.e.

[4] An *anytime algorithm* can be aborted and still deliver a valid result, even though it did not finish.

the utmost goal. The plan formulation can be thought of as a funnel where an abstract unexecutable high-level plan is first formulated (Wallace, 2004) in order to formulate more precise and ad hoc plans executable in the current situation. Just as in STRIPS-planning, the *operators* are the effects of a task. Each task is either *primitive* or *non-primitive* (i.e. *compound*) where a non-primitive task contains other tasks, which in turn could be either primitive or non-primitive (Erol et al., 1994). A non-primitive task can thus not be performed directly which is the case with a primitive task. In order to solve a non-primitive task, a *method* is applied which *reduces* the task, forming a new task network consisting of primitive and/or non-primitive tasks (Wallace, 2004). Muñoz-Avila & Hoang (2006, pp. 419) explains HTN planning as high level tasks being decomposed "…into simpler ones, until eventually all tasks have been decomposed into actions.".

According to Erol et al. (1994, pp. 1124) "The fundamental difference between STRIPS-style planning and HTN planning is the representation of 'desired change' in the world.". These "desired changes" are represented in the HTN as compound tasks which both incarnate a goal and evolves to contain the necessary actions to reach it. If a HTN task is seen and handled as an independent goal, then that goal could be fed to a GOAP planner which would work out a plan of more low level action details, whilst the HTN keeps track of the overall goal and subgoals. According to Wallace (2004) one of the most powerful features of HTN planning is its ability to perform *partial re-planning*, which in contrast to a regressive planner in GOAP does not have to formulate a whole new plan if the plan should fail. Muñoz-Avila & Hoang (2006) sees the reasoning process as the most crucial difference between STRIPS and HTN, where STRIPS reasons in level of actions whilst HTN reasons in level of tasks. **Error! Reference source not found.**Figure 5 depicts a HTN build up where the primitive tasks are incorporated in a subtask, which in turn builds up all the way to the top-level task.

3.2. Multi-agent and squad behaviour

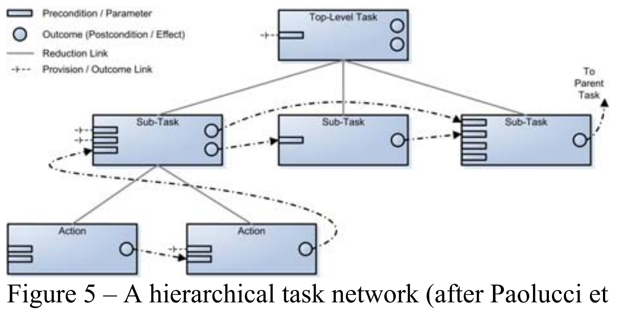


Figure 5 – A hierarchical task network (after Paolucci et al., 2000, pp. 4).

According to van der Sterren (2002a) the most appropriate action to perform is determined by each AI member taking account of the situation of his teammates and the threats known to the team, in addition to his own state. This is in the area of situation awareness which must be considered as a main precursor to decision-making, but it is nonetheless an independent process that must be recognized separate from the decision-making (Breton & Rousseau, 2005). The communication, interpretation and overall understanding of the intent between agents, determines whether a particular task succeeds; and there exists a need for a standard to communicate intent in a simulation of complex planning (Borgers et al., 2008). The complexity of a scenario incorporating plans for numerous agents can rapidly grow, not least from the resource handling stemming from the parallel tasks taking place. In a multi-agent environment where high-level strategic goals need to be dealt with, Muñoz-Avila & Hoang (2006, pp. 419) claims that even though it is possible to encode these strategies in STRIPS representation, a HTN is preferred since it "capture strategies naturally because of the explicit representation of stratified interrelations between tasks". The STRIPS-HTN relation could thus exist in symbiosis as the HTN planner reasons on what strategy to abide whilst the GOAP planner (i.e. a STRIPS realization) handles the more fast paced low-level action decisions.

In the SAMPLE (Situation Awareness Model for Pilot-in-the-Loop Evaluation) architecture, cognitive processing at squad commander level is made up of two streams consisting of tactical assessment and terrain assessment, which are responsible of rating the combat effectiveness of the squad (Aykroyd, Harper, Middleton & Hennon, 2002). By utilizing a *Battle Assessment Matrix* working with a list of known enemies and squad info, factors such as amount of team members and weapons used may determine the fuzzy rating of the squad's combat effectiveness (Aykroyd et al., 2002). The result is then together with the terrain data evaluated by expert systems and belief networks in order to determine the squad's activity for the current decision cycle. Individual agent combatants in SAMPLE receive various orders, where for example a combatant agent might be told to fire at a certain enemy or to restrict its movement to an area within the squad's line (Aykroyd et al., 2002). The combatants actual decisions stems from numerous analyzers where for example the *Shot Analyzer* can point out firing positions as well as weapon choices considering a certain enemy. Depending on what orders have been given, a combatant's selected threat target and best shot may be overridden as long as the target isn't deemed a critical threat, which for instance could be a nearby enemy with a clear shot and an assault rifle (Aykroyd et al., 2002).

According to Paolucci et al. (2000) it is of the utmost importance that planning systems in a multi-agent environment allow *execution while planning*, in order to allow “flexible interaction with other agents or with the domain while the agent plans locally” (Paolucci et al., 2000, pp. 18). In a multi-agent environment where GOAP is employed, various commands or doctrinal rules and conditions may be considered by inflicting a value or “cost” change of an agent’s goals and/or actions, making the goal or action more or less favorable. Doris & Silvia (2007) states that “A GOAP system imposes a modular architecture that facilitates sharing behaviors among agents and even across software projects.”. Depending on the cost degree and the agent’s susceptibility of the change, an agent might do the bidding of its surroundings. For example, a popular commander giving a dangerous move order to a military agent, might due to its expressives[5] considerably lower the cost of the `Move` action, making other actions like `StayInCover` or `Flee` less favorable. Hence, the military agent chooses and executes the `Move` action, even though the action per se is less favorable according to the agent. Another example is a doctrine stating that women and children shall be saved prior to men in a fire emergency. This doctrine could be implemented by giving the goals or actions `SaveWomanFromFire` and `SaveChildFromFire` a higher relevance value than `SaveManFromFire`. The commands, doctrines, personality preferences or other implicit conditions as described in 0, are thus applied as filters or layers on the primal functions. As more of these layers are applied a complex and implicit group behavior may emerge which takes many affectors into consideration. The GOAP architecture, much like the SAMPLE architecture, is thus capable of reacting to an adversary in a dynamic environment while employing a highly modular and extendable system.

