REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	14 October 2016	

Grundsatzbetrachtung der FLAMES Runtime Suite

(Basic Consideration Of The FLAMES Runtime Suite)

6. AUTHOR(S)

Alisan Öztürk

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

UNIBW

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES

Helmut-Schmidt-Universität
Holstenhofweg 85
22043 Hamburg Germany

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Text in German.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Public release. Copyrighted. (1 and 20)

12b. DISTRIBUTION CODE

ABSTRACT (Maximum 200 words)

Modeling of combat operations and the resulting gain in knowledge, as well as derived recommendations for action are considered since the availability of the first computer. More powerful computing cores, as well as the ability to switch a computer network from computers, allow ever faster calculations of increasingly complex simulation models. While it is customary to develop the required simulation from scratch for a specific task, there are also software solutions on the market that provide a framework for modeling and promise a faster and more favorable development of simulation models due to their modular structure. An in-house development is not only expensive and time-consuming, but is usually no longer used for a follow-up study. Among other things, this is due to the fact that the developed models are developed for a specific research question and are simulation-system specific. On the other hand, it is possible to remove individual modules from past models and to adapt them easily. This is a sustainable approach that has so far received little attention.

Machine assisted translation.

14. SUBJECT TERMS

UNIBW, German, FLAMES runtime suite

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19, SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Angewandte Mathematik und Optimierung Schriftenreihe
Applied Mathematics and Optimization Series
AMOS # 54(2017)

Alisan Öztürk

Grundsatzbetrachtung der FLAMES Runtime Suite

Herausgegeben von der
Professur für Angewandte Mathematik
Professor Dr. rer. nat. Armin Fügenschuh

Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg
Fachbereich Maschinenbau
Holstenhofweg 85
D-22043 Hamburg

Telefon: +49 (0)40 6541 3540
Fax: +49 (0)40 6541 3672

e-mail: appliedmath@hsu-hh.de
URL: http://www.hsu-hh.de/am

Angewandte Mathematik und Optimierung Schriftenreihe (AMOS), ISSN-Print 2199-1928
Angewandte Mathematik und Optimierung Schriftenreihe (AMOS), ISSN-Internet 2199-1936

Alisan Öztürk

Grundsatzbetrachtung der FLAMES Runtime Suite

Seminararbeit

Fakultät für Maschinenbau

Studiengang:	Wirtschaftsingenieurwesen
Matr.-Nr.	869950
Eingereicht:	14. Oktober 2016
Prüfer:	Prof. Dr. Armin Fügenschuh

In Zusammenarbeit mit dem Planungsamt der Bundeswehr Abt IV UA III

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst, keine anderen als die im Quellen- und Literaturverzeichnis genannten Quellen und Hilfsmittel, insbesondere keine dort nicht genannten Internet-Quellen benutzt, alle aus Quellen und Literatur wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe und dass die auf einem elektronischen Speichermedium abgegebene Fassung der Arbeit der gedruckten entspricht.

Hamburg, ...

(Datum) (Unterschrift)

Zusammenfassung

Die vorliegende Seminararbeit untersucht das Simulationsframework FLAMES, welches als Simulationswerkzeug für den Einsatz im Referat IV 3 (2) des Planungsamts der Bundeswehr überprüft wird.

Die zugrundeliegende Aufgabenstellung war hierbei, die Funktionalitäten der vorliegende Version von FLAMES (*Runtime Suite*) herauszustellen, und zu erarbeiten, wo die Grenzen der Software liegen. Dabei wurde der Aufbau der Software, sowie eines Beispielszenarios, systematisch untersucht und beschrieben, wie FLAMES versucht, menschliches kognitives Verhalten zu ermöglichen.

Neben den in dieser Arbeit - auf die Version beschränkten - demonstrierten Beispielen, wird dem Leser ein Überblick aus öffentlich zugänglichen Implementierungen von Simulationssoftware, welche auf Basis des FLAMES-Frameworks entwickelt wurden, sowie Modifikationen der Software durch die Entwicklerversion gegeben.

Die Seminararbeit schließt mit konkreten Empfehlungen an anschließende Untersuchungen. Dabei zeigt sich im Ergebnis der Untersuchung, dass das Simulationsframework durchaus viele Vorteile hat, welche sich auch das Planungsamt zu Nutze machen kann. Allerdings reicht die alleinige Auswertung der vorliegenden Version nicht aus, sondern erfordert eine - auf diese Arbeit aufbauende - weiterführende Untersuchung der Entwicklerversion.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ACO	Airspace Control Orders
ADC	Air Defence Controller
ATC	Air Traffic Controller
ATO	Air Tasking Orders
C2WSPTT	Command and Control Weapon System Part Task Trainer
CAOC	NATO Combined Air Operations Centres
CD&E	Concept Development & Experimentation
DIS	Distributed Interactive Simulation
DTED	Digital Terrain Elevation Data
FACT	FLAMES Advanced Correlated Terrain
FIRE	FLAMES Interactive Runtime Executable
FLAMES	FLexible Analysis and Mission Effectiveness System
FLASH	FLAMES Scenario Highlighter
FOI	Swedish Defence Research Agency
FORGE	FLAMES Operational Requirements Graphical Editor
GeoInfoDBw	Geoinformationsdienst der Bundeswehr
GUI	Graphical User Interface
HLA	High Level Architecture
ICC	Integrated Command and Control Software for Air Operations
ITC	Integrated Training Capability
JREAP	Joint Range Extension Applications Protocol
LVC	Live, Virtual and Constructive
M&S	Modellbildung und Simulation

NetOpFü	Vernetzte Operationsführung
NGA	National Geospatial-Intelligence Agency
OR	Operations Research
USMTF	US Message Text Format

1 Einleitung

Modellierung von Gefechtshandlungen und ein daraus resultierender Erkenntnisgewinn, sowie daraus abgeleitete Handlungsempfehlungen werden schon seit Verfügbarkeit der ersten Computer in Betracht gezogen. Dabei ermöglichen leistungsfähigere Rechenkerne, sowie die Möglichkeit, ein Rechennetzwerk aus Computern zu schalten, immer schnellere Berechnungen von immer komplexer werdenden Simulationsmodellen. Während es üblich ist, die geforderte Simulation für eine bestimmte Aufgabenstellung von Grund auf zu entwickeln, existieren auf dem Markt auch Softwarelösungen, die ein Grundgerüst zur Modellbildung bieten und durch ihre modular aufgebaute Struktur eine schnellere und günstigere Entwicklung von Simulationsmodellen versprechen. Eine Eigenentwicklung ist dabei nicht nur teuer und auch zeitintensiv, sondern findet meistens für eine Anschlussstudie keine Verwendung mehr. Das liegt unter anderem daran, dass die entwickelten Modelle für eine konkrete Fragestellung entwickelt werden und simulationssystemspezifisch sind. Dem gegenüber steht die Möglichkeit einzelne Bausteine aus vergangenen Modellen zu entnehmen und unkompliziert anzupassen. Dies ist ein nachhaltiger Ansatz, der bisher wenig Beachtung gefunden hat.

Auch die Bundeswehr nutzt die Modellierung und Simulation von Gefechtshandlungen in unterschiedlichsten Anwendungsgebieten. Das Referat IV 3 (2) des Planungsamts der Bundeswehr benötigt zur Unterstützung der eigenen Modellbildung und Simulation (M&S) eine Softwarelösung, welche die oben genannten Vorteile bietet. Die Software soll als unterstützendes Werkzeug eine bessere und schnelle Beurteilung von - an das Referat herangetragene - Fragestellung ermöglichen. Dazu wird in dieser Arbeit eine Grundsatzbetrachtung des FLAMES Simulationsframeworks von der US-amerikanischen Ternion Corporation durchgeführt. Dem vorangestellt ist eine Betrachtung der Grundlagen im Bereich Modellbildung und Simulation. Nachstehend wird die Softwarestruktur von FLAMES vorgestellt und auf den Programmaufbau eingegangen. Nachdem sowohl Szenarioaufbau, als auch wichtige Bedingungsfelder dargestellt wurden, werden anhand eines Beispielszenarios Fähigkeiten und Grenzen der Software demonstriert. Da die vorhandene Softwarelizenz beschränkt ist und diese Arbeit als einen ersten Schritt zur Untersuchung von FLAMES dient, werden - zur Vervollständigung der Möglichkeiten - Anwendung anhand öffentlich zugänglicher Fallstudien und veröffentlichten wissenschaftlichen Artikeln vorgestellt. Die Ternion Corporation behauptet auch, dass die Software die Methode Data Farming unterstützt. Da diese Fähigkeit ebenfalls von Interesse für das Referat ist, wird diese Aussage gleichermaßen untersucht. Die Arbeit schließt mit einem Fazit und liefert einen Ausblick für zukünftige Untersuchungen von und mit FLAMES. Hieraus lassen sich konkrete Handlungsempfehlungen für das Referat ableiten.

Planungsamt

Das Planungsamt der Bundeswehr ist direkt dem Bundesministerium der Verteidigung unterstellt und ist für den nicht-ministeriellen Anteil des Planungsprozesses verantwortlich. Die Aufgaben des Planungsamts sind zentraler Bestandteil der Weiterentwicklung der Bundeswehr. Das Planungsamt ist unterteilt in vier Abteilungen. Die Abteilung IV - Wissenschaftliche Unterstützung und Interoperabilität - ist ebenfalls in vier Unterabteilungen mit insgesamt elf Referaten aufgeteilt und in Abb. 1.1 dargestellt. Während das Planungsamt seinen Hauptsitz in Berlin hat, befinden sich zwei Unterabteilungen der Abteilung IV in Taufkirchen. Diese befassen sich mit Concept Development & Experimentation (CD&E), Operations Research (OR), M&S und Architektur, wobei die Unterabteilung IV 3 als Hauptaufgabe die Anwendungs- und Methodenunterstützung wahrnimmt, während die Unterabteilung IV 4 quantitative bzw. qualitative Analyseunterstützung bereitstellt. Diese Arbeit ist in Zusammenarbeit mit dem Referat IV 3 (2) Anwendungsunterstützung OR/ M&S und Architektur entstanden, welches vereinfacht im Weiteren als Planungsamt bezeichnet wird.

Abbildung 1.1: Planungsamt der Bundeswehr – Abteilung IV [1].

2 Grundlagen

Ein Modell, bzw. der Begriff der Modellbildung und Simulation (M&S) findet in den unterschiedlichsten Bereichen Anwendung, weshalb viele Definitionen existieren, welche abhängig von der jeweiligen Betrachtungsweise und des jeweiligen Anwendungsbereichs sind. In diesem Kapitel werden die grundlegenden Begriffe definiert, auf deren Verständnis die weitere Arbeit aufbaut. Die Informationen dazu stammen weitestgehend aus [2]. Weiterhin ist M&S in der Konzeption der Bundeswehr definiert als „[...] die auf Basis mathematisch formalisierter Modelle programmierte Softwareanwendung, die eine Simulation verschiedener Vorgänge erlaubt. Durch Simulationsläufe und Veränderung der Modellparameter lassen sich Erkenntnisse über das Modellverhalten gewinnen, die dann auf das reale System übertragen werden können." [3, S. 64]

Darüber hinaus wird auch der Begriff Data Farming erläutert. Diese Fähigkeit ist von besonderem Interesse des Planungsamts und wurde in einer Arbeitsgruppe der NATO unter Partizipation des Planungsamts näher untersucht.

2.1 Modellierung

Unter einem Modell versteht man eine vereinfachte Abbildung der Realität. Während die Formalisierung von Modellen im Allgemeinen keine Voraussetzung ist, ist dies für die Modellierung und späteren Durchführung von Computersimulationen zwingend notwendig. Die Modellierung befasst sich mit dem Erstellen eines passenden Modells. Während in einer Vielzahl von Anwendungsfällen die Systemanalyse - die durch Analyse, Dekomposition und Abstraktion komplexe Sachverhalte aufschlüsselt - bereits genügt, um die an das Modell gestellte Fragestellung zu beantworten, kann im Anschluss eine Computersimulation folgen. Ein passendes Modell bedeutet in diesem Zusammenhang, dass die Art der Modellierung, sowie die Auflösung festgelegt sind. Außerdem müssen die qualitativen und quantitativen Größen bekannt sein.

