Tactics Development for Radar Stations

A rule-based approach for a simulated air defence

LUDVIG OLSSON

Master of Science Thesis
Stockholm, Sweden 2006
Tactics Development for Radar Stations

A rule-based approach for a simulated air defence

LUDVIG OLSSON

Master’s Thesis in Computer Science (20 credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2006
Supervisor at CSC was Olle Bälter
Examiner was Lars Kjelldahl

TRITA-CSC-E 2006:010
ISRN-KTH/CSC/E--06/010--SE
ISSN-1653-5715
Abstract

Radar warning systems are found in all modern fighter aircraft and their main purpose is to warn pilots of enemy military units such as radar stations, aircraft and missiles. Testing these systems for fighters in real life is an expensive, complex and somewhat dangerous task. Hence engineers aim to test these systems in an artificial environment by the use of software in order to simulate hour-long, realistic scenarios.

This thesis investigates the topic of bringing human-like behaviour and tactical reasoning to radar stations in the above mentioned software. Focus is on the use of a rule-based system and the result of this evaluation shows that these declarative systems have several advantages compared to conventional, procedural programming. This is especially true if tactics and behaviour of units are to be changed frequently and easily.

This thesis was commissioned by SaabTech, part of the Saab Group.

Keywords: rule-based programming, tactics, military, simulator, radar, radar warning system, fighter, aircraft, Jess.

Taktikutveckling för radarstationer
- regelbaserad programmering för ett simulerat luftförsvar

Sammanfattning

Radarvarningssystem återfinns i alla moderna stridsflygplan och har som främsta uppgift att varna piloten för militära fientliga enheter såsom radarstationer, flygplan och missiler. Att testa dessa system på stridsflygplan under verkliga scenarier är dyrt, komplext och i viss mening farligt. Därför försöker man testa systemen i en artificiell miljö med mjukvara som simulerar timslånga, realistiska scenarier.

Detta examensarbete utvärderar sätt att införa människolikt beteende och taktiskt resonemang bland radarstationer i ovan nämnda mjukvara. Fokus är på användandet av ett regelbaserat system och resultatet av denna utvärdering är att dessa deklarativa system har flera fördelar framför konventionell, procedurell programmering. Detta gäller speciellt då det är önskvärt att ofta och enkelt kunna ändra enheters taktiska beteende.

Examensarbetet utfördes för SaabTech inom Saab gruppen.

Nyckelord: regelbaserad programmering, taktik, militär, simulator, radar, radarvarningssystem, stridsflygplan, Jess.
Chapter 1

Introduction

This chapter gives a brief background of the problem domain and a presentation of the employer. It also presents the purpose of the thesis and outlines the structure of the report.

1.1 Background

*Radar warning systems* (RWS) are found in all modern fighter aircraft and their main purpose is to warn pilots of enemy military units such as radar stations, aircraft and missiles. Testing RWS for fighters in real life is an expensive, complex and somewhat dangerous task. Hence, engineers aim to test these systems in an artificial environment, known as a rig, which renders the “radar environment” that the aircraft and its warning system would be exposed to during a real flight or operation. Such a rig typically contains a software simulator that manages units, flight paths, maps etc. in order to perform hour-long, realistic scenarios. The rig also contains hardware to generate radar signals and the RWS itself.

One way of controlling the simulated units, such as radar stations, is to program them what to do and when to do it. It is then easy to compare the scenario to the alerts of the warning system. However, scripting the behaviour of units that surround the aircraft, such as radar stations, in a realistic manner can be difficult and time consuming. To avoid these drawbacks, units can be made to act to their surroundings in an autonomous way on a lower level, as well as to co-operate with each other on a higher, tactical level. One way of implementing this kind of human-like behaviour and tactical reasoning is by the use of a *rule-based system*. Rule-based systems are used in many applications but its suitability in the above mentioned environment is in need of more research.
1.2 SaabTech

SaabTech, part of the Saab Group, is a supplier of avionics and electronic warfare (EW) systems to the international market and a principal supplier of the Gripen fighter. This thesis is written for SaabTech’s Electronic Warfare Systems Division which is in charge of integrating and testing RWSs.

1.3 Purpose

The main purpose of this Master’s project is to investigate the issue of introducing high level tactics in software used for testing RWSs. More to the point, these three topics are addressed:

- Situational awareness, i.e. a detecting scheme for radar systems
- Autonomous behaviour of radar stations that mimic the operations flow of radar operators and apparatus
- Implementing tactics for sets of units by the use of a rule-based system

Combined, these improvements can to some extent simulate an anti-aircraft defence consisting of different types of radar stations behaving and co-operating as they would in real life.

It is not the purpose of this thesis to attempt to derive or test new tactics or target selection algorithms, but rather to give subject matter experts the ability to specify different tactics for different scenarios. Neither is the idea to create a learning system that attempts to develop better tactics on its own.

For testing the models and ideas developed, an integrated scenario editor and simulator developed by SaabTech will be used.

1.4 Thesis Outline

This thesis is organized as follows: Chapter 2 describes the problem domain which is the field of electronic warfare. The topic of radar is especially considered. Chapter 3 describes the system currently in use for testing and evaluating SaabTech’s RWS. Chapter 4 has its focus on the simulation of individual radar stations. It also introduces a way of connecting them to a central node so that tactics can be carried out from one single place. The work of Chapter 4 is needed before an evaluation can be made of the suitability of a rule-based system as a mean for specifying and handling tactics on a high level – this is the topic of Chapter 5. Chapter 6 presents implementation details. Chapter 7 provides conclusions and recommendations for future work.
Chapter 2

Electronic Warfare

This chapter introduces the field of electronic warfare and its associated concepts and terms in order to support later discussions. It also includes a brief historical presentation.

“EW is the art and science of denying the enemy forces the use of the electromagnetic spectrum while preserving its use for friendly forces.” [1]

2.1 History

Electronic warfare dates back to World War I [12] and has since been a vital part in wars like World War II, the Korean War, the Vietnam War, the Yom Kippur War and the Persian Gulf War.