3.3. Centralized vs. decentralized approach

A squad may be decentralized meaning that there exists no obvious squad leader; all squad members have the ability to issue suggestions based on their perceptions and person preferences, i.e. style. Van der Sterren (2002a, pp. 235) identifies the following attractive reasons of choosing a decentralized approach:

- It simply is an extension of the individual AI.
- It robustly handles many situations.
- It deals well with a variety of capabilities within the team.
- It can easily be combined with scripted squad member actions.

In order to optimize the squad’s situation awareness, it is most important that the squad members communicate and share their intentions and observations. By doing this the squad may somewhat become a single entity with each member acting more as a sensor or suggestion input than a free roaming entity. Each squad member may thus be seen as a tool or weapon with which the entity, i.e. the squad, can achieve its goal; much like a human hand attains an arbitrary goal by using its squad members – the fingers. However, the decentralized approach to squad AI is just a solid base from which to work, and it is easily extended with increasingly realistic tactics (van der Sterren, 2002a).
In a centralized squad the communication and orders are sent in a hierarchical manner, possible of having multiple echelons. The benefits of this approach are according to van der Sterren (2002b):

- The amount of problems each type of AI has to cope with is significantly reduced.
- Each kind of AI can be changed and specialized independently from each other.
- Lower CPU demand as a result of fewer computations.

The main problems when dealing with a centralized approach are those of conflicting objectives. The AI of a squad member is usually not able to see the mission’s grand scheme and may have different goals than those of the squad commander’s AI, e.g. the squad member might prefer to pick up a better weapon instead of directly dealing with a certain threat seen by the squad commander (van der Sterren, 2002b). To deal with these conflicts van der Sterren points out that the squad members must know the rules-of-engagement based on the current situation. For example a situation where the squad’s actions must be coordinated meticulously is more sensitive to ego based behavior. Van der Sterren (2002b, pp. 249) further claims that “while there is no such thing as the optimal command style, you will go a long way addressing your squad requirements by a smart mix of the two styles [decentralized and centralized approach]…”.

[5] *Expressives* as described in 0 (Gustavsson et al., 2008).

In a dynamic real world scenario it is often desirable to avoid detailed low level command and control (C2)[6], and as stated by Alberts (2007, pp. 1) "The future of command and control is not *Command and Control*. In fact, the term *Command and Control* has become a significant impediment to progress.". It is in most cases instead preferable to only express orders (goals) in high level terms, i.e. intent, which the lower level echelons (agents) then may solve how they may see best fit, and by doing this exchanging *command and control* in favor of *focus and convergence* (Alberts, 2007). This means that even though there exist a needed command hierarchy, the commander should not act in a strong centralized manner and thereby hide relevant information and pinpoint actions to the subordinates. Instead, he or she should allow the subordinates to *self-synchronize*[7] and be part of *Network Centric Warfare* (Smith, 2003; Alberts, 2007). This relatively modern operational doctrinal view offers the subordinates a broader view of the situation as they together may decide on what course of actions to take, much like using a blackboard and emphasize emergent behavior, in contrast to being micro managed by a high level commander that perhaps has inadequate situation awareness. Alberts (2007, pp. 1) mentions the three core concepts agility, focus and convergence; where "…agility is the critical capability that organizations need to meet the challenges of complexity and uncertainty; focus provides the context and defines the purposes of the endeavor; convergence is the goal-seeking process that guides actions and effects."

GOAP do employ both a decentralized and centralized approach in that a specific order is having the effect of a certain goal having a higher preference, thus allowing the squad member to consider the commander's will, whilst still, to some extent, maintaining its own prioritization of goals. A squad in GOAP is therefore not strongly coupled to its commander which also allows for easy substitution in case the commander meets his or her demise or is otherwise compromised.

4. Challange

The purpose of this project is to extend the GOAP architecture with threat analysis capabilities by slightly modifying the GOAP architecture. This would enable entities using GOAP to have increased situation awareness as they are capable of determining how big a threat an assessed entity poses. By utilizing this knowledge and make the assessing entity communicate its findings to other entities (both virtual and real), perhaps better strategies and decisions could emerge. The inherent generic benefits of the GOAP architecture should also comprise the threat analyzer as it would perform planning in a similar way. Hence the threat analyzer could be seen as a GOAP planner, but instead of delivering an action plan the threat analyzer is to determine how big a threat a certain entity constitutes, based on that entity's current world states and what actions are known to the threat analyzer.

Much like how an ordinary GOAP planner is loaded with a set of actions, the threat analyzer is to be loaded with those actions it should be able to account for when performing a threat analysis; enabling it to formulate a "risk plan" given a certain goal world state fed by an arbitrary input, like a doctrine or a command. The goal world state of the analyzer is thus more as an "anti goal" or *unwanted* world state, and the world states deemed dangerous will be substeps of a plan leading to the unwanted world state. By comparing an entity's world states by those found in the formulated risk plan, the analyzer should be able to determine the threat level and its completeness of the entity being assessed. In other words, the analyzer is to somewhat anticipate the occurrence imminence[8] of its anti goal world state.