Die Auflösung bedeutet, wie detailliert das Modell ist: Wird ein Gefecht auf der Ebene der Einzelentität bis zu kleineren militärischen Verbänden (auch hochauflösend bezeichnet) oder auf der Ebene eines Zugs oder sogar auf Bataillonsebene (auch als aggregiertes Modell bezeichnet) modelliert. Wird eine zu hohe Auflösung gewählt, wird das Modell unflexibler und der Berechnungsaufwand für die Simulation steigt überproportional stark an. Im Gegensatz dazu wird eine Fragestellung durch eine zu niedrige Auflösung, bzw. zu starke Aggregation möglicherweise unzureichend beantwortet. Eine höhere Flexibilität in der Modellierung gewinnt man durch eine größere Anzahl an Freiheitsgraden, d.h. indem man ein Modell nicht spezifisch auf einen einzelnen Zweck überanpasst (*overfitting*), sondern möglichst Verallgemeinerungen und Annahmen tätigt, was wiederum in einem aggregiertem Modell einfacher möglich ist.

Eine qualitative Größe beschreibt die Struktur des Modells und ist damit ein Faktor zur Beschreibung des Aufbaus, also der Systemelemente und deren Zusammenhänge. Quantitative Größen hingegen beschreiben den Einfluss der qualitativen Größe auf das Gesamtmodell und die umliegende Umwelt an sich. Ebenso gilt es zu klären, wie sensitiv ein Modell sein darf. Die Sensitivität gibt an, wie stark die Lösung des Modells durch kleine Änderungen der Inputgrößen verändert wird. Diese gilt es besonders zu berücksichtigen, um den Einfluss von Fehlergrößen zu kontrollieren.

2.2 Simulation

Es gibt Realexperimente, die aufgrund ihrer Größe und Komplexität nicht durchgeführt werden können, zum Beispiel den Lebenszyklus einer Galaxie zu studieren oder das Wetter anhand von meteorologischen Daten vorauszusagen. Auch gibt es Experimente, die aus Kosten- oder ethischen Gründen nicht durchführbar sind. In solchen Fällen bietet es sich an, stattdessen Simulationen durchzuführen. Ziel einer Simulation ist es, neben dem Gewinn von Erkenntnissen über bekannte Szenarien, auch unbekannte Szenarien vorherzusagen. Es muss an dieser Stelle angemerkt werden, dass die Zielsetzung ein kritischer Parameter bei der Festlegung der Rahmenbedingungen einer Simulation ist und mitunter das Ergebnis stark beeinflussen kann. Deshalb muss der Zielsetzung einer Simulation besondere Aufmerksamkeit gewidmet werden.

International hat sich die Kategorisierung von Simulationen wie folgt durchgesetzt:

- Live Simulation: Eine Simulation, welche an realen Systemen mit realen Personen durchgeführt wird und Teile der Systeme simuliert werden.
- Virtuelle Simulation: Reale Personen bedienen virtuelle Systeme, zum Beispiel ein Flugsimulator.
- Analytische Simulation: Alle Elemente eines Modells werden simuliert, wobei der Ablauf der Simulation in festgelegten Grenzen frei variiert werden kann.

Neben diesen Kategorien existieren unterschiedliche Klassifizierungen, mit der Simulationen beschrieben werden können, deren Definitionen sich nach [4] richten:

Eine deterministische Simulation ist das Gegenstück zur stochastischen Simulation. Zufällige Ereignisse werden ausgeschlossen, weshalb sich diese Art der Simulation für Modelle ohne Ungewissheiten eignet. In stochastischen Simulationen werden Ungewissheiten als mathematisches Risiko modelliert. Diese werden in der Simulation mit Hilfe von Pseudozufallszahlengeneratoren und Wahrscheinlichkeitsverteilungen konkretisiert. In Simulationen, welche Wahrscheinlichkeiten beinhalten, werden oft mehrere Anläufe mit unterschiedlichen Startwerten (*Seeds*) für die Pseudozufallszahlengeneratoren durchgeführt, um den Einfluss der Zufallszahlen zu minimieren. Durch einen bekannten *Seed* ist es außerdem möglich, eine deterministische Simulation immer wieder gleich durchführen zu können. Deterministische Simulationen sind den stochastischen Simulationen, wenn möglich, immer vorzuziehen, da die Auswertung der Ergebnisse nachvollziehbarer

ist. Viele Phänomene - insbesondere in der Natur - lassen sich allerdings nicht deterministisch abbilden. Um dennoch ein deterministisches Modell aufzubauen, werden Eintrittswahrscheinlichkeiten häufig auf der Basis von Stichproben oder Expertenmeinungen ermittelt, womit dem deterministischen Modells oftmals ein stochastisches Verhalten unterbaut ist.

Darüber hinaus können Simulationen in geschlossene oder offene Simulationen unterschieden werden. In einer geschlossenen Simulation wird nach Eingabe aller Inputparameter nicht mehr eingegriffen. Offene Simulationen werden auch interaktive Simulationen genannt, da direkte Eingriffe eines Operateurs in die Simulation durchgeführt werden können. Offene Simulationen eigenen sich nur bedingt für analytische Zwecke, da sie vorrangig eine Schulung von Operateuren ermöglichen, Prozesse simulieren und durch die Unberechenbarkeit des menschlichen Verhaltens wertvolle Schlussfolgerungen aus der Simulation treffen, die als Unterstützung zur Entscheidungsfindung beitragen können. Eine besondere Form der interaktiven Simulation ist das *War Gaming*, welches neben computergestützten Simulation noch einen weitaus größeren Bereich abdeckt, welcher hier nicht näher thematisiert wird.

Außerdem wird zwischen diskreten und kontinuierlichen Simulationen unterschieden. Bei einer kontinuierlichen Simulation werden Zustandsänderungen durch Differenzialgleichungen modelliert. Dabei handelt es sich auch oft um eine deterministische Simulation. Im Gegensatz dazu werden in diskreten Simulationen zu diskreten Zeitpunkten Ereignisse abgerufen und bewertet. Die Zustandsänderung wird entweder in vordefinierten Zeitschritten (zeitschrittbasierte Simulation) oder durch extern erzeugte Ereignisse (ereignisgetriebene Simulation) hervorgerufen.

Eine Kombination aus beiden Methoden wird hybride Simulation genannt. Ein Beispiel hierfür ist die Simulation eines Abnutzungsgefechts. Dieses wird vorwiegend mit sogenannten Lanchester-Differentialgleichungen simuliert, während der Ablauf der restlichen Simulation ereignisgesteuert ist. Der Zeitverlauf einer Simulation kann in drei Stufen realisiert werden:

- In Echtzeit *(Real-Time)*,
- beschleunigt bzw. verlangsamt *(Scaled Real-Time)* oder
- so schnell, wie es die Simulation und die Hardware erlauben *(As-Fast-As-Possible)*.

Zuletzt ist es wichtig, zwischen quantitativen und qualitativen Simulationen zu unterscheiden. Dabei wird nicht die Art der Simulation betrachtet, sondern die Art der Ergebnisauswertung. Bei quantitativen Simulationen handelt es sich bei den Ergebnissen um Messgrößen, welche besonders relevant für analytische Simulationen sind. Dagegen liefern qualitative Simulationen Aussagen zu explorativen Fragestellungen - beispielsweise zum möglichen Verhalten des Feindes.

2.3 Data Farming

Der Begriff Data Farming wurde Ende der 1990er Jahren eingeführt. Ein aktueller Artikel zum Stand der Technik bezüglich Data Farming ist in [5] zu finden. Unter Data Farming ist ein Prozess zu verstehen, der auf Basis von Simulationsmodellen schnelle Antworten auf analytische Fragestellungen liefert. Der Aufbau eines Data Farming Prozesses ist in Abb. 2.1 dargestellt.

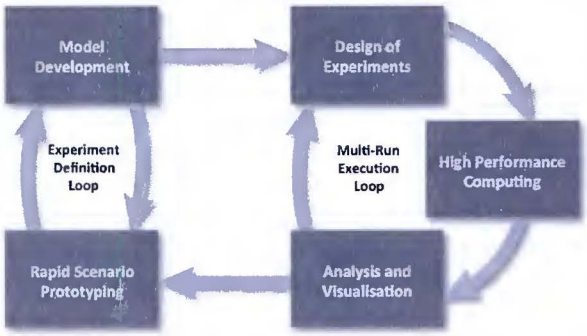

Abbildung 2.1: Data Farming Prozess [6, S. 2].

Tiefer gehende Details zu den fünf Bereichen des Data Farming Prozesses können in [6] nachgeschlagen werden.

Beim Data Farming handelt es sich um einen iterativen Ansatz, welcher sich grundsätzlich anhand eines Vorgehens innerhalb von zwei miteinander verbundenen Schleifen darstellen lässt. In der dort links dargestellten Schleife - der *Experiment Definition Loop* - wird ein Simulationsmodell in einer passenden Auflösung innerhalb eines Simulationssystems erstellt. Hierfür ist es primär und vorrangig von grundlegender Bedeutung, dass festgelegt wird, welche Inputdaten benötigt werden und welche Outputdaten zur Beantwortung der zugrundeliegenden Fragestellung generiert werden sollen. Um eine Abschätzung der Qualität dieser Festlegungen treffen zu können, muss ein passendes Szenario entwickelt werden, welches anschließend das Simulationsmodell in einem Simulationssystem ausführbar macht. Dieser Prozess verläuft im Data Farming iterativ, d.h. er wird so lange durchgeführt, bis das erstellte Simulationsmodell die ursprüngliche Fragestellung mit einer hinreichenden Genauigkeit und Güte beantworten kann. Mittels dieses sog. *Rapid Scenario Prototyping* lassen sich Szenarien schnell entwickeln und auch gegebenenfalls verwerfen. Dabei versteht sich Data Farming als ein auf Fragen basierter Ansatz. In diesem Zusammenhang taucht häufig die Begrifflichkeit *„What if?“* auf. Durch das wiederholte Fragen auf die durch die Methodik erhaltenen Ergebnisse, lassen sich eine Vielzahl von Fragestellungen beantworten. Gleichzeitig liefert diese auch Antworten auf Fragen, welche im Voraus nicht einmal beachtet wurden, denn Data Farming schafft - aufbauend auf den erfolgten Ergebnissen - eine Basis für weitere Fragestellungen. Auch lassen sich besonders auffällige und unerwartete Ergebnisse genauer untersuchen und gemeinsam mit Experten ermitteln, ob das Auftreten der Situation realistisch ist und

inwieweit daraus resultierende Folgen relevant für die eigenen Ziele sind. Daraus lassen sich dann gegebenenfalls Empfehlungen zur Entscheidungsunterstützung ableiten.

Zusammenfassend lässt sich Data Farming als Entscheidungsunterstützungsmethode einordnen, welche durch eine hohe Anzahl an Simulationsdurchläufen mit verschiedensten Parameterkombinationen ungeahnte Zusammenhänge aufdecken kann. In diesem Zusammenhang spricht man auch von der Untersuchung eines sogenannten Parameterraums. Obwohl es für analytische Zwecke bestimmt wünschenswert wäre, den gesamten Parameterraum zu untersuchen, ist dies auf Grund der hochempfindlichen Kombinatorik kaum möglich und nicht zielführend. Die Größe wird maßgeblich von der Anzahl der Variablen, sowie deren Definitionsbereiche bestimmt. Werden zum Beispiel 100 Sensoren untersucht, die jeweils ein oder ausgeschaltet werden können, dann gäbe es 2^{100} mögliche Kombinationen. Hinzu kommen noch zusätzliche Durchläufe, wenn die Simulationen mit unterschiedlichen Seeds gestartet werden sollen. Aus dem Grund muss im Voraus beim Experimentdesign (*design of experiments*) festgelegt werden, wie der Bereich des Parameterraums eingegrenzt wird, sodass das Data Farming Experiment in einem vertretbaren Zeitrahmen durchgeführt werden kann. In dieser Arbeit wird die Data Farming Fähigkeit von FLAMES geprüft und untersucht, wie ein solches Experiment technisch umgesetzt werden kann.