An example that illustrates the importance of EW is the lead-up to the Invasion of Normandy. In 1944 British forces managed to simulate a whole navy fleet, moving towards Pas-de-Calais and an area north of Le Havre, by the use of newly developed chaffs (special aluminum-foil strips) and radar reflectors. The Germans’ radar screens were filled with false targets, distracting them from the real invasion. They were so convinced of the fleet’s existence that they even started pouring 30 cm artillery shells into the ocean [12].
2.2 Divisions of EW

Electronic warfare is commonly divided into three divisions, as shown in Figure 2.1.

![Figure 2.1. Divisions of EW.](image)

Electronic support involves receiving enemy signals to identify and locate threat emitters and to help determine the enemy’s force structure and deployment. Radar warning systems clearly fall under this category. Electronic attack involves measures taken to defeat enemy electronic assets. It includes jamming¹, chaff, flares, directed energy weapons, and anti-radiation missiles [1, 12, 14]. Electronic protection comprises countermeasures to enemy electronic attack.

2.3 Radar

Radar is an acronym for radio detection and ranging². Electromagnetic waves, typically within the microwave band (1 - 30 GHz) are transmitted, and a receiver listens for any echoes [12]. By analysing the reflected signal, estimations can be made about the location and speed of the reflector (i.e., the aircraft). As radio waves can be generated at high strengths and detected at even tiny powers, radar is suitable for detecting objects at very large ranges.

2.3.1 Radar basics

There are two basic types of radar: pulse radar and continuous wave (CW) radar. The CW radar can determine angle and speed of the reflector. The pulse radar measures angle and distance. Since the signal travels at the speed of light, the distance to the reflected target can easily be calculated (as the speed of light times the round-trip time divided by two).

Radars can be designed to detect targets at ranges of thousands of kilometres [14]. As a rule though, they operate at much shorter ranges due to obstruction in the line of sight (a radar cannot “see” through mountains). The so-called radar range equation (see Appendix A.2) gives an estimate of the range of a particular radar and will be used in section 4.3.1 for the development of radar “sensing”.

¹ Deliberate interference intended to prevent reception of signals in a specific frequency band.
² Even though not quite correct, “radar” will also be used to refer to radar stations (real and simulated) and systems as well as the electromagnetic waves emitted by these systems.
2.3.2 Radar modulations

The most common type of radar modulation is a stream of pulses (see Figure 2.2). A pulse is a short transmission with a duration, or pulse width (PW), in the order of magnitude of a microsecond. The interval between pulses is called the pulse repetition interval (PRI) and is typically a fraction of a millisecond to several milliseconds. Since transmission and reception are separated in time, only one antenna is needed.

![Figure 2.2. Pulse train.](image)

The CW radar, on the other hand, transmits continuously and therefore needs two antennas. These antennas need to be isolated to prevent the transmitter from saturating the receiver. The frequencies of the transmitted and received signals are compared to determine the relative velocity of the target from the Doppler shift. Modulating the frequency of a CW over time adds the capability to determine the distance to a target.

Another type of radar that combines some of the features of both pulse and CW systems is the pulse Doppler radar. Its pulses are coherent, which means that they are a continuation of the same signal and have phase consistency. This type of radar can reject clutter and determine range and velocity of a moving aircraft even when it is flying close to a hill or mountain where the strong landmass echo would block detection with a basic pulse radar. However, pulse Doppler radars are characterized by very high pulse repetition frequency (PRF) and high duty cycle (30% to 50% [1]). This decreases the pulse repetition time (PRT) between pulses, resulting in the possibility of a target echo returning at the time of the next transmission. This, in turn, results in a blind spot in the range.

---

1 Unwanted echoes from land, sea, rain etc.
2.3.3 Antennas and lobes

The antenna in the radar system is the actual piece of hardware through which the electromagnetic wave is transmitted. In order to position an echo, both distance and direction are needed. Radar antennas, unlike radio antennas, therefore have directivity, which also increases its range. Just as the headlights of a car concentrate light, the antenna gathers microwaves in what is called the main beam or main lobe, centred around the bore sight (see Figure 2.3). Diffraction at the edges of the antenna creates undesired back and side lobes. The radar can not determine from which lobe an echo comes – it is always assumed that it originates from the main lobe (in bore sight) [1, 12].

![Lobe Diagram](image)

*Figure 2.3. Lobe diagram.*

2.3.4 Types of Radar

Radar stations and systems can broadly be classified as search or track radars or as a combination of both, normally referred to as track-while-scan (TWS) radars. Search radars are similar to the kind found at airports while track radars provide continuous positional data of a target. A track radar that is used to input target information to a weapon fire control system is often referred to as a fire-control radar.

TWS radars have search and acquisition processes that are performed simultaneously. This enables the radar to track multiple targets while searching for more and to search for targets that were found but then lost. However, this large scan area makes the radar highly vulnerable to jamming.
2.3.5 Scan patterns

The movement of the antenna is called the *antenna scan* or the *scan pattern*. The most basic scan pattern is the *circular scan* (shown in Figure 2.4a) [1], where the antenna just rotates in a full circle. The *sector scan* differs from the circular scan in that the antenna concentrates its attention by moving back and forth across a segment of angle. These scan patterns are often used by search radars with long range to give the operator a complete aerial picture.

![Figure 2.4. Circular scan (a) and spiral scan (b) shown as elevation angle vs azimuth](image)

Typical patterns used for acquisition and tracking are the *spiral* (shown in Figure 2.4b) and *conical scan* [1, 12]. Using these patterns, the radar rotates its beam around the target. In the case of a conical scan, a servo-mechanism adjusts the antenna to keep the echo strength constant for all elevation and *azimuth* angles, meaning that the target is in bore sight. Conical scan systems are inexpensive and often used in systems such as *anti-aircraft artillery* (AAA) or mobile *surface-to-air missile* (SAM) systems but suffer the serious disadvantage of not being able to see a target outside of their narrow scan pattern. This means that a tracked aircraft can easily “escape” if it is successful in breaking track.

Other common scan patterns include helical, raster and palmer scan (see [1]).

---

1 The horizontal component of a direction, measured in degrees.
2.3.6 Radar warning systems

Equipment for detecting radar signals is essential within the field of electronic warfare and is found in countermeasure systems, anti-radiation missiles etc [12]. The purpose of radar warning systems is to identify the presence of threats quickly so that countermeasures and/or manoeuvres can be taken to defeat the impending attack.