The experiment will be seen as a success if the implemented threat analyzer is able to assess a certain entity and assign it a corresponding threat level and completeness, by using a simplified GOAP architecture[9]. The benefits of this threat analyzer in conjunction with GOAP should among other things be the simplicity of not having to flag a multitude of individual world states as "dangerous", but rather just

[6] C2 is defined by Department of Defense Dictionary of Military and Associated Terms (Joint Publication 1-02, 2001) as: "Command and Control: The exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission. Command and control functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission…".

[7] "Self-synchronization is a way of distributing decision rights and is associated with a specific set of patterns of interactions among participants." (Alberts, 2007, pp. 10)

[8] The closer an entity is to achieving an undesired world state (from the threat analyzer's point of view), the bigger threat it poses.

[9] The need of utilizing the complete GOAP architecture with all performance optimizations is most likely an overkill, since the scenario will hold only two entities with a small amount of actions.

flag one or a few world states as undesired; from which all relevant interconnected dangerous world states, i.e. threats, can be found when assessing a certain entity.

5. Realization

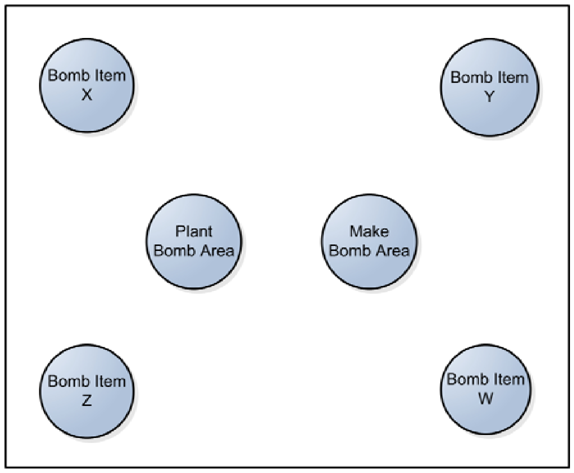


Figure 6 – The simulated world where six key areas reside.

The competitors will be equipped with a seeing sense, able to identify agents and items residing in a walled rectangular 2D world, as depicted in **Error! Reference source not found.**. The world size will be set to 800x600 pixels and the agents are to be given a movement speed of 100 pixels per second. This will enable the agents to go around their business fast enough while allowing an observer to follow the agents' behaviors. The agents seeing senses will be given a 90 degree field of view with a see range of 100 pixels. The field of view will thus somewhat map to a human counterpart, while the short see range keeps the world and scenario more manageable. To clearly see the agents and the bomb items, the entities are given a radius size of 10 or 15 pixels respectively.

In order to give the LECOF a chance to assess the Bandit while it is making or planting the bomb, the make and plant bomb actions will be given a duration of two seconds. Since the LECOF only assesses and never intervenes, the Bandit agent will at some point reach its goal. When this happens the scenario will be restarted and a new *round* will begin. The positions where the Bandit will be able to make and plant the bomb will be referred to as *The Shack* and *The Target* (see **Error! Reference source not found.**).

One of the competitors, the *Bandit*, will have the goal of reaching the world state `(kTargetIsBlownUp, true)`, which will be done by amassing four bomb items by walking up to them. Once the items have been collected, the Bandit will be able to make and plant a bomb at the make bomb and plant bomb area respectively. Upon planting the bomb, the value of the goal world state key `kTargetIsBlownUp` will change from `false` to `true`, meaning that the Bandit has reached its goal. The other competitor, referred to as the *Law Enforcing Coalition Force* (LECOF), will have a threat analyzer, loaded with the same goal world state as the Bandit, i.e. `(kTargetIsBlownUp, true)`. When the LECOF's seeing sense spots the Bandit, the LECOF's threat analyzer is to assess the Bandit's world states according to **Error! Reference source not found.**
.

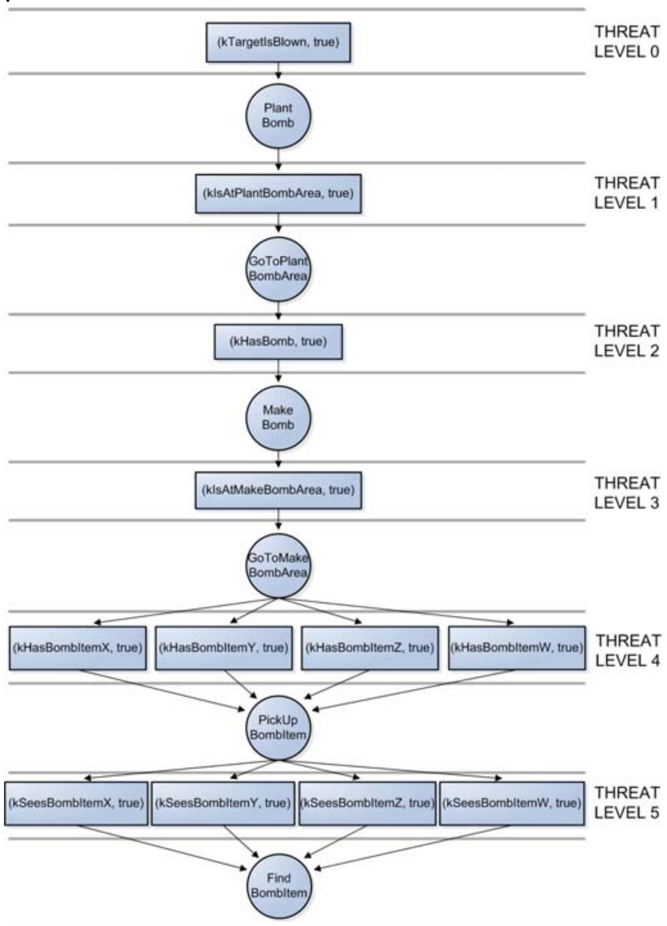


Figure 7 - The action graph used by the Bandit's GOAP planner and the LECOF's threat analyzer (rectangles represent world states, circles represent actions).