2.4 Verifizierung und Validierung

Ein wichtiger und kritischer Punkt im Zusammenhang mit M&S sind die Verifizierung und Validierung von Simulationsmodellen. Ziel ist es, die Verlässlichkeit der Ergebnisse sicherzustellen. Die Verifizierung trifft eine Aussage darüber, ob die technische Implementierung des Modells korrekt durchgeführt wurde. Die Verifizierung lässt sich auf Basis des Modellaufbaus durchführen. Beide Varianten sind äußerst kompliziert, wobei insbesondere die Verifizierung durch den Programmcode für heutige Zwecke leider nur unzureichend wissenschaftlich untersucht ist.

Die Validierung einer Computersimulation gibt an, ob das dargestellte Verhalten während der Simulation dem in der Realität zu erwartendem Verhalten in etwa entspricht. Weiterhin ist zwischen Datenvalidität (Validität der zugrundeliegenden Daten), Modellvalidität (Validität des internen Modellverhaltens) und operationaler Validität (Validität des Modellverhaltens in der Ausführung) zu unterscheiden [7].

3 FLAMES

Die Ternion Corporation wurde 1989 von drei Ingenieuren gegründet, welche zuvor an mehreren Simulationsprojekten mit der U.S. Regierung gearbeitet hatten. Gegründet wurde das Unternehmen mit dem Gedanken, dass die Qualität von Computersimulationen für militärische Zwecke gesteigert werden muss und deren Anschaffung nach dem alten Modell viel zu teuer sei. Stattdessen wurde das FLexible Analysis and Mission Effectiveness System (FLAMES) entwickelt, welches sich heute als Framework versteht und die Basis für eigene Simulationsmodelle bildet. Was genau ein Simulationsframework von einer klassischen Simulationssoftware unterscheidet, wird hierbei im folgenden Unterkapitel erläutert. Zur FLAMES Produktfamilie gehört heute eine Vielzahl von Produkten, welche in diesem Kapitel ebenfalls erläutert werden.

FLAMES ermöglicht das Erstellen von agentenbasierten Simulationsmodellen. Gemäß der Ternion Corporation sind mit dem Framework weitere unterschiedliche Anwendungen realisierbar. Abb. 3.1 listet einige davon auf. Am Ende dieses Kapitels werden öffentlich zugängliche Praxisbeispiele vorgestellt, um zumindest ein Bild von den tatsächlich in der Praxis umgesetzten Möglichkeiten zu ermöglichen.

Abbildung 3.1: FLAMES – Simulationstypen [8].

3.1 Simulationsframework

Die folgenden Informationen zu einem Simulationsframework stammen aus einer Veröffentlichung der Swedish Defence Research Agency (FOI) [9], welche sich mit der Beschaffung einer neuen Softwarelösung, speziell für die Forschungsrichtung *Information Fusion Research*, befasst.

Am Anfang einer Modellierung steht eine konkrete Fragestellung. Im Anschluss daran muss das Modell so weit formalisiert werden, damit dieses in eine Computersimulation überführt werden kann. Dazu wird heutzutage in den häufigsten Anwendungsfällen ein Softwareentwicklungsprojekt gestartet, um eine auf die Fragestellung zugeschnitte-

ne Simulationssoftware zu entwickeln und im Anschluss die Simulation durchführen zu können. Nun kann dieses Projekt entweder an ein externes Unternehmen weitergegeben werden, was einen erheblichen Kosten- und Koordinationsaufwand bedeutet, oder die eigenen Entwickler werden beauftragt. Im letzteren Fall bedeutet dies eine zeitliche Bindung der dafür qualifizierten Mitarbeiter, wobei die eigentliche Aufgabe - das Beantworten der Fragestellung - noch gar nicht bearbeitet werden kann. Erschwerend kommt hinzu, dass Softwareprojekte - egal ob extern oder intern - scheitern können, oder die Anforderungen nicht erfüllen. Dies ist umso dramatischer, wenn es nicht gelingt, das Simulationsmodell hinreichend zu verifizieren und zu validieren.

Ein Simulationsframework hingegen setzt sich aus einer Reihe von Produkten zusammen. Es ermöglicht dadurch eine Entwicklungsumgebung, welche eine Erstellung von Szenarien für M&S Vorhaben mit Hilfe von Werkzeugen und wiederverwendbaren bzw. anpassbaren Modellbausteinen - ähnlich einem Baukastenprinzip - deutlich beschleunigt. Außerdem kann die Qualität der Modellierung dadurch gesteigert werden, dass Erfahrungen - und eigene Entwicklungsbausteine - aus vorangegangenen M&S Projekten mit in die Entwicklung neuer Szenarien einfließen kann.

Allerdings fordert die Nutzung eines Simulationsframeworks auch intensive Ressourcen, welche zwingend berücksichtigt werden müssen. Wichtig ist in diesem Zusammenhang, dass eine Anschaffung nicht stattfinden sollte, um kurzfristige Vorteile zu erzielen. Erst langfristig stechen die Vorteile besonders heraus. Es kann durchaus dazu kommen, dass die Lizenzierungskosten - für alle benötigten Produkte einer befriedigenden Entwicklungsumgebung - die einmaligen Kosten eines Softwareprojekts überschreiten. Es ist ersichtlich, dass sich dieses Verhältnis erst mit zunehmender Nutzung zu Gunsten eines Frameworks ändert. Zusätzlich erfordert die Nutzung einer kommerziellen Software Expertise, welche zu Beginn teuer aufgebaut werden muss. Vorteilhaft ist hierbei allerdings, dass ein kommerzielles Produkt durch klare Schnittstellen und niedrigere Barrieren eine Aufteilung der Ressourcen durch die Zusammenarbeit mit anderen Institutionen ermöglicht. Diese Art der Zusammenarbeit muss nicht - wie bei einem klassischen gemeinsamen Entwicklungsprojekt - vertraglich festgehalten werden und ist dadurch deutlich flexibler. Um die oben genannten Vorteile nutzen zu können, müssen jedoch folgende Rahmenbedingungen erfüllt sein:

- Offenheit und Erweiterbarkeit – Dadurch ist es erst möglich, Modelle für einen breiten Anwendungsbereich zu entwickeln.
- Ausreichend dokumentiert – Ohne dieses ist es nicht möglich, eine fremd entwickelte Software in dem benötigten Umfang zu benutzen.
- Softwarewartung wird durch den Hersteller durchgeführt – Die Arbeiten dafür sind zeit- und kostenintensiv und sollten, wenn möglich, durch die Lizenzierungskosten abgedeckt worden sein.
- Modular aufgebaute Architektur – Diese ermöglicht eine Entwicklung von unabhängigen Plug-Ins, sowie eine Einbindung von *third party software*, was dem Kernvorteil eines Frameworks entspricht.

Es werden noch weitere Punkte erwähnt, wobei FLAMES all diese Punkte erfüllt. Es muss aber bedacht werden, dass der zitierte Artikel gemeinsam mit dem Präsidenten der Ternion Corporation verfasst wurde. Daher muss darauf geachtet werden, dass auch die Alternativen zu FLAMES zumindest die genannten Rahmenbedingungen erfüllen, um für eine engere Auswahl in Betracht gezogen zu werden.

3.2 FLAMES Produktfamilie

Wie bereits angedeutet wurde, handelt es sich bei FLAMES nicht um eine Simulationssoftware, sondern um ein Simulationsframework, welches zusammen mit weiteren Produkten das Erstellen eigener Simulationsszenarien ermöglicht. Hier wird zwischen der FLAMES *Runtime Suite* und der FLAMES *Development Suite* unterschieden, wobei beide Varianten jeweils um optionale Features erweiterbar sind. Neben diesen Optionen gibt es auch die Möglichkeit, das Produkt FAST zu erwerben, welches im Abschnitt 3.2.3 noch nähere Erläuterung findet. Die Funktionen und Inhalte der jeweiligen Produkte werden im Folgenden vorgestellt.

3.2.1 FLAMES Runtime Suite

Die FLAMES Runtime Suite stellt die Standardversion des Frameworks dar. Es kommt grundsätzlich mit den drei Programmen FORGE, FIRE und FLASH, deren Beziehung zueinander in Abb. 3.2 dargestellt wird.

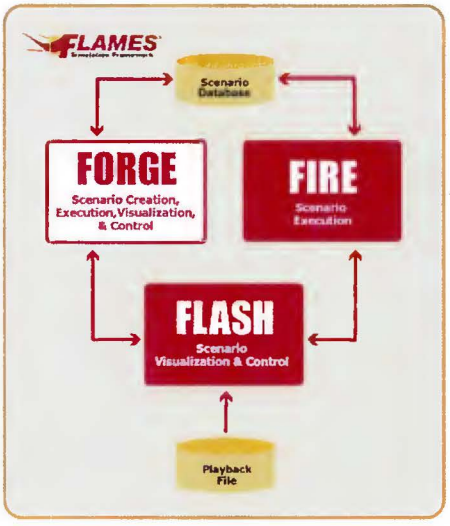

Abbildung 3.2: Aufbau der FLAMES Runtime Suite [10].

FORGE

Der FLAMES Operational Requirements Graphical Editor (FORGE) ist das Herzstück der drei Programme, wobei jedoch aufgeführt werden muss, dass alle drei Werkzeuge miteinander im Austausch stehen. Sofern das Szenario manuell aufgebaut, bearbeitet und visualisiert werden soll, wird FORGE verwendet. Dafür bietet FORGE ein Graphical User Interface (GUI), welche den dafür benötigten Programmieraufwand für den Operateur auf ein Minimum reduziert. Eine detaillierte Vorstellung von FORGE ist im Abschnitt 3.11 zu finden.

FIRE

Das FLAMES Interactive Runtime Executable (FIRE) ist ähnlich wie FORGE, wird allerdings im *batch mode* ausgeführt, also per Kommandozeilenbefehl gestartet. Das bedeutet, dass insbesondere die Funktionen, welche durch das GUI ermöglicht werden, nicht verfügbar sind. Allerdings bietet diese Art der Ausführung andere Möglichkeiten, um Simulationsstudien durchzuführen. Diese unterschiedlichen Funktionen sind in Tabelle 3.1 dargestellt.

Unterschied	**FIRE**	**FORGE**
Szenario mit einem einzigen Befehl ausführen (auch mehrfach)	X	
Szenario mit externer Experiment-Datei ausführen	X	
Checkpoint-Dateien erstellen und Szenario an diesen fortsetzen	X	
Grafische Benutzeroberfläche um Szenario zu kontrollieren		X
Szenario erstellen und bearbeiten und visualisieren		X

Tabelle 3.1: Unterschied FIRE - FORGE.

FLASH

Der FLAMES Scenario Highlighter (FLASH) wird benutzt, um eine bereits durchgeführte Simulation eines Szenarios erneut visuell abzuspielen. Dazu wird eine Playback-Datei benötigt, die während der Simulation erstellt wird, sofern man dies eingestellt hat. Sowohl FORGE als auch FIRE können eine Playback-Datei erstellen. Gleichzeitig kann FLASH eine laufende Simulation überwachen und visualisieren. Besonders interessant ist in diesem Zusammenhang die Möglichkeit, FLASH auf einem anderen Computer innerhalb des Netzwerks durchzuführen, obwohl dieser die Simulation nicht durchführt. Wurde die *Interactive Server Option* dazugekauft, kann FLASH während der Simulation in das Geschehen eingreifen und beispielsweise Kommandos an einzelne Einheiten geben.