By detecting, measuring and analysing radar signals from the surrounding environment, the RWS can determine each radar source as either a SAM, AAA, aerial intercept radar etc. with corresponding characteristics such as pulse length, PRI, track method, angular resolution and range resolution [1, 12, 14].

For detecting radar signals, a fighter aircraft usually has four wing-tip units (WTU) as part of its RWS (see Figure 2.5) [1, 12]. The signals are picked up by the antennas located in the WTUs and are forwarded to the RX receiver.

The RX transforms the signals into digital information which is sent as descriptor words to a pulse processor (PP) which then interleaves/de-interleaves pulses to form emitter descriptions. These are sent to the radar warning computer (RWC) for identification. The threats are finally displayed on the radar threat display (RTD) if classified as hostile and dangerous.

**Figure 2.5. Radar warning system on a fighter.**
2.4 EW Tactics

Tactics, in the military domain, is the arrangement and use of forces in relation to each other and the enemy, or, as defined in [7],

“The employment of units in combat.”

The employment, or use, of radar in battle is most often a trade-off between function and protection because an emitting radar system reveals its position to an enemy receiver much like a submarine reveals itself by using active sonar. As such, a radar should avoid emitting more than necessary to carry out its mission and experience shows that 50% of intermittent sending does not increase acquisition time considerably [12].

As an example of hide-and-seek tactics, consider the situation during the Kosovo Conflict and the use of anti-radiation missiles, such as the AGM-88 HARM (High-Speed Anti-Radiation Missile). Anti-radiation missiles are designed to detect and home in on radar emissions and are therefore a special threat to radar installations. NATO used HARMs extensively during the conflict but was troubled by the guerrilla-like tactics that the Serbs applied to their anti-aircraft defences [12]. These tactics involved camouflaging, regrouping and using radar only for short periods of time while an aerial threat was present. The Serbs also tried to lock on a target from behind, since it takes time for an aircraft to turn around. Alternatively the aircraft can fire sideways or even backwards, but with great loss of precision.

The fact that being silent is an important part of radar tactics will be used in the sample tactic presented next.
2.4.1 Sample tactic and scenario

Consider the situation shown in Figure 2.6. It depicts a red, ground-based team defending a bridge under the threat of the airborne blue team. R₁ is a long-range search radar, R₂ and R₃ are fire-control radars and the threat axis is east.

![Figure 2.6. Red air defence against attacking blue.](image)

Suppose that the red team suspects that blue team is using RWSs (and/or HARMs). The commander of the red team could then lower the risk of losing units by silencing (and perhaps regrouping, although not considered here) R₁ as soon as a target is within the range of R₂ or R₃. This tactic could be expressed in these orders, or rules:

- By default, R₁ sends continuously to detect hostile aircraft. R₂ and R₃ are silent
- R₁ continuously informs R₂ and R₃ of any targets
- Once a target is within the range of R₂ or R₃, R₁ shuts down as R₂ and/or R₃ commence/-s acquisition
- If a fire-control radar loses the target, it shuts down
- If both fire-control radars lose the target then R₁ is again needed

This simple tactic and scenario will be referred to in later chapters.

---

¹ An estimate of the likely direction from which an enemy attack will come.
Chapter 3

Simulation Environment

This chapter describes the rig that is used at SaabTech to support test and evaluation of radar warning systems used for military purposes. The rig creates an artificial environment so that the equipment can be tested under realistic conditions without the issues associated with testing in the real world. Since the focus of this thesis is on tactics development, rig components other than the scenario simulator will not be described in detail.

3.1 Rig Setup

A simplified view of the rig’s components is depicted in Figure 3.1. Together the High Level Scenario Simulator (HSCENS [3]), Emitter Generator (EMGEN) and Radio Frequency Generator (RFGEN) simulate an environment that produces real radar signals, which the four wing-tip units perceive as if they were installed in a real aircraft flying a mission.

Testing the RWS follows the subsequently described procedure. First, a scenario is created or loaded from file using HSCENS. Unit data from the scenario are then transferred to the EMGEN digital system via FTP. After initialisation, the scenario is started and the aircraft takes off. As the scenario progresses, HSCENS continuously sends data about mode (described in section 3.2.1) changes and unit positions to the EMGEN.
EMGEN is capable of handling dense radar signal scenarios (millions of pulses per second [6]) and produces digital pulse data in real time, which are sent to the RFGEN subsystem.

RFGEN is used to convert digital input into analogue signals which then are distributed to the RWS antennas.

The RWS and the radar threat display operate as described in section 2.3.6.

### 3.2 HSCENS

HSCENS is a combined scenario editor and simulator written in Java. It is built from the *model-view-controller* (MVC) design pattern, visualized in Figure 3.2 and described next.

![Figure 3.2. The MVC design pattern.](image)

#### 3.2.1 Model

Units like radar stations and aircraft in HSCENS are called *platforms* and the *subject under test* (SUT) is the platform carrying the RWS. Every platform has a position (latitude, longitude and elevation) and is either part of the red, blue or white (neutral) team. Aerial platforms, such as the SUT, also have speed, roll, pitch, flight path etc. Radar emitting platforms always operate in one of its *modes* (including the standby mode).

A radar mode defines how and with what characteristics a radar emits its electromagnetic waves and is specified through a large set of parameters in the HSCENS GUI. These parameters include power, frequency, pulse width, PRI, sector centre azimuth, sector width and scan pattern.
A radar usually has a number of modes which it chooses between according to its radar controller. The radar controller, also available from the GUI, holds the conditions that must be met before a mode is “turned on”. These conditions are based on:

- Time elapsed since scenario start
- Distance from the radar to the SUT
- Azimuth angle to the SUT
- Elevation angle to the SUT

However, the conditions are specified manually and are not linked to mode parameters. This can result in unrealistic scenarios and the issue will be addressed in Chapter 4.

One platform can carry several emitting systems which makes it easy to add complex units such as battleships or carriers to the model.

### 3.2.2 View

The viewing, or GUI (illustrated in Figure 3.3) part of HSCENS runs in a process of its own and communicates with the controlling part through a socket\(^1\). The presentation is a two-dimensional map even though the model is in 3D.