The action graphs of the Bandit and the threat analyzer will thus be identical, but used in different ways. The actions (circles), as depicted in **Error! Reference source not found.**, are loaded into the Bandit's GOAP planner and the LECOF's threat analyzer, which is to assign the Bandit a certain threat level and completion percentage depending on the world states assessed. If, for example, the LECOF spots a Bandit that is carrying

bomb item x and has all world state keys set to `false` (except `kHasBombItemX`), the threat analyzer should be able to find out that the Bandit is of threat level 4 with a 25 % completion, i.e. one of four of that level's world states correspond to the threat analyzer's action graph.
The following components have been identified and deemed necessary for the project:

- **Entities, i.e. agents and bomb items**. The embodiment of an agent or bomb item represented by a graphic primitive or picture. The simulation will contain two competing agents, and the bomb items will be used as means of changing the world state (and consequently the threat level) of the Bandit agent.
- **Senses**. Each agent will have a seeing sense responsible for detecting other agents and bomb items. The field of view will be conical and the see range will be limited in order to have the agents look for each other or bomb items.
- **Working memory**. Each agent will have its own working memory which will contain representations of sensed entities of the world, i.e. memory facts. For instance may the seeing sense post its primitive findings to the working memory, where they later on may be accessed, questioned or evaluated by other subsystems of the agent. This does in some sense let the working memory have the role of a blackboard, that for example could be asked what known entity is currently closest.
- **Simple GOAP planner**. The simplified GOAP planner will be responsible of formulating a plan given a certain goal world state, found by the goal planner. As performance optimizations will not be necessary given the size of this project, A* searches will be replaced by brute force searches[10].
- **Threat analyzer**. The threat analyzer will be responsible of calculating the threat level and its completeness of a certain entity by assessing the world states found in that entity, given a certain unwanted world state, i.e. "anti goal" of the threat analyzer.
- **Plan**. The plan represents a component that holds a set of necessary actions which will enable its user to reach its current goal world state. The plan will be passed from the GOAP planner to its owner, and is to be replaced with a new plan whenever the plan fails or completes.
- **Goal planner**. The goal planner will be responsible of finding the most promising goal, i.e. the goal with highest relevancy. Once the most promising goal has been found, its goal world state is returned and used by the GOAP planner in order to find a suitable action plan.[11]
- **Goals**. Goals will be loaded into a goal planner and consider the current world states and/or other contextual data in order to calculate their relevancy.
- **Actions**. Actions are representations of world state transitions which are linked by preconditions and effects. Actions will be used by a GOAP planner or threat analyzer in order to formulate a plan or to find dangerous (unwanted) world states.

5.1. Identified actions and goals

To keep the implementation simple and focus on threat analysis with GOAP, only two goals will be implemented, one for each agent. This should ease the debugging complexity as well as keeping the scope of the scenario to a manageable and intuitive size, regarding the needed actions. One could of course argue about the need of a goal planner when only one goal exists, but the architecture will this way be extendible if a need of more goals arises. So even though there will exist only one goal per agent, the functionality of the goals will be uncompromised, and each goal will have a `Relevance` function responsible of returning a representative value of the particular goal to the goal planner.
In order to build a scenario of the suggested design the following goals have been identified and need to be implemented:

- **Plant bomb**. The goal `PlantBomb` will be pursued by the Bandit agent and has the goal world state `(kTargetIsBlownUp, true)`. This goal will also be loaded into the LECOF's threat analyzer.
- **Find threat**. The goal `FindThreat` will be pursued by the LECOF agent and has the goal world state `(kSeesThreat, true)`.

[10] A *brute force search* could also be described as an *exhaustive search* where in this case *all* actions are examined in order to formulate a plan.

[11] The goal planner could be seen as a commander telling or influencing the agent (i.e. the agent's GOAP planner) what the desired effects or end-state is. The commander is thus allowing the (subordinate) agent to formulate a plan of its own, i.e. to find a set of actions, not specified by the commander. Also, note that the goal planner could be an incarnation of a HTN planner that handles the overall strategy and feeds world state goals to the GOAP planner.

To enable the agents to reach these goals and make an interesting scenario, the following actions will also have to be implemented:

- **Find Bandit**. The action `FindBandit` will enable a LECOF agent to find a Bandit by roaming the world in a random direction until it hits a wall or sees a Bandit. If the LECOF hits a wall a new direction is randomly chosen. This means that no neat steering behavior is used since it is not required for the problem at hand.
  Precondition: None
  Effect: `(kSeesBandit, true)`
- **Assess Bandit**. By using the action `AssessBandit` a LECOF may assess a seen Bandit (the Bandit's world states). Depending on the Bandit's world states the LECOF's threat analyzer comes up with the current threat level and completion of the Bandit assessed.
  Precondition: `(kSeesBandit, true)`
  Effect: `(kSeesThreat, true)`
- **Find bomb item**. The action `FindBombItem` will work much like the `FindBandit` action and will enable the Bandit agent to find a bomb item. A Bandit using this action will roam the world in a random direction until it hits a wall or sees a bomb item. If the Bandit hits a wall a new direction is randomly chosen. Just like the LECOF agent, the Bandit does not use any advanced steering behavior.
  Precondition: None
  Effects: `(kSeesBombItemX, true) OR (kSeesBombItemY, true) OR (kSeesBombItemZ, true) OR (kSeesBombItemW, true)`
- **Pick up bomb item**. The action `PickUpBombItem` will enable the Bandit to pick up a bomb item, i.e. changing the value of the appropriate world state key. If for example the agent lacks bomb item x but sees it, this action makes the agent pick up the bomb item, and thereby changing the Bandit's world state key `kHasBombItemX` from `false` to `true`.
  Preconditions: `(kSeesBombItemX, true) OR (kSeesBombItemY, true) OR (kSeesBombItemZ, true) OR (kSeesBombItemW, true)`
  Effects: `(kHasBombItemX, true) OR (kHasBombItemY, true) OR (kHasBombItemZ, true) OR (kHasBombItemW, true)`
- **Go to make bomb area**. The action `GoToMakeBombArea` will enable the Bandit to go to the make bomb area (see **Error! Reference source not found.**) once all bomb items have been collected.
  Preconditions : `(kHasBombItemX, true) AND (kHasBombItemY, true) AND (kHasBombItemZ, true) AND (kHasBombItemW, true)`
  Effect: `(kIsAtMakeBombArea, true)`
- **Make bomb**. By using the action `MakeBomb` at the make bomb area, the Bandit will turn the bomb items into a bomb.
  Precondition: `(kIsAtMakeBombArea, true)`
  Effect: `(kHasBomb, true)`
- **Go to plant bomb area**. Much like the action `GoToMakeBombArea`, this action will enable the Bandit to go to the plant bomb area (see **Error! Reference source not found.**) once the Bandit possess a bomb.
  Precondition: `(kHasBomb, true)`
  Effect: `(kIsAtPlantBombArea, true)`
- **Plant bomb**. By using the action `PlantBomb` at the plant bomb area, the Bandit plants the bomb.
  Precondition: `(kIsAtPlantBombArea, true)`
  Effect: `(kTargetIsBlownUp, true)`