OPTIONAL Features

Da FLAMES schon seit einigen Jahren verfügbar ist, haben sich über die Zeit mehrere offizielle Erweiterungen - sogenannte Features - entwickelt:

- 3D Option – Ermöglicht das Szenario, welches standardmäßig in 2D dargestellt wird, in einem 3D Modell darzustellen. Die Art der Darstellung ist abhängig von den zur Verfügung stehenden Daten.
- Analysis Option – Unterstützung für analytische Simulationen, sowie Auswertung und Visualisierung von Sensoren.
- Interactive Simulation Option – Ermöglicht das Eingreifen in laufende Simulationen, auch mithilfe externer Software durch den FLAMES Interactive Client bzw. FLAMES Interactive Server.
- High Level Architecture (HLA) / Distributed Interactive Simulation (DIS) Option – Dabei handelt es sich um Schnittstellen, welche den Austausch zwischen mehreren Programmen bzw. Computern während einer Simulation ermöglichen.
- Multithreaded Execution Option – Erlaubt *multithreading*, welches die Simulation beschleunigen kann.
- Checkpoint/Restart Option – Ermöglicht das erneute Starten eines Szenarios an einem vordefinierten Checkpoint, z.B. nach einem Absturz des Systems.

Nach der Beschreibung durch die Ternion Corporation sollte die *Development Suite*, welche im Anschluss beschrieben wird, die Entwicklung jeglicher Art von Zusatzkomponenten ermöglichen. Daraus lässt sich schließen, dass auch die *optional features* selbst programmiert werden können. Ob dieser Schritt sinnvoll ist, hängt auch von den Kosten ab, welche für die Lizenzierung anfallen.

3.2.2 FLAMES Development Suite

Die FLAMES *Development Suite* unterscheidet sich von der *Runtime Suite* in dem Punkt, dass alle mitgelieferten Komponenten - sei es FORGE, FIRE oder FLASH - verändert und angepasst werden können. Grundsätzlich, so die Dokumentation, sei alles möglich, sofern man sich an die vertraglich festgelegten Konditionen hält. Allerdings ist es beispielsweise erst mit der *Development Suite* möglich, neue Modelle für die einsetzbaren Simulationseinheiten zu entwickeln. Abschnitt 3.7 behandelt diese mitgelieferten Modelle im Detail und zeigt auf, wo die Grenzen ohne *Development Suite* liegen. Des Weiteren können die 2D/3D Ansichten von der Bedienung bis zur Darstellung angepasst werden. Ein weiteres Feature der *Development Suite* stellt die Möglichkeit dar, neue Datenaufnahmeschnittstellen (*data recorders*) zu implementieren. Dies ist besonders wichtig, da der auswählbare Standardparameterraum von FLAMES nicht ausreichen wird, um damit zum Beispiel die M&S-Aufgaben im Planungsamt zufriedenstellend zu bearbeiten.

3.2.3 FAST

Der FLAMES Automated Simulation Trainer (FAST) ist im Gegensatz zu FORGE, FIRE und FLASH nicht Teil der *Runtime Suite*. Stattdessen ist FAST eine allein lauffähige Software mit dem Ziel, Live, Virtual and Constructive (LVC) Simulationen schnell und effektiv zu ermöglichen. Eine Lizenz für FAST beinhaltet alle Komponenten der *Runtime Suite* und einige ausgewählte *Optional Features*. Gleichzeitig wird FORGE durch FAST als neues „Hautprogramm" ersetzt, wobei Funktionen von FORGE im Hintergrund abgerufen werden. FAST ist dabei nach Aussage der Ternion Corporation in der Lage, tausende Entitäten in Echtzeit auf einem einzigen Computer simulieren. Es zeichnet sich insbesondere dadurch aus, dass Szenarien automatisch aus einigen Standarddateiformaten des U.S. Militärs erstellt werden können. Dies sind zum Beispiel Air Tasking Orders (ATO), beziehungsweise Airspace Control Orders (ACO), sowie Dateien im US Message Text Format (USMTF). Zusätzlich ist die Kommunikationsschnittstelle Joint Range Extension Applications Protocol (JREAP) vorhanden, welche dem NATO Standard entspricht.

Im Gegensatz zu FORGE kann FAST nicht modifiziert werden. Dennoch lassen sich mit der *Development Suite* weitere Funktionen zu FAST hinzufügen. FAST ist ebenfalls eine FLAMES basierte Software.

3.3 Nutzer

Die Informationen zu den Nutzern der Software von Ternion stammen von der Website des Herstellers [11]. Dies sei gemäß der Ternion Corporation jedoch auch nur ein Ausschnitt des Kundenstamms. Es ist sehr auffällig, dass zumindest die Bezeichnungen der deutschen Kunden sehr alt sind. So finden sich im Bereich International Government das HQ AIRNORTH, Ramstein Air Base wieder. Dieses wurde aber schon seit 2004 bereits mehrfach umstrukturiert und ist heute als HQ AIRCOM Ramstein, Germany wiederzufinden [12]. Hier haben interne Untersuchungen ergeben, dass das HQ AIRCOM die in Zusammenarbeit mit Ternion entwickelte Software, auf Basis von FLAMES, heute noch verwendet. Als weiteren staatlichen Kunden in Deutschland wird das NATO Combined Air Operations Centres (CAOC) aufgeführt, welches zurzeit im CAOC Uedem aufgestellt ist. Im Bereich internationaler Unternehmen wird EADS Deutschland GmbH als deutscher Kunde angegeben. Hier ist zu vermuten, dass das heutige Airbus Defence and Space Kunde der Ternion Corporation war bzw. ist [13].

Die meisten Firmenkunden stammen aus den USA. Neben 26 aufgelisteten Kunden aus dem U.S. Government - durchweg militärische Einheiten - benutzen mindestens 30 US-amerikanische Unternehmen die Produkte der Ternion Corporation. Von der Liste der nicht-US-amerikanischen Kunden ist die Swedish Defence Research Agency (FOI) der interessanteste Nutzer. Der Hauptakteur in diesem Zusammenhang ist Johan Schubert, welcher nun schon seit fast 30 Jahren im Bereich *Decision Support* arbeitet und seit Beginn seiner akademischen Laufbahn an dem Royal Institute of Technology (KTH) in Stockholm tätig ist [14]. Nennenswert ist dessen Wirken innerhalb der *NATO RTO Task Group Data Farming in Support of NATO* (MSG-088) [15] und dessen momen-

tanes Mitwirken in der *NATO RTO Task Group Developing Actionable Data Farming Decision Support for NATO* (MSG-124) [16]. Mehr über seine Person und seinem wissenschaftlichen Arbeiten ist unter [14] zu finden. Im Folgenden wird zumindest sein Paper *Decision Support for Simulation-Based Operation Planning* näher betrachtet, da dieses eine konkrete Implementierung von FLAMES beinhaltet.

In jedem Fall wird eine Kontaktaufnahme zu den jeweiligen Nutzern empfohlen, um Erfahrungen im Einsatz mit der Software aufzugreifen. Dies kann neben den vielseitigen Möglichkeiten, welche die bereitgestellte Plattform FLAMES bietet, auch technische bzw. organisatorische Probleme bei der Zusammenarbeit mit der Ternion Corporation aufzeigen. Außerdem können aus gemeinsamen Projektarbeiten mit anderen FLAMES Nutzern Add-Ons, Bausteine für die kognitiven Modelle, sowie *Patterns* ausgetauscht werden. Auch das führt wieder zu eingesparten Kosten und verkürzter Entwicklungszeit, was zwar im Vorfeld schwer messbar ist, aber einen signifikanten Unterschied zu der Alternative Eigenentwicklung bedeuten sollte.

3.4 Aufbau der Programmoberfläche

Zum Zeitpunkt der Erarbeitung dieses Dokuments lag die FLAMES *Runtime Suite* in der Version 14.0.1 vor, weshalb beim Öffnen der FLAMES-Verknüpfung das Programm FORGE gestartet wird. Es ist wichtig, das Programm im Administratormodus zu starten, da ansonsten keine Daten abgespeichert werden können. Beim erstmaligen Starten muss außerdem ein eigener Nutzer angelegt werden. Für diesen Nutzer werden eigene Datenbanken geladen. Wenn keine eigenen Datenbanken vorhanden sind, ist es möglich, die Beispieldatenbanken zu öffnen und für den neu angelegten Nutzer als eigene Datenbank zu speichern. Somit ist auch garantiert, dass beim modifizieren der Daten die Beispieldatenbank nicht überschrieben werden.

Das vollständig geladene Programm ist in Abb. 3.3 dargestellt. In dem Fenster *„Forge Status“* wird aufgelistet, welche zusätzlichen Optionen geladen wurden. Wichtig ist, dass diese ohne gültige Lizenz nur in Verbindung mit den *Bundled Components* funktionieren. Dies ermöglicht lediglich ein Ausprobieren der Funktionen. Die *Bundled Components* sind ein - von der Ternion Corporation bereitgestellter - Baukasten aus Geländedaten, Einheiten und Beispielszenarios, um den Nutzer einen Teil der Möglichkeiten zu präsentiere. Gleichzeitig bezwecken die *Bundled Components*, dass Entwickler bei einer Implementierung von eigenen Bausteinen oder Klassen auf die Beispielmodule aufbauen können und dadurch eine schnellere Einarbeitung bei der Entwicklung ermöglicht wird.

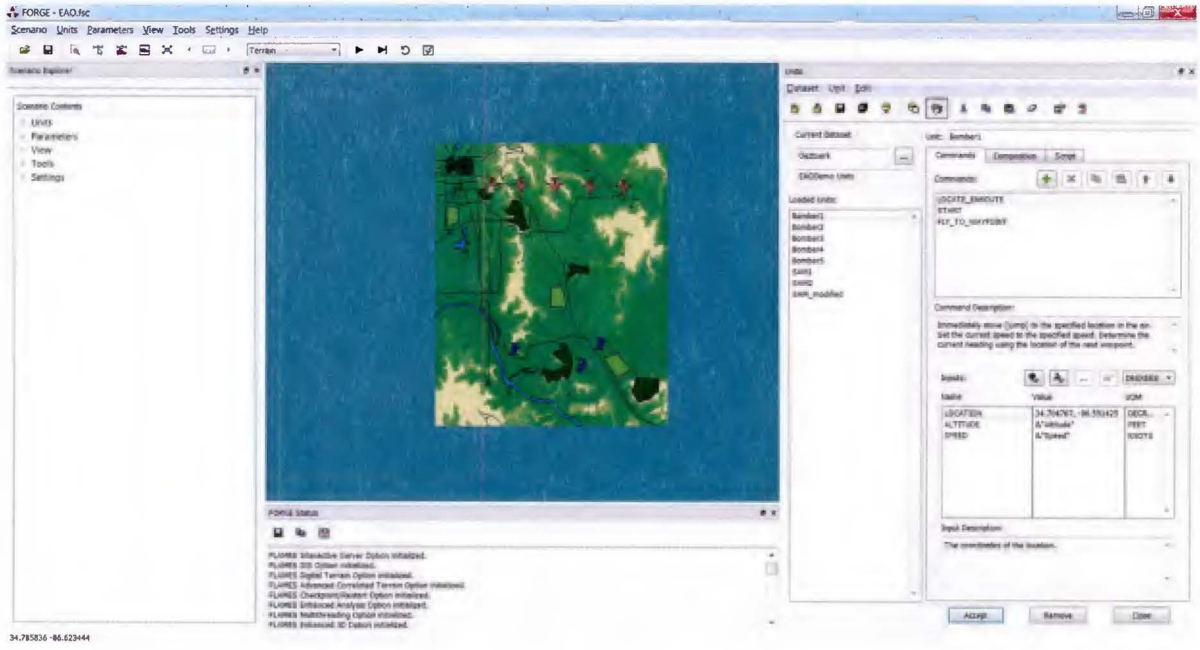

Abbildung 3.3: FORGE Benutzeroberfläche.

Die Fensteranordnung lässt sich frei gestalten. Außerdem sind neben dem *„Scenario Explorer“*, welcher alle Information zu dem Szenario in einer Baumstruktur gliedert, sowie dem *„Units“*-Fenster, einer Übersicht aller simulierten Einheiten in dem geladenen Szenario, noch weitere Ansichten vorhanden. Im Hintergrund der Fenster befindet sich eine 2D-Ansicht des Szenarios, in der beliebig weit hinein- oder herausgezoomt werden kann. Weitere wichtige Ansichten sind:

- Eine Übersicht der geladenen Datenbanken,
- Einstellungen zu den Szenariovariablen,
- Übersicht aller verfügbaren *Patterns*.