![Figure 3.3. Illustration of the HSCENS GUI. The part containing the map is from a screen-shot.](image)

---

\(^1\) More information about sockets is found at [http://java.sun.com/](http://java.sun.com/)
3.2.3 Controller

Once the user is done editing, he or she sends the model, or scenario, to the controlling part of HSCENS which is called XSIM. XSIM then computes flight paths for all participating aircraft based on way points set by the user. While running, XSIM continuously:

- Checks for mode changes among radar emitters
- Receives and handles input from the GUI such as ‘stop’ and ‘fast-forward’
- Updates the map
- Sends scenario data (positions, modes etc.) to EMGEN
Chapter 4

Pre-tactics Development

This chapter discusses issues encountered when modelling low level radar operations. It also introduces a commanding unit, the headquarters (HQ), to which all members of a team will be connected.

4.1 Development Overview

The goal is to develop a system that mimics human behaviour which, in this context, can be modelled as a cycle of perception, reasoning and action (more shortly put sense-think-act). This model will be used both for simulating radar stations and their personnel as well as for the HQ, even though the action part often is a direct consequence of the reasoning process\(^1\).

The development process follows a bottom-up approach:

- First, the situational awareness problem for radars will be addressed
- Second, the radars will be given an autonomous behaviour
- Third, a command central, or HQ, will be added to each team. The subordinates of each HQ will store contacts which will be used by the HQ’s data fusion and reasoning process

The reasoning, tactical process of an HQ is the topic of Chapter 5.

\(^1\) The model could also be applied to non-human units such as a homing missile.
4.2 A Threaded Environment

In order not to block HSCENS’s processes of updating the GUI and EMGEN, a new ‘tactics thread’ is proposed. In this thread, platforms go through what they perceive as an endless cycle of sensing, thinking and acting. The thread of the sample scenario from section 2.4.1 (only considering the defending red team) is presented in Figure 4.1.

Figure 4.1. Tactics thread. In the figure, time flows down.

Even though the latest *revolution in military affairs* (RMA, [12]) strives to optimize the decision-making process by networking sensors, military units and command centrals, humans are still a part of the equation. The cycles should therefore contain delays. As noted in [13]: “No model has any inherent value of its own. The value of every model is based entirely upon the degree to which it solves someone’s real world problem. Accuracy and fidelity are driven by the problem that the model is supposed to solve.” The frequency at which radars are polled should be investigated, but in a first prototype, once a second will do.

---

1 A thread is a single sequential flow of control within a program.
4.3 Radar Operations Model

Radar stations act autonomously after an order has been given. They have a predictable behaviour and try to carry out their orders using available modes in response to units in their local environment.

4.3.1 Radar sensing

A typical search radar completes a scan pattern in 4 seconds and has a beam width\(^1\) of 2° [12]. A radar which scans a full 360 with these characteristics would require to be polled 45 times per second in order to cover the full circle\(^2\). Detecting aircraft or missiles with high angular velocity versus the radar at close range would require an even higher sampling frequency. However, since the purpose of the simulator is to test the radar warning system, not the radar stations and since computer resources are needed elsewhere, a much simpler detection scheme is proposed.

A target is considered detected if it resides within the two-dimensional (latitude and longitude) sector that a radar’s mode defines (see Figure 4.2). Elevation is not considered in this detection scheme because of time constraints. Beam width is used for modes that do not have a sector defined.

![Figure 4.2. Radar sector.](image)

The maximum range, \(R_{\text{max}}\), is computed using a version of the radar range equation (see Appendix A.2). Positions of platforms in HSCENS are specified using latitude and longitude, rather than x- and y-coordinates. For a description on how distance and direction between two platforms are calculated, please refer to Appendix A.3 and A.4 respectively. Once the range and direction from a radar to another platform is known, it is straight forward to determine whether the platform is in the field of regard of the radar or not. Every radar keeps a list of contacts including time and position of the latest detection.

---

\(^1\) The angular distance between the two offset angles (in a single plane) between the two points at which the antenna gain is down to 50% of the peak value

\(^2\) Assuming that the radar only detects objects within the 2° field of regard
4.3.2 Radar thinking and acting

A fire-control radar typically emits a narrow, intense beam of radio waves to ensure accurate tracking information. Before such radars can use such a narrow beam, they:

- Must first be directed to the general location of the target
- Home in on the target by the use of a predetermined search pattern

To express this target-processing sequence, *radar states*\(^1\) are introduced [1, 6]. These states will increase the abstraction level from modes containing electrical variables and will also be used for graphical representation (grey sector for SEARCH, red for TRACK for example). At any point in time, every radar is operating in one of its states. While a pure search radar is either off or in its SEARCH state, a TWS radar can operate in all states. The states are described in Table 4.1.

*Table 4.1. Radar states.*

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACQUISITION</td>
<td>The radar is trying to lock on a target.</td>
</tr>
<tr>
<td>ACQUISITION COAST</td>
<td>The radar lost the acquisition.</td>
</tr>
<tr>
<td>STANDBY*</td>
<td>The radar is off.</td>
</tr>
<tr>
<td>SEARCH*</td>
<td>The radar tries to determine the presence of potential targets. This may be done by scanning a full 360° or in a certain sector.</td>
</tr>
<tr>
<td>SEARCH ASSIGNED*</td>
<td>The radar is assigned a target which it may or may not already have detected.</td>
</tr>
<tr>
<td>TRACK</td>
<td>The radar has established a lock on a target.</td>
</tr>
<tr>
<td>TRACK COAST</td>
<td>The radar lost the track.</td>
</tr>
</tbody>
</table>

* State that can be ordered from HQ.

\(^1\) Target-processing sequence steps are often referred to as ‘modes’, but to avoid confusion with the mode concept described in section 3.2.1, ‘state’ will be used.
Figure 4.3 depicts the states and the possible transitions between them. A transition occurs once its corresponding requirement is met. For example, in order for a radar to go from ACQUISITION to TRACK, it must detect the target a certain number of times (pulse radar) or seconds (CW radar) with an S/N ratio\(^1\) above a certain level\(^2\). If the target is lost, the state of a radar starts trickling down to its original SEARCH or SEARCH ASSIGNED state.