5.2. Identified world state keys

The value type of all world state keys will be Boolean since more advanced value types are deemed unnecessary for this project. For instance will the world state key `kSeesPaper` have the value `true` or `false`, instead of a handle to the paper being seen.
To string together the bomb items with more accessible and intuitive real world counterparts, the naming of the bomb items will hereafter be Paper, Fertilizer, Sulphur and Bombshell. As the author is no bomb expert, it is at the reader's discretion to look upon the bomb items as he or she may see best fit. The following world state keys have been identified and are to be used in the project:

```
kSeesPaper
kSeesFertilizer
kSeesSulphur
kSeesBombshell
kHasPaper
kHasFertilizer
kHasSulphur
```

```
kHasBombshell
kIsAtMakeBombArea
kHasBomb
kIsAtPlantBombArea
kTargetIsBlownUp
kIsAtTargetNode
kSeesBandit
kSeesThreat
```

6. Results

As in most simulations the result is highly coupled with what starting values and abilities the performing entities are given. In this project parameters such as world size, item placement, agent speed, field of view, see range, entity size and other ability limitations such as always looking in the direction of movement, have a critical impact on the overall result. By tweaking these parameters to a somewhat realistic or accessible scenario, the result can be evaluated in the context of the problem at hand. The point here is that the values that were used do share a resemblance of a real world counterpart, even though they have been tweaked to serve the outcome in a positive way. For example were the agents' field of view set to 90 degrees and their movement speed to 100 pixels per second. If the LECOF's field of view was to be impaired to zero, i.e. turning the LECOF blind, the Bandit would never be assessed and no threat assessments would occur, even though the underlying threat analysis technique remained the same. As it turned out, the scenario was simplified enough to be accessible, whilst remaining true to a somewhat similar real world setting.

7. Discussion

7.1. Message passing

Reynolds (2002) states that "The use of message passing is very effective when implementing team-based AI. Using messages to communicate between different layers of the hierarchy is a very natural way of passing orders and information.". By employing message passing the system benefits from increased information hiding and decoupling of interacting components, since the entity sending the message does not have to know anything about the recipient, other than its ID. By sending messages the interacting entities are also able to share their awareness of the situation, which for instance built the squad's shared tactical picture in the SAMPLE agent architecture (Aykroyd, Harper, Middleton & Hennon, 2002), further described in section 3.2. Van der Sterren (2002a) gives these reasons to why it is more attractive to use messages to pass squad member state information around, than inspecting the AI objects directly:

- You can model communication latency by briefly queuing the messages.
- You can present the message in the game using animations or sound.
- You can filter out messages to the player to prevent overloading him.
- The squad member will send messages to dead squad members it assumes still to be alive, adding to realism (and preventing "perfect knowledge").
- You can use scripted entities to direct one or more squad members by having the scripted entities send messages to these squad members.
- You can accommodate human squad members, whose state information is largely unavailable, but for whom the AI can emulate messages with observations and intentions.

Besides incorporating an agent-to-agent communication system, the message passing system may also serve as a tool for communication between other components of the system. By using messages the system becomes an event-driven architecture, which generally is preferred because of its efficiency and polling avoidance (Buckland, 2005). The event-driven architecture allows an entity to be sent the information of interest and thereafter act upon it as it may seem fit. Yet another benefit is explained by Reynolds (2002, pp. 267):

Fortunately, this implementation [the use of message passing] offers the programmer the opportunity to log the messages between the levels of command. This can be used to provide quite natural dialogue between levels of command outlining all the information and decisions that have been made. This can be invaluable when tracing where the errant information has come from or which decision was at fault.

7.2. Intent and the advent of C-BML

Another aspect of message passing is the use of a consistent language which in the best of worlds would be unambiguous and interpretable by both man and machine. This would enable the message receivers to without subjective influences grasp the sender's ambition of the current subject. The benefits of a common and unambiguous language, fit for human-to-human, human-to-machine, or machine-to-machine interaction, seem almost endless. Alberts (2007) claims that "Changing the language we use will free us in our search for a solution for a better approach. If we develop a suitable language, it will point us in a right direction as well.". Work in this area is currently underway with the evolution of the Coalition Battle Management Language (C-BML) and Tactical Situation Object (TSO), as described by Gustavsson, Nero, Wemmergård, Turnitsa, Korhonen, Evertsson and Garcia (2008). The C-BML standard is based on the 5W, which according to Kleiner et al. (1998, pp. 883) makes it:

…possible to describe any mission or task given to a subordinate in a standardized manner with a who (relates to a task organization database entry), what (in doctrinal terms), when (specified time, on order, or keyed to a triggered event), where (related to a coordinate or graphical control measure), and why (in doctrinal terms).