Zu dieser Ansicht lassen sich eine Vielzahl von Fenster hinzufügen, beispielsweise eine Übersicht aller *Units* in einem Szenario, Bearbeitungsfenster für das Szenario und die 2D-/ 3D-Ansichten, sowie eine Übersicht der geladenen Datenbanken. Oberhalb der 2D-Ansicht lässt sich in dem Auswahlfenster mit dem Inhalt *„Terrain“* das Gelände vollständig ausblenden. Rechts daneben befinden sich vier Bedienelemente, um den Simulationslauf zu konfigurieren und zu starten bzw. zu stoppen.

Mit Hilfe der *Development Suite* können viele dieser Fenster auch inhaltlich angepasst, sowie neue Bedienelemente entwickelt werden.

3.5 Aufbau eines Szenarios

Die ersten Schritte, um ein Szenario in FLAMES aufzubauen, sind festgelegt und können mit entsprechender Leichtigkeit bewältigt werden. Hierzu sind durch das Programm verschiedene Datensätze zu landen, welche zusätzlich aufeinander abgestimmt sein müssen. Werden nicht kompatible Datensätze geladen, führt dies unmittelbar zu Fehlern. Deswegen stellt sich hier bereits die Frage, weshalb der Importvorgang nicht ursprünglich vereinfacht wurde. Jedes Szenario benötigt Geländedaten und Informationen zur Atmosphäre, alles Weitere ist optional. Eine Übersicht der geladenen Datensätze bietet die Funktion „Scenario → Manage Datasets ...". Gleichzeitig können hier auch alle verfügbaren Datensätze für den jeweiligen Benutzer eingesehen und bei Bedarf fehlende Datensätze zusätzlich geladen werden. Sofern bereits ein Datensatz zu den Entitäten in dem Szenario erstellt wurde, kann dieses ebenfalls geladen werden.

3.6 Geländedaten

Die Umgebung in FLAMES setzt sich aus mehreren Datensätzen zusammen. Zuerst wird ein Datensatz *terrain data* geladen, welcher den Ort des Szenarios enthält und die dazugehörigen Geländedaten. FLAMES unterstützt die beiden Formate Digital Terrain Elevation Data (DTED) und FLAMES Advanced Correlated Terrain (FACT). Bei FACT handelt es sich um ein FLAMES internes Format.

DTED

DTED wurde von der National Geospatial-Intelligence Agency (NGA) entwickelt und wird auch hauptsächlich von dieser eingesetzt. Es handelt sich um ein Dateiformat, um Karten und Geoinformationen nach militärischen Standards zu speichern. Dies kann in drei Auflösungsstufen von DTED0 bis DTED2 erfolgen. Die genaue Spezifikation von DTED *MIL-PRF-89020B* ist in [17] zu finden. DTED-Daten werden mit dem *Digital Terrain Importer* geladen. Stichprobenartige Versuche zum Laden von zufälligen DTED-Daten - von der Website der NGA - sind hierbei jedoch fehlgeschlagen.

FACT

FLAMES Advanced Correlated Terrain (FACT) erlaubt eine mathematische Beschreibung - mittels Geometrie- und Vektordaten - des Geländes (Erdoberfläche, Unterwasser) und den dazugehörigen Metadaten (*Features*), also beispielsweise Straßen, Seen und Gebäude. Diese Umgebungsdaten können im Anschluss mit in die Modellierung integriert werden, um das Verhalten von Sensoren und Truppenteilen an die Geländegegebenheiten anzupassen. Zusätzlich können mit diesen Daten 2D- und 3D-Darstellungen des Geländes realisiert werden. Der FACT Importer ist Bestandteil der *3D Option* und ist in der vorliegenden Version fehlerhafterweise nicht verfügbar. Daher konnte dieser nicht getestet werden. Der vorliegenden Dokumentation folgend lassen sich mithilfe von FACT Vektordaten (*Shapefiles*) laden. Allerdings müssen diese den *„FACT Importer Requirements"*

entsprechen, welche gemäß der Ternion Corporation bisher nur durch das Programm *Terra Vista* von dem Softwarehersteller Presagis erfüllt werden.

Interessant zeigt sich - in diesem Zusammenhang - die Auswahl von *Terra Vista*. In dem bereits vorgestelltem Paper in Abschnitt 3.1 war die Modellierung des Geländes durch FLAMES unzureichend für die FOI . Als Einziger von den untersuchten Anbietern hatte die Ternion Corporation angeboten, das Geländemodell den Anforderungen der FOI entsprechend weiterzuentwickeln. Daraufhin wurde die Software *Terra Vista* als Ergebnis der Untersuchung ausgewählt. Die Ternion Corporation ist also zur Generierung der notwendigen Geländedaten auf eine externe Software umgestiegen, dessen *Shapefileoutput* nahtlos in FLAMES integriert wird. Die äußerst komplexe Geländedatengenerierung auszulagern hat den Vorteil, dass diese von einem Spezialisten ständig weiterentwickelt wird. Außerdem ist dieser im Notfall durch einen anderen Spezialisten austauschbar. Nachteilig ist eindeutig, dass die Software separat beschafft werden muss, was bei der Kostenrechnung mit einkalkuliert werden muss. Das grobe Importverfahren ist hierzu in Abb. 3.4 dargestellt.

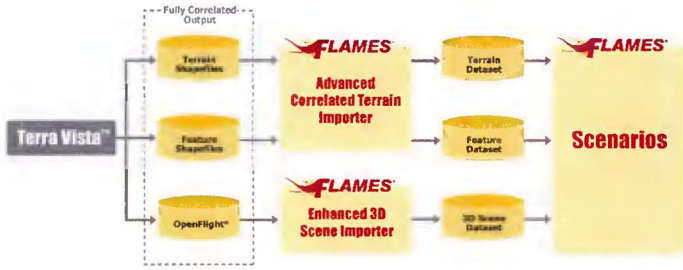

Abbildung 3.4: Importieren von Geländedaten [18].

Für die Bundeswehr ist der Geoinformationsdienst der Bundeswehr (GeoInfoDBw) für die umfassende Versorgung von Daten und Information über Geländedaten und Umwelteinflüssen zuständig. Es ist zu prüfen, ob der GeoInfoDBw passende Daten bereitstellen kann, um das Geländemodell in demselben Umfang - wie die Software *Terra Vista* - zu integrieren. Nachdem das *Terrain Dataset* geladen wurde, muss ein *Earth Model Dataset* geladen werden, welches bei einem FACT Import mit dem *Terrain Dataset* erstellt wurde. Der Datensatz enthält Parameter zur Form der Erde (sphärisch oder elliptisch), welche auch manuell anpassbar sind. Danach ist ein *Terrain Display Dataset* zu laden, welches dem Gelände die jeweilige passende Farbe - in Abhängigkeit von den Höhendaten - zuteilt. Zum Schluss kann ein *Feature Dataset* geladen werden, welches die in *Terra Vista* erstellten Features, also Straßen, Seen und Gebäude, in das Szenario einfügt.

Abschließend können noch simulierte Einheiten (*Units*) eingefügt und diesen Befehle zugeordnet werden. Im nächsten Abschnitt wird dieser Vorgang erläutert und abschließend in einem Beispielszenario dargestellt. An dieser Stelle muss noch angemerkt werden, dass zwar - abhängig vom Typ des Datensatzes - neue Datensätze über die GUI erstellt und abgespeichert, jedoch nicht mit Inhalt gefüllt werden können.

3.7 Aufbau einer Einheit

Das FLAMES Framework ist modular aufgebaut. Alle Akteure in einem FLAMES-Szenario werden als Einheit (*Unit*) bezeichnet. Diese ist ohne die zugehörigen Bausteine, siehe Abb. 3.5 , eine funktionslose Einheit. Eine Einheit kann zum Beispiel ein Bataillon, ein Fahrzeug oder auch ein einzelnes Geschoss sein. Dies ist völlig davon abhängig, wie dieses gestaltet wird.

Die Gestaltung bestimmt unter anderem, wie sich eine Einheit bewegt, kommuniziert und mit der Umgebung (und andere Einheiten) interagiert.

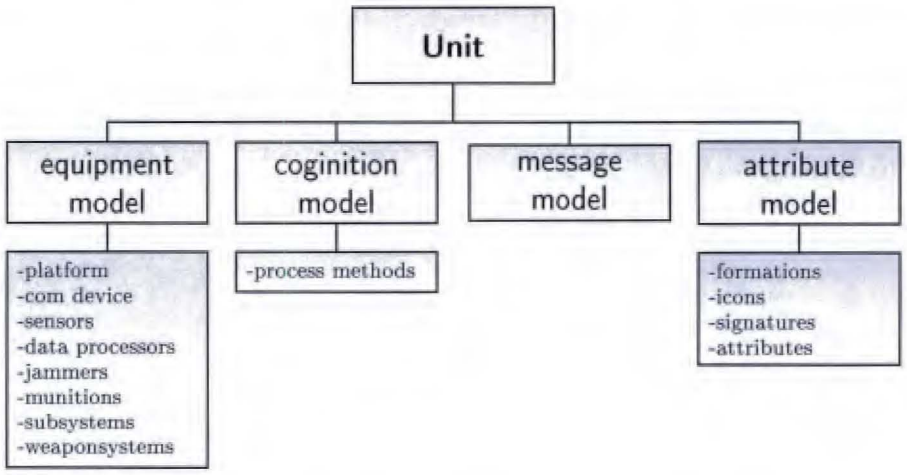

Abbildung 3.5: Grundlegender Aufbau einer Unit.

Eine *Unit* wird durch ein Skript erstellt. Allerdings ist es möglich (und auch empfohlen), dass diese interaktiv erstellt werden. Dies geschieht mit Hilfe von Templates (*Patterns*). Auf diese kann immer wieder zurückgegriffen werden, womit das Erstellen neuer Einheiten schnell und fehlerfrei (sofern das *Pattern* ebenfalls fehlerfrei ist) durchführbar ist. Sobald ein Entwickler ein *Pattern* implementiert hat, kann der Operateur - ohne weitere Programmierkenntnisse - die *Patterns* nutzen, um ein Szenario aufzubauen. Dadurch wird nicht nur die notwendige Modellierungszeit, sondern auch die Einarbeitungszeit für einen neuen Operateur in die Software reduziert.

Die konkrete Umsetzung einer *Unit* in einem *Pattern* wird nun folgend beispielhaft an einem F-15 Jet gezeigt. Dieser ist Teil der FLAMES *Bundled Components* und wird bei der Standardausführung mitgeliefert. Allerdings betont hierzu die Ternion Corporation deutlich, dass es sich um inoffizielle Beispiele handelt. Hier ist zu vermuten, dass sich die Ternion Corporation damit gegen Fehlfunktionen bzw. falschen Inhalten rechtlich absichert.

Pattern

Für den F-15 Jet wurde bereits ein *Pattern* erstellt, welches unter der Bezeichnung `F-15 Sweep Leader (Active Missile)` abgespeichert wurde. Das zugehörige Skript ist in Abb. 3.6 dargestellt.

```
1 INCLUDE "F-15 Sweep (Active Missile)";
2 INCLUDE "Aerial Warfare (Behavior)";
3
4 INITIATE "USE_FORMATION"  INPUT  ("FORMATION" = "Air,
      Echelon Left");
5
6 INITIATE "ASSIGN_ROLE"
7 INPUT (
8 "ROLE" = "LEADER");
9
10 INITIATE "LOCATE_ENROUTE"  INPUT  ("LOCATION" =
      TO_COORDINATE (?{$LOC}),
11 "ALTITUDE" = 10000.0 FEET, "SPEED" = 300.0 KNOTS);
12
13 START 0;
14
15 INITIATE ASSESS;
16
17 INITIATE "FLY_TO_WAYPOINT"  INPUT  (
18 "WAYPOINT" = TO_COORDINATE (?{FIRST WAYPOINT}),
19 "ALTITUDE" = 10000.0 FEET, "SPEED" = 300.0 KNOTS);
```

Abbildung 3.6: `F-15 Sweep Leader (Active Missile)` – Pattern.