![Radar States and Transitions Diagram]

---

When entering a state, a radar needs to choose one and only one mode (except for the STANDBY state, of course). If a radar has more than one mode available for a certain state, a way of selecting the most suitable is needed. In the proposed model, a fairly simple scheme is used. For example, the algorithm used when entering the TRACK state searches for modes whose range covers the target and among these, the scan pattern with the highest ‘TRACK score’ is used. If there are more then one mode with the same scan pattern available, the one with the shortest range (still reaching the target) is chosen.

In a more elaborate model, the user should be able to override the automatic procedure described above by associating modes to states and specifying the transition requirements for different platforms. The radar could then behave in a unique manner and in accordance with target type, distance, weather conditions, jamming conditions etc. For example, an anti-ship missile radar typically uses sector scan while an airborne fire-control radar often uses raster scan during acquisition [1].

---

\(^1\) Signal-to-noise ratio: the power ratio between a signal (meaningful information) and the background noise.

\(^2\) These numbers are specific for different radar- and weapon systems.
4.4 Connecting the Dots

A new unit is introduced to the simulator: the HQ. The HQ (a new Java class) is connected to all members of its team and will implement tactics by coordinating and commanding its subordinates in response to the progressing situation.

4.4.1 HQ sensing

The HQ starts its cycle by filtering the contacts of its subordinates in order to get an aerial picture which is up to date. This process is shown by example in Figure 4.4, where radar stations $R_1$, $R_2$ and $R_3$ are reporting in their contacts of the blue team to the Red HQ.

Three radar stations reporting in the same target should render one contact, not three and so the latest observation is used. This type of process is often referred to as data fusion.

4.4.2 HQ thinking and acting

Normal Java methods will be used for sending orders and instructions. Under what circumstances these methods will fire and how these circumstances should be specified will be discussed next.
Chapter 5

Rule-Based Programming and Tactics

The idea behind rule-based programming is to specify rules and facts instead of writing instructions in the usual, linear way. This and later chapters contain arguments that a rule-based approach of implementing high-level radar tactics has several advantages over conventional, procedural coding.

“A rule-based system is a computer program that uses rules to reach conclusions from a set of premises.” [5]

5.1 The Need for a Rule-Based System

In procedural programming, the programmer tells the computer what to do, how to do it, and in what order. This style of programming is well suited for problems in which the inputs are well specified and for which a known set of steps can be carried out to solve the problem. Code for solving equations (such as the radar range equation, Appendix A.2) and rendering graphics are best written procedurally.

In order to introduce tactical behaviour to units or sets of units, high-level control code is needed, which could be referred to as artificial intelligence (AI). As a contrast to say, mathematical computations, AI code tends to be condition rich, requiring many tests and conditional branching based on the results of those tests [16]. To see this, consider the questions that an air defence commander might pose: where is the enemy now and what is he doing? Can we predict what he will do next? What are his strengths and weaknesses? What are ours? How can we maximize protection while carrying out our mission? To summarise – what actions do we take in response to the current situation?

A procedural implementation codes condition checking as a succession of ordered conditional statements. Developing a procedural version of a reasoning engine may soon, depending of the complexity, contain dozens of levels of nested if statements. This kind of code is difficult to manage and understand.
Rule-based systems are not procedural, but declarative. Declarative, or data-driven programs require a different kind of programming in which a runtime system is used to make scheduling and flow of control decisions. Declarative programming is often a natural way to tackle problems without clear algorithmic solutions such as situation awareness, classification, prediction and control. Also, a declarative program can be more flexible in the face of fragmented or poorly conditioned inputs [5].

A rule-based program does not consist of a long sequence of instructions. Instead, it is made up of discrete rules, each of which applies to some subset of the problem.

Rule-based programs that capture the knowledge of human experts in a certain field of expertise are called expert systems and became a success story for AI research in the 1970s and 1980s [5]. Apart from military simulation environments, such as the TacAir-Soar System [8], rule-based engines have been used to develop a broad range of commercial software, including:

- Expert systems that evaluate discount pricing, insurance claims and mortgage applications
- Agents that predict stock prices and buy and sell securities
- Network intrusion detectors and security auditors
- Design assistants that help mechanical engineers
- Smart network switches for telecommunications
- Servers to execute business rules
- Intelligent e-commerce sites
- Games
5.2 The Structure of Rule-Based Engines

A typical rule engine contains facts, rules and an inference engine which fire actions, as depicted in Figure 5.1.

![Diagram of a rule engine](image)

**Facts**
- “Item #123 is expensive speakers”
- “Gold cables are expensive”
- “Bob is a customer”
- “Bob is buying item #123”

**Rules**
- “Recommend appropriate cables when customer buy speakers”

**Inference Engine**

**List of Actions to Take**
- “Recommend that Bob buy gold cables”

**Figure 5.1. Parts of a rule engine.**

Facts are gathered in a fact base, or the working memory (WM) of the rule engine.

5.2.1 The working memory

In a typical rule engine, the working memory contains all the pieces of information with which the rule-based system is working. The working memory is somewhat akin to a relational database to make searching a very fast operation. Some working memories can hold only objects of a specific type while others can include, for example, Java objects.

5.2.2 The rule base

A rule is a kind of instruction or command that applies in certain situations. Rules are a lot like the if-then statements of traditional programming languages. The if part of the rule written in this form is often called its left-hand side (LHS), predicate, or premises and the then part is the right-hand side (RHS), actions, or conclusions.
Two rules from the sample tactic in section 2.4.1 might look like these, in pseudo code:

IF  
   all radars are off
THEN  
   start search radar
END  

IF  
   search radar has contact with target
AND  target is in range of a track radar
THEN  
   start the track radar
AND  shut down search radar
END

A typical rule-based system contains hundreds or thousands [5] of rules which are gathered in what is known as the rule base.

5.2.3 The inference engine
The inference engine is the central part of a rule engine and its primary task is to match rules to data. The purpose of the pattern matcher is to decide which rules apply, given the current contents of the working memory.

Once the inference engine figures out which rules should be fired, it still must decide which rule to fire first. The list of rules that could potentially fire is stored on the agenda, which then uses a conflict strategy to decide which of the rules have the highest priority and should be fired first. As an example, consider these two rules:

Rule 1  
IF  
   ordered into SEARCH
THEN  
   search
END

Rule 2  
IF  
   radiation time exceeds 10 min
THEN  
   stop sending for 2 min
END

Since both rules may apply at the same time, either one can be given a higher priority to solve the conflict.