C-BML is supposed to express the knowledge which flows within a certain system like an autonomous agent or automated decision-support system, and unambiguously describe Commander's Intent (CI) without in any way impairing or constraining the CI (Lagervik & Gustavsson, 2006; Blais, Galvin & Hieb, 2005; Gustavsson, Hieb, Eriksson, Moore & Niklasson, 2008). As C-BML stems from BML (Battle Management Language), it shares the same purpose in being able to formulate orders, requests and reports through a standardized language for military communication. In turn, BML is built on the Command and Control Lexical Grammar (C2LG) which was first formalized by Hieb & Schade (2006) and is still under development. Borgers, Spaans, Voogd, Hieb & Bonse (2008) claims that "As long as every agent, operator and C2SS [Command and Control Support System] sends and receives BML messages, a seamless connection between them is possible.". The representation of CI is to contain the mission team's purpose, the anticipated End-State of the mission and desired key tasks (Gustavsson et al., 2008). Furthermore, the intent can be divided into an *explicit* and *implicit* part, where the explicit intent can be seen as the commander's straightforward orders, while the implicit intent is developed over a longer time, prior to the mission, and consist of the *expressives*[12] and the concepts, policies, laws and doctrine agreed to by military, civil, organizations, agencies, nations and coalitions (Gustavsson et al., 2008). Farrell (2004) gives the example that if explicit intent is "*to capture the hill*", then implicit intent might be "*to capture the hill with minimal battle damage*" or "*to capture the hill with Air Force assets only*".

The Operations Intent and Effects Model (OIEM) in figure 9 presents that simulation systems can use for both forward and backtracking planning. The model shows how an initial state is detected by a Decision Making Process that produces an order that describe actions that cause effects that change the state into the desired End-State. The OIEM is to be a basis for a system design that combines the grammar and engineered knowledge with search mechanisms (Gustavsson et al., 2008).

The huge effort of developing a language capable of preserving the Commander's Intent to all participants of the command hierarchy is still in its prime, but will hopefully satisfy the ever increasing need to control, command, and coordinate the complex planning situations a coalition[13] may face.

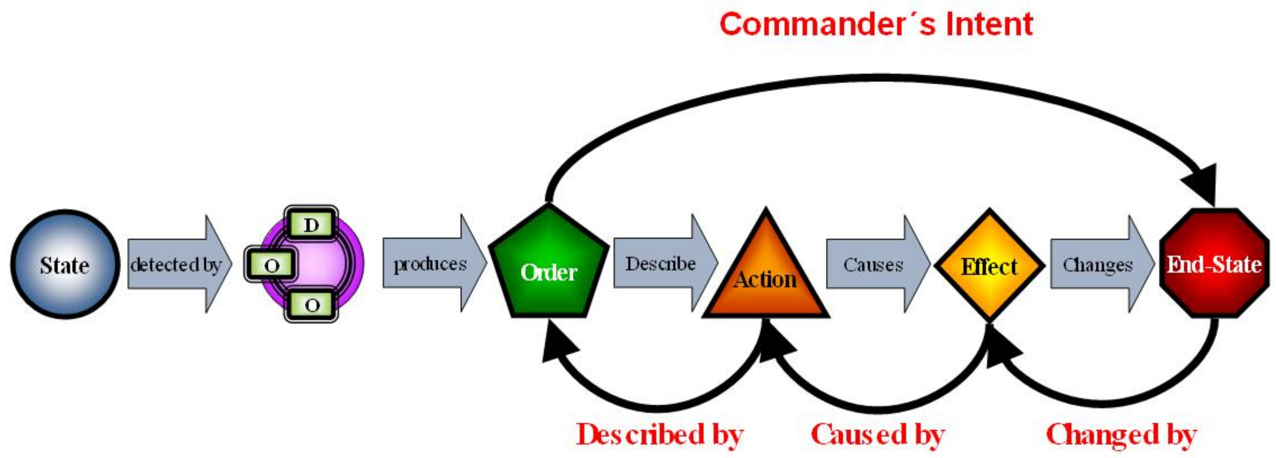


Figure 9 – Operations Intent and Effects Model

3. References

Ackoff, R. L. 1999. *Ackoff's Best – His Classic Writings on Management*. John Wiley & Sons, Hoboken, New Jersey.

Alberts, D. S. 2007. Agility, Focus, and Convergence: The Future of Command and Control. *The International C2 Journal*, 1(1), pp. 1-30. Available from: http://www.dodccrp.org/files/IC2J_v1n1_01_Alberts.pdf [Accessed 28 April 2008]

Aykroyd, P., Harper, K. A., Middleton, V. and Hennon, C. G. 2002. Cognitive Modeling of Individual Combatant and Small Unit Decision-Making within the Integrated Unit Simulation System. In Proceedings of the 11th Computer-Generated Forces and Behavior Represe, May 7-9, Orlando, FL, USA, ref 11TH-CGF-073.

Baxter, J. and Hepplewhite, R. 2000. A Hierarchical Distributed Planning Framework for Simulated Battlefield Entities. *In Proccedings of the 19th Workshop of the UK Planning and Scheduling Special Interest Group*, Milton Keynes, UK, December 13-14. Available from: http://mcs.open.ac.uk/plansig2000/Papers/P7.pdf [Accessed 4 April 2008]

Bedney, G. and Meister, D. 1999. Theory of Activity and Situation Awareness. *Int. J. Cognitive Ergonomics*, 3(1), pp. 63-72.