Zu Beginn des Skripts wird die Bestückung der Plattform geladen. Bei einigen *Units* müssen dabei dieselben Standardkomponenten geladen werden. Hierfür bietet FLAMES die Möglichkeit, sogenannte *Dictionary Entries*, d.h. eine Zuordnung von Sensoren und Effektoren, für ein Modell zu definieren. Im Fall des F-15 Jet wurde ein solcher definiert, wie Abb. 3.7 zeigt. In dem vorliegenden Beispiel wurden gleich zwei *Dictionary Entires* mittels des `INCLUDE` Befehl in das *Pattern* geladen.

```
1 /* F-15 fighter  */
2
3 EMPLOY PLATFORM "F-15E";
4 EMPLOY SENSOR "Blue Aircraft Radar";
5 EMPLOY SENSOR "Blue Radar Warning Receiver";
6 EMPLOY SENSOR "Blue Sight";
7 EMPLOY WEAPONSYSTEM "Blue AAM Weapon System";
8 EMPLOY COMDEVICE "Blue RadioNtoN" ALIAS "PEER"
9   PARAMETER ("NETWORK" = 2);
10 ADD MUNITION "AIM-9B" QUANTITY 6;
```

Abbildung 3.7: `F-15 Sweep (Active Missile)` – Dictionary Entry.

Dictionary Entries müssen lediglich die Syntax der in FLAMES benutzten Skriptsprache einhalten. Sie tragen positiv zur Übersichtlichkeit bei und erlauben eine schnelle Zu-

sammenstellung neuer Entitäten aus vorhandenen Bausteinen. Die Inhalte des Skripts werden im Folgenden jeweils zeilenweise erklärt.

Plattform

Innerhalb des *Platforms*-Dialog ist zuerst eine Plattform zu laden, wobei jede Einheit genau auf einer Plattform aufbauen kann. Diese ist einem Leistungsblatt ähnlich. Im dargestellten Fall wird durch den Befehl `EMPLOY PLATFORM "F15-E";` die Plattform `F-15E` aufgerufen, deren Inhalt in Abb. 3.8 dargestellt wird.

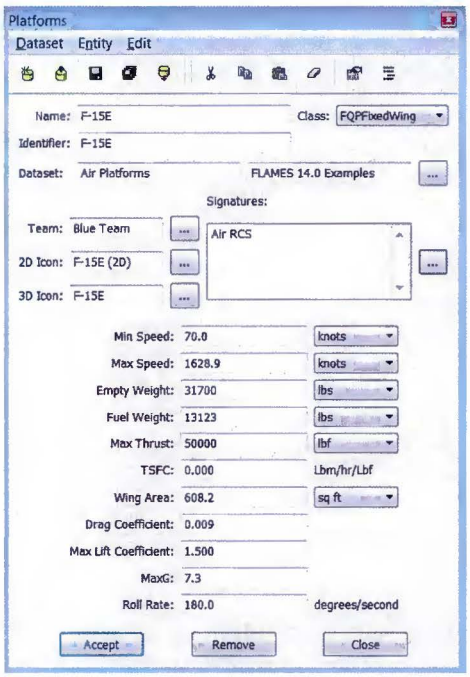

Abbildung 3.8: `F-15E` – Plattform.

Jede Plattform ist hierbei mit einem Rufzeichen (*Name*) und einer Identifikationsbezeichnung (*Identifier*) zu versehen. Zusätzlich muss ein Team, eine Sensorsignatur (*Signature*), sowie ein 2D/3D Icon angegeben werden. In den *Bundled Components* sind die Teams Rot (*Red Team*), Blau (*Blue Team*) und Zivilist (*Civilian*), sowie einige Icons bereits enthalten. Für den `F-15 Sweep (Active Missile)` hat die Plattform zusätzliche Angaben, welche durch die Klasse `FQPFixedWing` definiert werden. Es lassen sich für Luftfahrzeuge typische Parameter einstellen. Ändert man diese im Plattform-Dialog, dann sind alle Einheiten des Szenarios davon betroffen, welche auf diese Plattform zugreifen. Für Änderungen der F-15, welche nur ein bestimmtes Team betreffen sollen, muss eine zusätzliche Plattform definiert werden. Zwar ist eine Definition einer weiteren Plattform durch direktes Kopieren möglich, allerdings kann ohne die FLAMES *Development Suite* weder eine neue Klasse erstellt, noch eine vorhandene Klasse verändert werden.

Sensoren

Nachdem die Plattform geladen wurde, ist diese mit einem oder mehreren Sensoren zu versehen. Im vorliegenden Beispiel werden hierzu die Sensoren `Blue Aircraft Radar` (aktives Luftradar), `Blue Radar Warning Receiver` (passives Luftradar) und zusätzlich `Blue Sight` (Sichtlinie) geladen. Diese sind technisch identisch zu den Sensoren `Red Aircraft Radar` etc., allerdings benötigt das rote Team anderes Equipment, da der Funkverkehr und die Freund-/ Feinderkennung nur so korrekt zugeordnet und damit fehlerfrei funktionieren kann. Abb. 3.9 zeigt die unterschiedlichen Einstellungen, welche an einem Sensor, hier beispielhaft anhand der Klasse `FQSACRadar` demonstriert, vorgenommen werden können.

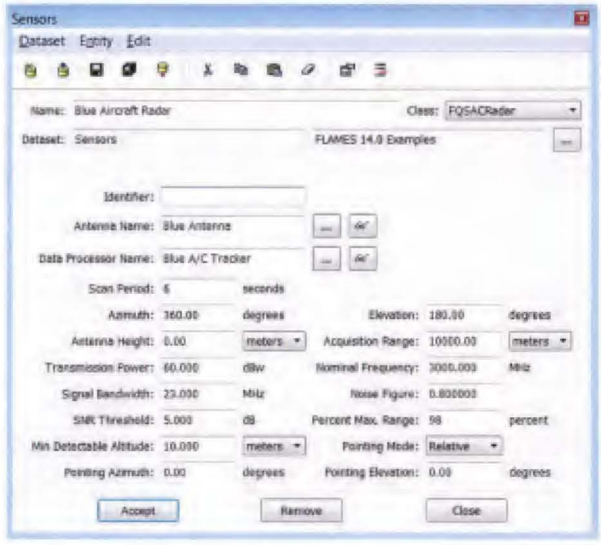

Abbildung 3.9: `Blue Aircraft Radar` – Einstellungen.

Waffensystem und Munition

Anschließend ist die Plattform mit einem Effektor zu versehen, hier anhand des `Blue AAM Weapon System` demonstriert. Dieser wird mit einem entsprechenden Meta-Munitionstyp versehen, hier einer Luft-Luft-Rakete der Klasse `FQWAAMissle`, an der keine sichtbaren Einstellungen vorgenommen werden können. Dies bildet eine Schnittstelle zwischen den Prozessmethoden und der Munition. Hierfür wurde der tatsächliche Munitionstyp anhand einer konkreten Luft-Luft-Rakete - der `AIM-9B` - spezifiziert und separat der Einheit zugeordnet. Der Munitionstyp gehört damit neben dem konkreten Effektor ebenfalls zum Equipment. Der Munitionstyp `AIM-9B` ruft zusätzlich die Klasse `FQMSemiActiveAAM` auf, welche mittels Bewegungsgleichungen die Flugbahn und das Verhalten der Munition steuert. Außerdem lassen sich weitere umfangreiche Einstellungen vornehmen, welche in Abb. 3.10 dargestellt sind.

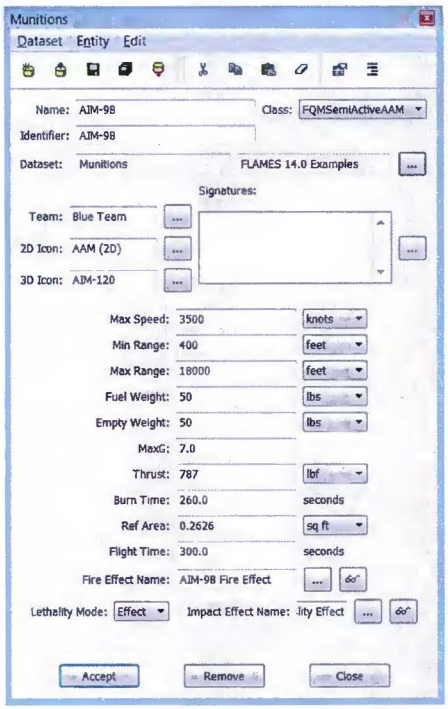

Abbildung 3.10: `AIM-9B` – Munition.

Kommunikationsmodul

Abschließend wird der Entität noch das Kommunikationsmodul `Blue RadioNtoN` hinzugefügt. Dieses wird im Abschnitt 3.9 näher erläutert. Der Name des Moduls deutet dabei auf den konkreten Zweck dieses Moduls hin: Der Implementierung einer Kommunikationsmöglichkeit aller mit dem Modul ausgestatteten Einheiten im Team Blau. Für die jeweilige Einheit werden ein Kommunikationsnetzwerk und eine Kanalnummer zugewiesen. Dadurch lassen sich mehrere Kommunikationsebenen aufbauen.

3.8 Einfügen der Unit in das Szenario

Nachdem das *Pattern* für die Einheit erstellt wurde, kann diese einfach und schnell in das Szenario eingefügt werden. Dies kann direkt per Mausklick an eine bestimmte Position in der 2D-Kartenansicht geschehen. Alternativ lassen sich die Koordinaten auch im Anschluss eingeben bzw. korrigieren. In dem folgenden Beispiel wurde eine Einheit mit dem *Pattern* `F-15 Sweep Leader (Active Missile)` erstellt und als `FIGHTER1-1` bezeichnet. Es ist ferner möglich, die Eigenschaften der erstellten Einheiten nachzuschlagen und eingeschränkt zu bearbeiten. Wird die zuvor erstellte Unit mit dem Namen `FIGHTER1-1` betrachtet, wird unter dem Reiter *Composition* die Zusammenstellung der einzelnen Module aufgelistet, welcher der jeweiligen Einheit zugeordnet wurden. Diese Auflistung der Module im Falle des Beispiels zeigt Abb. 3.11 .

Abbildung 3.11: `FIGHTER1-1` – Composition.

Hierzu ist bemerkenswert, dass hierbei auch Module auftauchen, welche beim vorherigen Erstellen des *Patterns* nicht manuell hinzugefügt wurden. Dies hängt vermutlich damit zusammen, dass innerhalb der Klassendefinitionen zusätzlich erforderliche Module aufgerufen werden. Hinzugekommen sind an dieser Stelle unter anderem drei kognitive Modelle, auf die aber in einem separaten Abschnitt detailliert eingegangen wird. Von der dargestellten Ansicht aus ist es möglich auf die einzelnen Module zuzugreifen und diese anzupassen. Allerdings muss auch hier angemerkt werden, dass eine Veränderung sich auf alle *Units* auswirkt, welche das jeweilige Modul integriert haben.

Unter dem Reiter *„Commands“* ist es nun möglich, die Einheit zu steuern. Der Aufbau ist in Abb. 3.12 dargestellt.

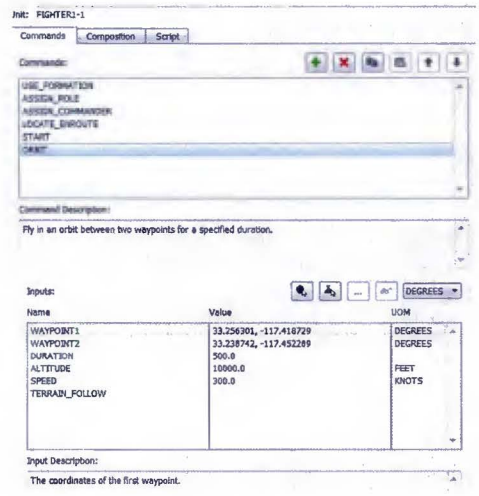

Abbildung 3.12: `FIGHTER1-1` – Commands.