Finally, once the rule engine decides what rule to fire, the execution engine fires the action part of the rule. Modern rule engines often offer a complete programming language that can be used to define what happens when a rule fires, including invocation of compiled code.
5.2.4 The Rete algorithm

The Rete algorithm was designed by Dr. Charles L. Forgy of Carnegie Mellon University in 1979 [4]. Rete (pronounced “ree-tee”) is Latin for “net” and the algorithm is used to increase the run-time efficiency of rule processing. The details of the Rete algorithm are omitted in this thesis but, briefly, the Rete algorithm maintains a network of dependencies between rules and data to allow run-time caching of partial results to reduce the amount of condition checking required [5, 16].

5.3 Existing Rule-Based Systems

There are many rule-based systems on the market – some freely available, some targeted at enterprise use. Jess [5], described in the next section, was developed in the late 1990s at Sandia National Laboratories in Livermore, California. Many of the core concepts of Jess were originally derived from CLIPS1, which was itself influenced by early rule engines like OPS5 and ART.

Other rule-based systems and language extensions include JENA2, Pellet3, JTP4, OPSJ5, JRules6 and RC++ [16].

---

1 http://www.ghg.net/clips/CLIPS.html
2 http://jena.sourceforge.net/
4 http://www.ksl.stanford.edu/software/JTP/
5 http://www.pst.com/opsj.htm
6 http://www.ilog.com/products/jrules/
5.4 Jess

Jess (Java™ Expert System Shell) \(^1\) is a rule engine and scripting environment. As a mean of investigating whether a rule-based system is suitable for implementing tactics in a RWS simulator, Jess was chosen because it is:

- Written entirely in Java
- Small (a few hundred kilobytes)
- Freely available for academic use

Because Jess is Java-based, it is platform independent and allows for easy integration with other Java-based applications, such as HSCENS. Jess can directly access all Java classes and libraries. Likewise, the HSCENS code can access all parts of the Jess library. This capability is shown schematically in Figure 5.2.

Jess, like most rule-based systems, uses the Rete algorithm to process rules.

5.4.1 The working memory of Jess

The working memory of Jess can contain ordered facts, unordered facts and shadow facts, each type of fact useful in certain situations. Unordered facts are general-purpose facts and are quite literally like rows in a relational database table, as shown in Figure 5.3.

```
Jess> (deftemplate contact
(slot id)
(slot lat)
(slot long))
Jess> (assert (contact (id "Air20") (lat 59.41) (long 15.82)))
Jess> (assert (contact (id "Air20") (lat 59.41) (long 15.82)))
```

```
<table>
<thead>
<tr>
<th>CONTACT</th>
<th>Id</th>
<th>Lat</th>
<th>Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air20</td>
<td>59.41</td>
<td>15.82</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.3. Unordered facts are similar to rows in a relational database table.

Ordered facts are useful for small bits of information such as constants.

\(^1\) http://herzberg.ca.sandia.gov/jess/
Finally, shadow facts are unordered facts that are linked to Java objects (JavaBeans\(^1\)) outside of Jess. Therefore, shadow facts serve as a connection between the working memory and the Java application inside which Jess is running.

### 5.4.2 Rules, XML and Jess

Rules can be created and stored using XML\(^2\) and then translated into the native rule language of rule engines such as Jess. This approach keeps the development of rules vendor-neutral and many high-quality parsers and APIs are available for working with XML directly from a Java program such as HSCENS.

However, developing rules with XML also has a couple of serious drawbacks. First, XML is not very readable. Second, development has to be restricted to the common core concepts that can be expressed in all rule languages.

### 5.4.3 Editing facts and rules

Jess includes a development environment called the *Jess Developer’s Environment* (JessDE) which is supplied as a set of plug-ins for Eclipse\(^3\). Extensions are also available for Emacs\(^4\), jEdit\(^5\) and EditPlus\(^6\).

### 5.4.4 Jess performance

Using Sun’s HotSpot JVM on an 800 MHz Pentium III, Jess can, according to [5]:

- Fire more than 80,000 rules per second
- Perform almost 600,000 pattern-matching operations per second
- Add more than 100,000 facts to working memory per second
- Outperform CLIPS by a factor of 20 or more on the same hardware

### 5.5 Further Reading

For more on rule-based systems and related topics, please refer to [2, 4, 5, 8, 9, 10, 16].

---

2. Extensible Markup Language
3. Eclipse is an extensible, open source IDE (integrated development environment).
5. [http://www.jedit.org](http://www.jedit.org)
6. [http://www.editplus.com](http://www.editplus.com)
Chapter 6

Implementation

The ideas developed in Chapter 4 and Chapter 5 were implemented and tested. This chapter shortly describes the realization of Jess as a rule engine for HSCENS and the sample tactic of Chapter 2. But first a very simple non-tactical scenario (a good starting point as suggested in [11]) is presented.

6.1 Simple scenario

The ideas and models of Chapter 4 were implemented using Java directly in HSCENS. The colour codes of the radar states introduced in chapter 4.3.2 proved to be very useful during the tests that were carried out.

The screenshots of Figure 6.1 show a very simple scenario, which nevertheless encapsulates much of Chapter 4. The detection scheme (see Chapter 4.3.1, Appendix A.2, A.3 and A.4 respectively) is in place and the R1 radar continuously tracks the target. As the target gets closer, R1 is able to switch to a mode with shorter range. Not apparent from the two screenshots the radar station also changes state and reports the target to its HQ throughout the scenario.
Figure 6.1.

(a) Target is detected  
(b) As the target comes closer, the radar chooses a different mode

6.2 Scenario using Jess

Starting the inference engine of Jess is as easy as the following Java code snippet:

```java
import jess.*;
...
Rete engine = new Rete();
engine.reset();
engine.executeCommand("(batch tactics.clp)");
engine.run();
```

However, the engine needs to be able to reason directly about and react to data available in the HSCENS model. When the user presses 'Start' in the HSCENS GUI, the new HQ Java class fills the Jess working memory with data from the scenario. This process is automatic and done for all scenarios without any intervention from the user. For example: there is always a ?redHQ Jess variable available to use in any script and if there is a “radar5” in a scenario, the user may use the ?radar5 variable anywhere in the script without any manual declaration or initialization.

Additional tactical data are also provided from the HQ class to Jess within a scenario.
If the user does not want to specify tactical behaviour, he or she can use a default tracking behaviour for all radar stations by including one single line of code in the Jess script:

```java
(?redHQ orderSearchAll())
```

The scenario of Chapter 2.4.1 is presented in Figure 6.2. The tactical rules involved were specified in very few lines of Jess code as all functionality, such as the radar operations model and the data fusion process, is “hidden” in conventional Java code.
(a) $R_1$ has detected an aircraft and continuously informs $R_2$ and $R_3$ of its position
(b) Once in range the track radars commence acquisition. $R_1$ shuts down
(c) The aircraft is out of range of $R_3$ and so it shuts down
(d) As both track radars have lost the target, $R_1$ is again needed
Due to lack of time, rules were written directly in the Jess scripting language, not XML.

6.3 Third Party Software

Development was done directly from HSCENS source code using JBuilder, Eclipse, EditPlus 2 and Jess 7.0b1 on the Windows XP OS.
Chapter 7

Discussion

This chapter concludes the thesis by a presentation and discussion of the results. It also points out ideas for future work.

7.1 Pros and Cons of the Rule-Based Approach

The evaluation of a declarative, rule-based approach to develop tactics was made in comparison to procedural programming and based on the experience of integrating Jess with HSCENS. The most important pros and cons are presented next.

7.1.1 Pros

Rules separate high-level tactics from implementation logic. Extracting the tactical behaviour out of the Java code simplifies the core application and enables radar- and air defence experts to work with scenarios in parallel with engine developers.

It is easier to understand and overview tactics when low-level algorithms, loops and branches are hidden.

Individual rules can be added, altered or removed more easily than changing the logic within procedural code.

There is no need to recompile the application while developing and testing scenarios since the rules are not part of the source code.

Rules are saved as text files and can therefore easily be distributed and used by different users. This is a great advantage: developers can create, edit, gather, fetch and debug scenarios and rules over a network. As long as the primary application is backward compatible, ‘old’ rules survive software upgrades.

The inference engine automates rule checking and shares condition checks between rules to increase run-time efficiency and reduce code bloat. It listens to changes in the model: no looping or polling is needed from the programmer.
If written in (or converted to) XML, rules can survive the migration to other rule-based system.

If licensed or sold, end users can be allowed to change the behaviour of units in a simulator. This without revealing the source code – only the interface is given.

### 7.1.2 Cons

Programmers and users must learn and understand yet another tool. This hurdle should not be underestimated – declarative programming requires a slightly different set of mind compared to the one used when writing normal, procedural code. Also, developers must put time and effort into a new scripting language: new syntax, new methods.

Code is required to bridge the original application to the rule engine. Also, documentation should be provided to the rule author so that he or she knows what tools are at his or her disposal.

The Rete algorithm explicitly trades space for speed, so memory usage might become an issue for very large scenarios and sets of rules.

In addition, Jess is coupled with problems and costs of its own, most notably:

- Type conversion when calling overloaded methods in Java code. For example, if the string “TRUE” is passed to a Java method that is overloaded to take either a boolean or a String, it is generally impossible to predict which overload Jess will choose. In these cases, an explicit wrapper class can usually fix the problem [5]
- No special interface for working with multidimensional arrays [5]
- License fees for commercial use

### 7.2 Conclusion

For the purpose of developing and expressing tactics in a combined simulator and editor, the conclusion of this thesis is that the advantages of a rule-based system clearly outweigh the drawbacks. This is especially true if tactics and behaviour of units are accessed and changed frequently. Furthermore – several of the issues associated with integrating a rule-based system with an already existing simulator and editor such as HSCENS are of ‘solve-once’ character.

Because of time constraints, alternatives to Jess where never evaluated. Therefore, *which* rule-based system is most appropriate for SaabTech remains an open question.
7.3 Future Work

The list of possible enhancements of software used for simulating the surroundings of an aircraft and its RWS seems to be never-ending. This list includes:

- Developing advanced detection schemes used by radar systems by considering the effects of chaff, *radar horizon*, topology and line of sight, radar cross section, *radar resolution cell*, weather conditions, echoes in back and side lobes, *ghost echoes*, clutter, jamming etc.
- A detection scheme for radar stations using modes with scan patterns whose field of regard is not well represented by a sector. Also, the approximation of a radar station’s ability to detect targets could be deleted by use of hardware.
- Automate target selection, radar assignment and *emissions control*\(^1\) (EMCON). If a runway is to be defended, priority should be given to enemy bombers since these are expected to carry heavy payload. If the mission is to protect mechanized infantry, priority should be given to enemy helicopters. The field of *operations research* could be used to investigate ways of assigning radars to targets.
- Developing an algorithm for determining the threat axis for a set of units which then adjust their tactics correspondingly.
- Evaluate the pros and cons of introducing an *agent communication language* (ACL). For more on ACLs, please refer to [2].

\(^1\) EMCON is basically the manner in which commanders determine who may radiate, and under what circumstances.
7.3 Acknowledgments