Blais, C., Galvin, K. and Hieb, M. 2005. Coalition Battle Management Language (C-BML) Study Group Report. *In Proceedings of the 2005 Fall Simulation Interoperability Workshop*, Orlando, Florida, USA, September, ref 05F-SIW-041 Available from: http://www.movesinstitute.org/~blais/Documents/05F-SIW-041.pdf [Accessed 4 February 2008]

Borgers, E., Spaans, M., Voogd, J., Hieb, M. R. and Bonse, R. 2008. Using a Command and Control Language to Simulate Operations in a Multi-Agent Environment. *In Proceedings of the 13th ICCRTS: C2 for Complex Endeavors*, Bellevue, WA, USA, June 17-19. [CD-ROM]

Breton, R. and Rousseau, R. 2005. The C-OODA: A Cognitive Version of the OODA Loop To Represent C^2 Activities [online]. *In Proceedings of the 10th International Command and Control Research and Technology Symposium*, McLean, USA, June 13-16. [CD-ROM] Available from: http://www.dodccrp.org/events/10th_ICCRTS/CD/presentations/280.pdf [Accessed 4 February 2008]

Buckland, M. 2002. *AI Techniques for Game Programming*. Premier Publishing Ltd, Glasgow, UK.
Buckland, M. 2005. *Programming Game AI by Example*. Wordware Publishing Inc, Plano, Texas, USA.
Buschmann, F. 1996. *Pattern-Oriented Software Architecture*. Wiley & Sons Ltd, West Sussex, UK.

Burke, R., Isla, D., Downie, M., Ivanov, Y. and Blumberg, B. 2001. Creature Smarts: The Art and Architecture of a Virtual Brain. *In Proceedings of Game Developers Conference*, San Jose, California, USA, March 20-24, pp. 147-166. Available from: http://www.media.mit.edu/characters/additional%20resources/gdc01.pdf [Accessed 21 February 2008]

Doris, K. and Silvia, D. 2007. Improved Missile Route Planning and Targeting using Game-Based Computational Intelligence. *In Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications*, Honolulu, Hawaii, April 1-5, pp. 63-68.
Dybsand, E. 2004. AI Game Programming Wisdom 2. In: Steve Rabin, editor.

Goal-Directed Behavior Using Composite Tasks. Charles River Media, Massachusetts, chapter 3.6, pp. 237-245.

Endsley M. R. 1995. Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors*, 37(1), pp. 32-64.

Erol, K., Hendler, J. and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Seattle, Washington, USA, July 31-August 4, pp. 1123-1128.

Farrell, P. S. E. 2004. Measuring Common Intent during Effects Based Planning [online]. *In Proceedings of 2004 Command and Control Research and Technology Symposium: The Power of Information Age Concepts and Technologies*, Defence Research and Development Canada – Toronto, San Diego, CA, USA, June 15-17. [CD-ROM] Available from: http://www.dodccrp.org/files/2004_CCRTS.zip [Accessed 29 April 2008]

F.E.A.R. – First Encounter Assault Recon 2005. Monolith Productions/Sierra Entertainment Inc, Kirkland, Washington, USA. [Computer program]

Fikes, R. E. and Nilsson N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4), pp. 189-208.

Gustavsson, P. M., Nero, E., Wemmergård, J., Turnitsa, C., Korhonen, M., Evertsson, D., and Garcia, J. 2008. Tactical Situation Object – Enabling joint Crisis Management Training. *In Proceedings of the 2008 Spring Simulation Interoperability Workshop*, IEEE CS Press, Providence, Rhode Island, USA, April 14-17, ref 08S-SIW-018.

Gustavsson, P. M., Hieb, M., Eriksson, P., Moore, P. and Niklasson, L. 2008. Machine Interpretable Representation of Commander's Intent. *In Proceedings of the 13th ICCRTS: C2 for Complex Endeavors*, Bellevue, WA, USA, June 17-19, ref I-188. [CD-ROM]

Hieb, M. R. and Schade, U. 2008. A Linguistic Basis For Multi-Agency Coordination. *In Proceedings of the 13th ICCRTS: C2 for Complex Endeavors*, Bellevue, WA, USA, June 17-19, ref I-152. [CD-ROM]

Hieb, M. R. and Schade, U. 2006. Formalizing Battle Management Language: A Grammar for Specifying Orders. Presented at *Spring Simulation Interoperability Workshop.* Simulation Interoperability Standards Organization, Huntsville, AL, USA, April 2-7, ref 06S-SIW-068.

Hoang, H., Lee-Urban, S. and Muñoz-Avila, H. 2005. Hierarchical Plan Representations for Encoding Strategic Game AI. *In Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*, AAAI Press, June 1-5, Marina del Rey, CA, USA, pp. 63-68.

Holland, T. O. 2008. Towards a Lexicon for the Modeling and Simulation of Emergent Behavior Systems. *In Proceedings of 2008 Spring SIW/BRIMS Conference*, April 14-18, Providence, RI, USA, ref 08S-SIW-058.

International Game Developers Association, Working Group on Goal-Oriented Action Planning 2003. *The 2003 AIISC Report*. Available from: http://www.igda.org/ai/report-2003/aiisc_goap_report_2003.html [Accessed 14 February 2008]

International Game Developers Association, Working Group on Goal-Oriented Action Planning 2004. *The 2004 AIISC Report*. Available from: http://www.igda.org/ai/report-2004/goap.html [Accessed 14 February 2008]

International Game Developers Association, Working Group on Goal-Oriented Action Planning 2005. *The 2005 AIISC Report*. Available from: http://www.igda.org/ai/report-2005/goap.html [Accessed 11 February 2008]

Kleiner, M. S., Carey, S. A. and Beach, J. 1998. Communication Mission-Type Orders To Virtual Commanders. In: D. J. Medeiros, E. F. Watson, J. S. Carson, M. S. Manivannan, editors. *In Proceedings of the 1998 Winter Simulation Conference*, Washington DC, USA, December 13-16, pp. 881-886.