Durch das interaktive Erstellen der *Unit* mit Hilfe eines *Patterns*, sind bereits einige Befehle vorgegeben. Diese werden beim Initialisieren der *Unit* erstellt. Befehle können erstellt, editiert und gelöscht werden. Zu jedem Befehl kann es außerdem Inputparameter geben. Dies können beispielsweise Koordinaten, Geschwindigkeit, Zeitdauer etc. sein. Der Reiter *„Script"* zeigt das dazugehörige High-Level Skript, welches die jeweilige *Unit* in der gewünschten Konfiguration in das Szenario integriert. Dargestellt ist dieses in Abb. 3.13 . Hier finden sich viele Einträge wieder, die in dem *Pattern* bereits beschrieben wurden. Obwohl die *Unit* - aus technischer Sicht - von einem Skript erstellt wird, ist die dargestellte Einheit lediglich durch Mausklicks und ohne Code erzeugt worden.

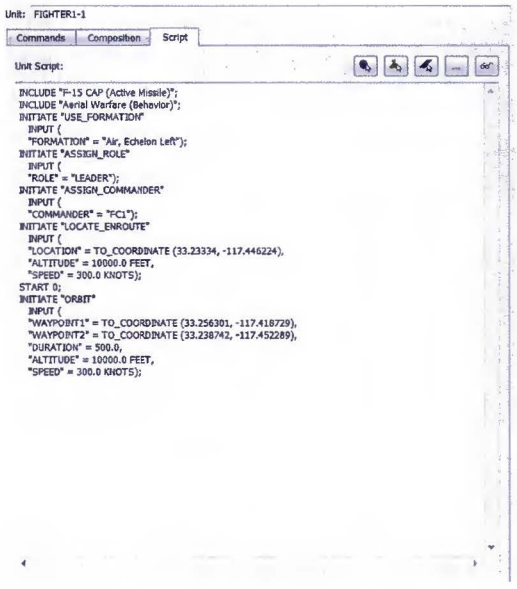

Abbildung 3.13: `FIGHTER1-1` – Script.

3.9 Kommunikation

Die Kommunikation in FLAMES soll realitätsnah und nachvollziehbar simuliert werden. Das bedeutet, dass *Units* in einem Szenario nur Zugang zu den Informationen haben, die auch für sie bestimmt sind oder welche selbst von der *Unit*, zum Beispiel durch Sensoren, aufgenommen und verarbeitet wurden.

Dafür existieren in FLAMES sogenannte Nachrichtenmodelle (*message models*). Diese Modelle beinhalten lediglich Daten und kein eigenes Verhalten und sollen reale Nachrichten simulieren. *Message models* werden während eines Szenarios dynamisch erstellt und nach der Übermittlung an eine oder mehrere Einheiten wieder zerstört. Einzelne Einheiten sollen nur über diesen Weg kommunizieren können, da ein Zugriff auf die Randbedingungen eines Szenarios, bzw. die kompletten Informationen, welche im Kernel verarbeitet werden, unrealistisch sind. Der FLAMES Kernel ist das metaphorische Rechenzentrum der Software. Es speichert und verarbeitet alle Datenstrukturen der aktuellen Simulation.

Eine Kommunikation zwischen zwei Kommunikationsmodulen (*ComDevices*) kann dabei wie folgt ablaufen:

- Eine Nachricht wird erstellt und attribuiert (Datenelemente). Dies geschieht fast immer durch einer *process method* eines *cognition models.*
- Die Nachricht wird einem oder mehreren *ComDevices* von der Einheit übergeben.
- Abhängig von der Spezifikation des *ComDevices* wird die Nachricht sofort oder zu einem späteren Zeitpunkt an ein *ComDevice* einer anderen Einheit übermittelt.
- Das Empfängermodul leitet diese Nachricht weiter an FLAMES, welches überprüft ob die Empfängereinheit die Nachricht verarbeiten kann.
- Ist diese in der Lage dazu, wird die Nachricht von FLAMES an die passende *message processing method* weitergeleitet und verarbeitet.
- Diese sind spezielle *cognition process methods*, wobei jede *messaging process method* einen speziellen Nachrichtentyp verarbeiten kann.
- Nach der Verarbeitung wird die Nachricht von FLAMES gelöscht.

In dem Beispielset von FLAMES sind folgende Kommunikationsmodule (ComDevices) vorhanden:

- Blue Radio1to1,
- Blue Radio1toN,
- Blue RadioNtoN.

Diese Kommunikationsmodule verknüpfen Einheiten von Team Blau untereinander. In entsprechender Ausprägung sind diese auch für das Team Rot vorhanden. Jede dieser dargestellten Kommunikationsmodule ruft die Klasse `FQCSimpleRadio` auf. Diese Klasse ermöglicht drei GUI-Einstellungen:

- Network Type (1to1, 1toN, NtoN)
- Network (INTEGER)
- Channel (INTEGER)

Die Klasse `FQSimpleRadio` basiert auf der Klasse `FComDevice`. Diese wiederum hat mehrere wichtige Standardmethoden zum Umgang mit Nachrichten (Verbindungsaufbau/ und -überprüfung, Nachrichten empfangen und versenden, ...) definiert.

Die *ComDevice* Klassen können mit speziellen Methoden von einigen *equipment models* interagieren. Dabei kann es sich um Jammer oder um spezielle Sensoren zum Erfassen von elektromagnetischen Signalen handeln.

3.10 Simulation von menschlichem Verhalten

In FLAMES wird das Verhalten durch kognitive Modelle (*cognition models*) simuliert. Bevor man sich für diese Richtung entschied, wurden Implementierungen von Expertensystemen und regelbasierten Verfahren getestet, welche keine zufriedenstellenden Ergebnisse produzierten. Das *cognition model* wird dagegen besonders hervorgehoben und als das Kernstück der Simulation bezeichnet. Das *cognition model* stellt das Gehirn einer *Unit* da und soll *human cognitive behaviour* simulieren. Im Folgenden wird der grobe Aufbau dieser Modelle anhand eines Beispiels aufgeschlüsselt, wobei dies ohne *Development Suite* nur in einem begrenzten Umfang möglich ist.

Die Entscheidung einer einzelnen Einheit ist abhängig von der Situation, in der sich diese Einheit gerade befindet. Diese wird gebildet von den eigenen Fähigkeiten und von externen Einflüssen. Eigene Fähigkeiten und (ursprüngliche) Absichten werden bereits vor dem Szenario definiert. Externe Einflüsse, wie zum Beispiel Umwelteinflüsse, aufgenommen durch die Einheit selbst (Sichtaufklärung), oder durch die verfügbaren Sensoren der eigenen Ausrüstung, beeinflussen ebenfalls das eigene Lagebild. Zusätzlich dazu erhält eine Einheit über Kommunikationsmodule Informationen oder Anweisungen von einer anderen Einheit. Die gesamten Informationen werden von den *cognition models* aufgenommen und verarbeitet, was am Ende des Prozesses zu der Entscheidung für ein bestimmtes Verhalten führt.

3.10.1 Sensoren

Um andere Einheiten im einen Szenario aufzuklären, sammeln Sensorenmodelle in unterschiedlichen Verfahren Informationen über umliegende Einheiten. Zum einen gibt es die `FSensor Detect` Methode, welche in festen Zeitintervallen über alle Einheiten in einem Szenario iteriert und überprüft, ob die jeweiligen Einheiten vom Sensor erkannt werden.

Zum anderen gibt es die `FSensor DetectEvent` Methode, welche ausgehend von einem Ereignis, welches von einem anderen Modell erstellt wurde, aktiviert wird.

Ein Sensor kann eine Einheit nur durch die verknüpften Modelle identifizieren. Dies können entweder *equipment models* oder *attribute models* sein. Einige Sensoren können eine Einheit nur identifizieren, wenn diese ein passendes *signature model* verknüpft hat. In den meisten Fällen sind die Beispieleinheiten ohne *signature model* nicht ermittelbar.

Zusätzlich ist es mittels der FLAMES Sensor Coverage Option möglich, die Reichweite und die Fähigkeiten von Sensoren sichtbar zu machen, um bei der Positionierung von Sensoren zu unterstützen.

3.10.2 Nachrichten

Im vorherigen Abschnitt zur Kommunikation wurden *message models* bereits erläutert. Nachrichten in Form von simulierten Sprach- oder Datennachrichten werden von FLAMES verarbeitet und an eine passende *message processing method* weitergeleitet, welche eine spezielle Methode der kognitiven Modelle ist.

3.10.3 Kognitive Modell

Das kognitive Modell soll realistisches menschliches Verhalten simulieren. In FLAMES wird dies durch das Aufrufen besonderer Methoden realisiert. Diese gehören dem jeweiligen kognitiven Modell an und sind, im Gegensatz zu den normalen Methoden - die beispielsweise bei *equipment models* aufgerufen werden - spezielle Prozessmethoden (*process methods*). Diese kontrollieren alle anderen Modelle, welche mit der Einheit verknüpft sind. Es handelt sich um individuelle Routinen, welche eine einfache und relativ simple Aufgabe durchführen, welche ein Mensch in dieser Art ebenfalls durchführen würde. Eine Verkettung dieser einfachen Aufgaben führt dann im günstigsten Fall zu einem Abbild komplexen menschlichen Verhaltens.

FLAMES kontrolliert die Ausführung der Prozessmethoden in einer ereignisbasierten Umgebung. Dies soll folgendes ermöglichen:

- Simultane Verarbeitungsprozesse,
- situationsabhängige Modellausführung,
- inputabhängige Entscheidungen, teilweise verknüpft an Bedingungen und
- Integration von Entscheidungen und Information von echten menschlichen Bedienern in einer interaktiven Simulation.

Die Dokumentation von FLAMES behauptet, dass diese Konstellation ausreicht, um alle denkbaren menschlichen Entscheidungsfindungsprozesse simulieren zu können, ohne eine zu komplexe Struktur zu erfordern.

3.10.4 Prozessmethoden

Um das Verhalten von Prozessmethoden zu verstehen, wird der Ablauf kurz skizziert. Sobald eine *Unit* eine Handlungsanweisung (nicht von einem Bediener, sondern von der Software) erhält, prüft FLAMES, ob die *Unit* diesen Befehl verarbeiten kann. Ist diese in der Lage, den Befehl zu verarbeiten, wird die zugehörige Prozessmethode ausgeführt, oder in einem *master event calender* in die Warteschlange geschrieben. Ob eine Prozessmethode sofort ausgeführt wird, ist abhängig von der Definition. In diesem Zusammenhang gilt es, mit Hilfe der *Development Suite* herauszufinden, inwieweit eine Durchführung eines bestimmten Verhaltens unterbrochen werden kann.

Prozessmethoden können dabei grundsätzlich all das umsetzen, was der Entwickler fordert. Es sollte erwähnt werden, dass Prozessmethoden ursprünglich dafür eingerichtet worden sind, kurze und einfache Aufgaben auszuführen, welche dem von einem Menschen ausgewählten Aufgaben nahe kommen. Deshalb sollen Prozessmethoden im Idealfall:

- Die in den *equipment models* erhaltenen Informationen verwalten und auswerten.
- Das Verhalten und die Steuerung der *equipment models* regeln.
- Nachrichten für andere Einheiten generieren.
- Kommandos initiieren, welche weitere Prozessmethoden ausführen.

Die Art und Weise, wie FLAMES Prozessmethoden verarbeitet, soll dabei folgendes ermöglichen:

- Simulieren von menschlichem Verhalten durch interaktiven Benutzerinput.
- Verzögern der Ausführung von Prozessmethoden durch eine vorgegebene (ggf. randomisierte) Zeit, um auch den Zeitverzug durch Denkprozesse zu berücksichtigen.
- Definieren von Handlungsanweisungen vor Simulationsbeginn in FORGE oder interaktive Eingliederungen während der Simulation.
- Ausführen mehrerer Prozessmethoden mit einer einzigen Handlungsanweisung.
- Ermöglichen einer dynamischen Veränderung von *Units* und deren verknüpften Modulen während der Simulation.