First and foremost I would like to thank the staff of the Electronic Warfare Systems Division at SaabTech including my manager Jan Wennström, who pointed out great resources for the project and who patiently answered my questions about radar technology, air defence tactics etc. I would also like to thank my supervisor Mikael Eriksson who helped me with coding considerations and explaining the details of HSCENS.

I would like to thank my supervisor Assistant Professor Olle Bälter (Royal Institute of Technology, Department of Numerical Analysis and Computer Science) for showing interest in the project and for his comments on my ideas. Furthermore I would like to thank Assistant Professor Rand Waltzmann for his comments on rule-based programming and Jess.
References


Appendix

A.1 Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>Anti-aircraft artillery</td>
</tr>
<tr>
<td>ACL</td>
<td>Agent communication language</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial intelligence</td>
</tr>
<tr>
<td>CW</td>
<td>Continuous wave</td>
</tr>
<tr>
<td>EMGEN</td>
<td>Emitter Generator</td>
</tr>
<tr>
<td>EW</td>
<td>Electronic warfare</td>
</tr>
<tr>
<td>HARM</td>
<td>High-speed anti-radiation missile</td>
</tr>
<tr>
<td>HQ</td>
<td>Headquarters</td>
</tr>
<tr>
<td>HSCENS</td>
<td>High Level Scenario Simulator</td>
</tr>
<tr>
<td>JessDE</td>
<td>Jess Developer’s Environment</td>
</tr>
<tr>
<td>LHS</td>
<td>Left-hand side</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-view-controller</td>
</tr>
<tr>
<td>PP</td>
<td>Pulse processor</td>
</tr>
<tr>
<td>PRI</td>
<td>Pulse repetition interval</td>
</tr>
<tr>
<td>PRF</td>
<td>Pulse repetition frequency</td>
</tr>
<tr>
<td>PW</td>
<td>Pulse width</td>
</tr>
<tr>
<td>RFGEN</td>
<td>Radio Frequency Generator</td>
</tr>
<tr>
<td>RHS</td>
<td>Right-hand side</td>
</tr>
<tr>
<td>RMA</td>
<td>Revolution in military affairs</td>
</tr>
<tr>
<td>RTD</td>
<td>Radar threat display</td>
</tr>
<tr>
<td>RWS</td>
<td>Radar warning system</td>
</tr>
<tr>
<td>SAM</td>
<td>Surface-to-air</td>
</tr>
<tr>
<td>SUT</td>
<td>System under test</td>
</tr>
<tr>
<td>TWS</td>
<td>Track-while-scan</td>
</tr>
<tr>
<td>WTU</td>
<td>Wing-tip unit</td>
</tr>
</tbody>
</table>
A.2 The Radar Range Equation

The radar range equation is given by:

$$\text{Maximum detection range} \approx \frac{P \cdot \tau \cdot G^2 \cdot \sigma \cdot \lambda^2}{(4\pi)^3 \cdot \text{SN} \cdot k \cdot T \cdot L}$$

where

- $P$ = transmitted power [W]
- $\tau$ = pulse width [s]
- $G$ = antenna gain factor [not in dB]
- $\sigma$ = radar cross section [m²]
- $\lambda$ = wavelength [m]
- $\text{SN}$ = signal to noise factor [not in dB]
- $k$ = Boltzmann’s constant $1.38 \cdot 10^{-23}$ [J/K]
- $T$ = system temperature [K]
- $L$ = loss factor [not in dB]

The equation applies to pulsed radar. For Doppler radars, the *pulse width* variable is replaced by $1/B_r$ ($B_r =$ receiver bandwidth).

In the model the radar is assumed to be monostatic which means that the same antenna is used for transmission and reception and hence, the $G^2$ factor.

*Radar cross section* is the measure of a target’s ability to reflect radar signals in the direction of the radar receiver.

The *signal to noise factor* is the amount of times stronger the signal has to be compared to the mean background noise energy in order for the radar to distinguish the target.

The model, as stated above, is a very simple mean of calculating the range of a radar and could be extended in many ways. For example, the *loss factor* could be divided into transmission losses, scan pattern losses, atmospheric losses etc. The *Radar cross section* could be based on target models (ranging from bombers to stealth fighters) and the orientation of the target as viewed from the radar.

For more on the radar range equation, see [1, 12, 14].
A.3 Distance Between Two Positions

Latitude is an angular measurement ranging from $0^\circ$ at the equator to $\pm 90^\circ$ at the north and south pole respectively. Longitude is ranging from $0^\circ$ at the Prime Meridian to $+180^\circ$ eastward and $-180^\circ$ westward.

![Figure A.1. Two positions on the globe.](image)

The distance ($a$) between pos$_1$ and pos$_2$ (see Figure A.1) can be calculated using the law of cosines for the sides of a spherical triangle, which states that

$$ \cos a = \cos b \cos c + \sin b \sin c \cos A $$

Solving for $a$ gives:

$$ a = \arccos(\cos b \cos c + \sin b \sin c \cos A) $$

However, by using the Pythagorean theorem and the fact that longitudes are shorter towards the poles, one can lessen the number of trigonometric functions needed through an approximation:

$$ a = \sqrt{(lat_1 - lat_2)^2 + \left(\cos\left(\frac{lat_1 + lat_2}{2}\right)\right)^2} $$

The approximation is acceptable for normal ranges of radar stations. Elevation is not considered. To express the distance in meters, $a$ has to be multiplied with the earth radius$^1$.

---

$^1$ http://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html
A.4 Direction Between Two Positions

The direction, or heading, from pos1 to pos2 is measured in degrees clockwise from north and is found by using the spherical analogue of the Law of Sines:

\[
\frac{\sin a}{\sin A} = \frac{\sin b}{\sin B} = \frac{\sin c}{\sin C}
\]

Rewrite:

\[
\sin B = \frac{\sin A \sin b}{\sin a} = \frac{\sin A \sin (90^\circ - \text{lat}_2)}{\sin a} = \frac{\sin A \cos \text{lat}_2}{\sin a}
\]

since \(b\) is the polar distance of pos2 \((90^\circ - \text{lat}_2)\). Solving for \(B\):

\[
B = \arcsin \left( \frac{\sin A \cos (\text{lat}_2)}{\sin a} \right)
\]

Note that (see Figure A.1):

- \(A\) is the longitude difference between pos1 and pos2
- \(a\) is the distance between pos1 and pos2 in radians

Since the \(\arcsin\), or inverse sine function ranges from \(-90^\circ\) to \(90^\circ\) and heading is given from \(0^\circ\) to \(360^\circ\), the following code snippet is introduced to correct the \(B\) angle. Imagine pos1 at the origin in a coordinate system and pos2 in one of the quadrants:

```c
if(lon2 > lon1 && lat2 < lat1) { // Q4
    B = 180 - B;
} else if(lon2 < lon1 && lat2 < lat1) { // Q3
    B = 180 + B;
} else if(lon2 < lon1 && lat2 > lat1) { // Q2
    B = 360 - B;
} else if(lon2 > lon1 && lat2 > lat1) { // Q1
    B = 360 - B;
} else if(lat1 == lat2) { // Q0
    B = 0;
} else if(lat1 > lat2) { // Q5
    B = 360 - B;
}
```