Lagervik, C. and Gustavsson, P. M. 2006. A System Theoretical Approach to Situation Awareness and Architecture. *In Proceedings of the 11th International Command and Control Research and Technical Symposium (ICCRTS)*, Cambridge, UK, September 26-28. [CD-ROM]

Long, E. 2007. Enhanced NPC Behaviour using Goal Oriented Action Planning. M.Sc. thesis. School of Computing and Advanced Technologies, Division of Software Engineering, University of Abertay Dundee, Scotland, UK. Available from: www.edmundlong.com/Projects/Masters_EnhancedBehaviourGOAP_EddieLong.pdf [Accessed 19 February 2008]

Muñoz-Avila, H. and Hoang, H. 2006. AI Game Programming Wisdom 3. In: Steve Rabin, editor. *Coordinating Teams of Bots with Hierarchical Task Network Planning*. Charles River Media, Massachusetts, chapter 5.4, pp. 417-427.

No One Lives Forever 2 : A Spy in H.A.R.M.'s *Way* 2002. Monolith Productions/Sierra Entertainment Inc, Kirkland, Washington, USA. [Computer program]

O'Brien, J 2002. AI Game Programming Wisdom. In: Steve Rabin, editor. *A Flexible Goal-Based Planning Architecture*. Charles River Media, Massachusetts, chapter 7.5, pp. 375-383.

Ontañón, S., Mishra, K., Sugandh, N. and Ram, A. 2007. Case-Based Planning and Execution for Real-Time Strategy Games. *In Proceedings of the International Conference on Case-based Reasoning (ICCBR 2007)*, Belfast, Northern Ireland, UK, August 13-16, pp. 164-178. Available from: http://www.cc.gatech.edu/faculty/ashwin/papers/er-07-

11.pdf
[Accessed 11 February 2008]

Orkin, J. 2004a. Symbolic Representation of Game World State: Toward Real-Time Planning in Games. *In Proceedings of AAAI Workshop - Challenges in Game Artificial Intelligence 19th National Conference on Artificial Intelligence*, AAAI Press, San Jose, CA, USA, July 25-26, pp. 26-30.

Orkin, J. 2004b. AI Game Programming Wisdom 2. In: Steve Rabin, editor. *Simple Techniques for Coordinated Behavior*. Charles River Media, Massachusetts, chapter 3.2, pp. 199-206.

Orkin, J. 2004c. AI Game Programming Wisdom 2. In: Steve Rabin, editor. *Applying Goal-Oriented Action Planning to Games*. Charles River Media, Massachusetts, chapter 3.4, pp. 217-227.

Orkin, J. 2005. Agent Architecture Considerations for Real-Time Planning in Games. *In Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*, Marina del Rey, California, USA, June 1-3, pp. 105-110. Available from: http://web.media.mit.edu/~jorkin/aiide05OrkinJ.pdf
[Accessed 21 February 2008]

Orkin, J. 2006. Three States and a Plan: The A.I. of F.E.A.R. *In Proceedings of the Game Developer's Conference 2006*, San Jose, CA, USA, March 20-24. Available from: www.cs.ualberta.ca/~bulitko/F06/papers/gdc2006_orkin_jeff_fear.pdf
[Accessed 31 January 2008]

Paolucci, M., Shehory, O. and Sycara, K. 2000. Interleaving Planning and Execution in a Multiagent Team Planning Environment. *Technical report CMU-RI-TR-00-01, Robotics Institute*, Carnegie Mellon University, USA, February 5. Available from: http://www.ri.cmu.edu/pubs/pub_3274.html [Accessed 21 April 2008]

Reynolds, J. 2002. AI Game Programming Wisdom. In: Steve Rabin, editor. *Tactical Team AI Using a Command Hierarchy*. Charles River Media, Massachusetts, chapter 5.5, pp. 260-271.

Schubert, J., Wallén, M. and Walter, J. 2007. Morphological Refinement of Effect Based Planning. *Proceedings of the Third Int. Conf. Military Technology (MilTech3)*, June, Stockholm, Sweden, FOI – Totalförsvarets forskningsinstitut, ref Paper Or21.

Simple and Fast Multimedia Library 2008. Laurent Gomila. [Computer program]
Available from: http://www.sfml-dev.org/
[Accessed 21 March 2008]

Smith, E. A. 2003. *Effect-Based Operations* [online]. The Command and Control Research Program. Available from:
http://www.dodccrp.org/files/Smith_EBO.PDF
[Accessed 28 April 2008]

Straatman, R., van der Sterren, W. and Beij, A. 2005. Killzone's AI: Dynamic Procedural Combat Tactics. *In Proceedings of the Game Developers Conference 2005*, San Francisco, CA, USA, March 7-11. Available from: http://www.cgf-ai.com/docs/straatman_remco_killzone_ai.pdf
[Accessed 5 February 2008]

S.T.A.L.K.E.R.: Shadow of Chernobyl 2007. GSC Game World/THQ, Kiev, Ukraine. [Computer program]

van der Sterren, W. 2002a. AI Game Programming Wisdom. In: Steve Rabin, editor. *Squad Tactics: Team AI and Emergent Maneuvers*. Charles River Media, Massachusetts, chapter 5.3, pp. 233-246.

van der Sterren, W. 2002b. AI Game Programming Wisdom. In: Steve Rabin, editor. *Squad Tactics: Planned Maneuvers*. Charles River Media, Massachusetts, chapter 5.4, pp. 247-259.

Wallace, N. 2004. AI Game Programming Wisdom 2. In: Steve Rabin, editor. *Hierarchical Planning in Dynamic Worlds*, Charles River Media, Massachussetts, chapter 3.5, pp. 229-236.

Author Biographies

Philip Bjarnolf is a

Per M. Gustavsson is a Senior Research Scientist at the Training Systems and Information Fusion office at Saab Microwave Systems working with applied research in the area of advanced decision support systems. He is also an industrial Ph.D. student at University of Skövde, Sweden and Assistant Professor at C4I-Center, George Mason University. Gustavsson holds a M.Sc. in Computer Science, New Generations Representations, Distributed Real-Time Systems and a B.Sc. in Systems Programming both from University of Skövde, Sweden.

Christoffer Brax

Mikael Fredin