Wird dem F-15 Jet beispielsweise die Handlungsanweisung `REQUEST_TAKEOFF` gegeben, dann wird dadurch die Prozessmethode `FCWPilotRequestTakeoff` aufgerufen. Aufgrund der notwendigen Kopplung mit Hilfe der GUI eines Air Traffic Controller (ATC) und der *Unit*, welche die Starterlaubnis einholt, organisiert die Prozessmethode wahrscheinlich die Kommunikation und leitet letztendlich weitere Prozesse ein, um den Start durchführen zu können. Für den Start selber existiert eine separate Handlungsanweisung `TAKEOFF`, welche die Prozessmethode `FCWPilotTakeoff` durchführt.

3.10.5 Beispiel eines kognitiven Modells

Abb. 3.14 zeigt die zugänglichen Informationen der Klasse `FCFWPilot`, welche dem kognitiven Modell `FCFWPilot` zu Grunde liegt. Unter dem Reiter *„Parameters“* lassen sich bei einigen der verfügbaren Beispielmodellen Einstellungen vornehmen. Bei dem `FCFWPilot` sind keine weiteren Einstellungen möglich.

Stattdessen ermöglicht der Reiter *„Methods“* eine Übersicht über die Prozessmethoden, welche abhängig vom Kommando bzw. der Nachricht eines *message models* aufgerufen werden. An dem Modell kann weder eine Änderung vorgenommen werden, noch lassen sich die Prozessmethoden aufrufen, um den dahinter stehenden Algorithmus zu untersuchen.

Von den in Abb. 3.14 dargestellten Kommandos lassen sich einige im *Unit* Editor unter dem Bereits beschriebenen Reiter *„Commands“* einfügen und dazugehörige Parameter - zum Beispiel Wegpunkte - einstellen. Die *Runtime Suite* ermöglicht also eine Simulation auf Basis von einigen wenigen Beispielmodellen, deren Funktionsweisen nicht einsehbar sind. Die Blackbox-Struktur verhindert eine Validierung der Simulation, da das Verhalten der Einheiten nicht nachvollziehbar ist.

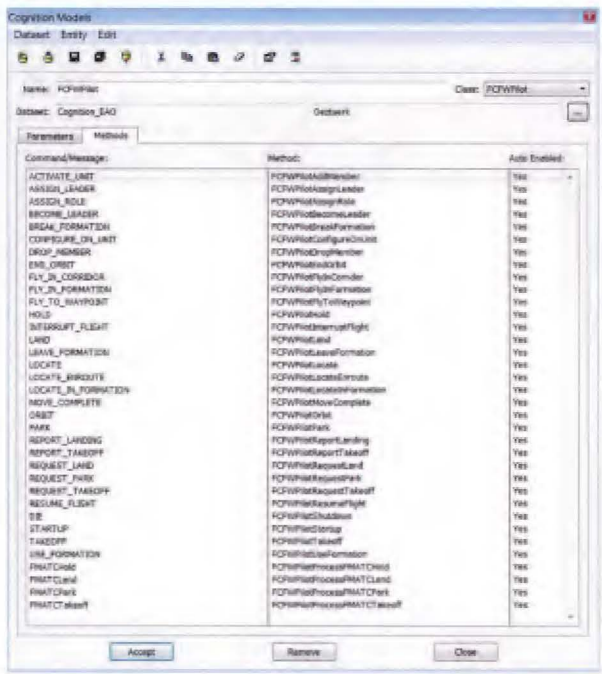

Abbildung 3.14: `FCFWPilot` – kognitive Modell.

3.11 Beispielszenario

Die Basis des Beispielszenarios bildet ein in den *Bundled Components* enthaltenes Szenario, welches mit der Datei `AirDefence.fsc` geladen wird. In Folge werden Möglichkeiten der FLAMES *Runtime Suite* zum Anpassen des Szenarios aufgezeigt. Die Visualisierung, wie sie sowohl in FORGE, als auch in FLASH dargestellt wird, zeigt Abb. 3.15 .

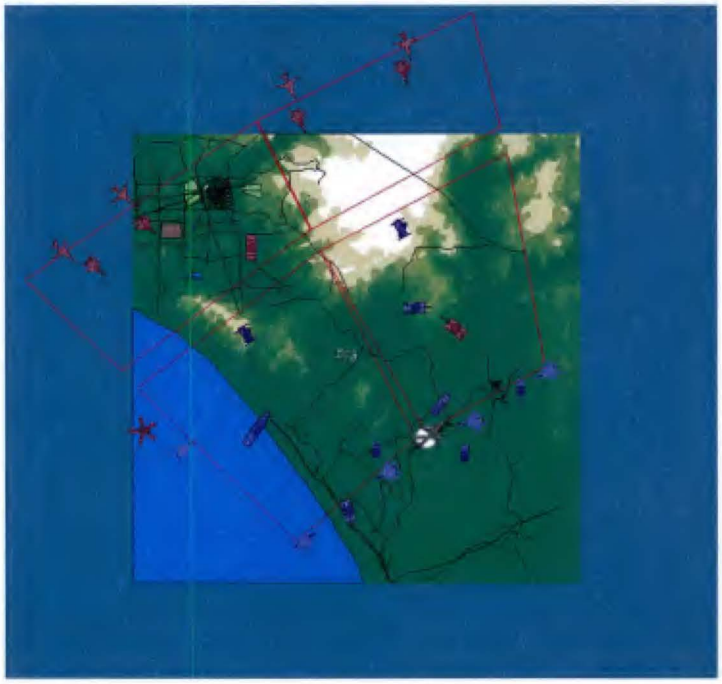

Abbildung 3.15: Grafische 2D Darstellung eines Szenarios.

Im Nordwesten ist eine Stadt mit einem Straßensystem dargestellt, welches Ausläufer auf der gesamten Karte hat. Im Südwesten befindet sich ein Gewässer, welches *„Pacific Ocean"* genannt und durch die Klasse `FCFLake` bestimmt wird. Über weitere Eigenschaften - wie zum Beispiel die Wassertiefe - sind keine Informationen verfügbar, bzw. einsehbar. Im Osten befindet sich ein weiterer Gebäudekomplex. Im Südosten - im Bereich des weißen Kreises mit grauem Querbalken - befinden sich weitere Gebäude. Bei dem Symbol handelt es sich um einen ATC , welcher nachträglich hinzugefügt wurde. Gleichzeitig werden beim Hinzufügen Parameter zur Start- und Landebahn, sowie zum Parkbereich für Flugzeuge abgefragt. Zu Beginn des Szenarios startet dort ein Passagierflugzeug vom Typ B757, welches standardmäßig dem blauen Team zugeordnet ist. Aus diesem Grund wird diese von dem Kampfjets des roten Teams sofort abgeschossen. Ändert man die Plattform der B757 und weist das Team *Civilian* zu, dann wird dieses von keinem der Jets beachtet. Auffällig ist an dieser Stelle, dass das Flugzeug mitten im Szenario in der Luft stehen bleibt. Das liegt daran, dass dem Flugzeug nach dem Kommando `FLY_TO_WAYPOINT` kein weiteres Kommando vorliegt. In diesem Fall wäre die Erwartung gewesen, dass das Flugzeug sich am Zielpunkt aufhält und dort

zumindest herumkreist. Dafür gibt es aber extra ein Kommando, welches `ORBIT` lautet und manuell hinzugefügt werden muss. Im Beispielszenario finden sich zwei weitere Flugzeuge, an denen das Verhalten beim Kommando `ORBIT` und beim Kommando `LAND` demonstriert wird. Das Beispielszenario stellt darüber hinaus einen Angriff von roten Bombern und Kampfjets dar, welche aus dem Nordwesten starten. Demgegenüber steht das blaue Team, welches neben eigenen Kampfjets auch zwei Luftabwehrsysteme vom Typ `MIM104Patriot` aufgestellt hat. Weiterhin sind vier Zonen erkennbar, welche verschiedene Lufträume darstellen. Diese können Korridore für Flugsysteme darstellen oder wie in diesem Fall Kampfkorridore (*Engagement Zones*) vorschreiben. Zusätzlich gehört ein Zerstörer zum blauen Team, welcher Luftabwehrraketen vom Typ `SM-2` verschießt. Dem entgegengestellt sind drei Mi-24 Kampfhubschrauber. Dem roten Team wurde außerdem eine Boden-Boden Rakete vom Typ `SS-21` zur Verfügung gestellt, welche als Ziel den Air Defence Controller (ADC) des blauen Teams zugewiesen bekommen hat. Eine Zerstörung von diesem hat unmittelbaren Einfluss auf das Ergebnis des Gefechts.

Weiterhin wurden Panzer und LKW-Konvois hinzugefügt, um diese ebenfalls in dem Beispiel zu simulieren. Einem LKW-Konvoi kann ein bestimmter Wegpunkt als Ziel vorgegeben werden. Ein LKW fährt dabei - gemäß seines *cognition models* - entlang des kürzesten Weges zum Zielpunkt auf einer Straße. Zufälligerweise ist bei dem Konvoi, welcher im Nordwesten der Karte - im Bereich des bebauten Gebiets - positioniert ist, ein interessantes Verhalten zu beobachten. Zu jedem Konvoi, wie auch zu einem *Platoon*, gehört ein Führungsfahrzeug (*Leader*) und auch Kolonnenfahrzeuge (*Follower*, bzw. bei kämpfenden Einheiten *Wingmen*). Während der *Leader* die Richtung vorgibt, folgen die *Follower* in der vorgegebenen Formation. In diesem Fall sollten die *Follower* dem *Leader* hinterherfahren. Stattdessen fährt das letzte Fahrzeug aus dem Konvoi eine andere Route, weil dies die kürzeste Strecke ist. Hier liegt ein Fehler im *cognition model* vor. Weiterhin gilt es, an dieser Stelle zu überprüfen, ob das Verhalten des Konvois an Kreuzungen dem entspricht, was man in der Realität erwarten würde.

Ebenfalls im Bereich des bebauten Gebiets wurde ein Schützenpanzer vom Typ M2A2 platziert. Diesem wurden ein Anfangswegpunkt und ein Zielwegpunkt mit dem Kommando `MOVE_TO_WAYPOINT` zugewiesen. Auf der Luftlinie befindet sich unbefahrbares Gelände und ein Wohnhaus, siehe Abb. 3.16 .

Abbildung 3.16: Vorgegebene Route des M2A2.

Korrektes Verhalten würde ein Umfahren des Geländes und des Wohnhauses bedeuten. Stattdessen fährt der Panzer auf dem direkten Weg über die Hindernisse. Diese Situation wird in Abb. 3.17 dargestellt.

Abbildung 3.17: Tatsächliche Route des M2A2.

Es ist zwar möglich, stattdessen das Kommando `MOVE_ON_ROUTE` vorzugeben, allerdings bedeutet dies, dass das Fahrzeug die nächst gelegene Straße aufsucht und befährt. Ein realistisches Verhalten im Gelände - ohne verfügbare Straßen - wird dadurch nicht erreicht. Hier fehlt eine Überprüfung der geplanten Route und eine anschließende selbständige Umfahrung.

Mittig auf der Karte ist ein grüner Panzer dargestellt. Standardmäßig sind drei Teams vorhanden: Team Rot (*Red Team*), Team Blau (*Blue Team*) und Zivilisten (*Civilian*). Zusätzlich wurde festgelegt, dass Team Rot der Feind von Team Blau ist. Der Versuch, eine zusätzliche Partei einzuführen, gestaltete sich als sehr aufwändig. Generell fehlt eine Möglichkeit, einen neuen Eintrag zu den jeweiligen Datenbanken hinzuzufügen. Stattdessen muss ein Umweg gegangen werden. Dafür wählt man einen vorhandenen Eintrag aus, in diesem Fall *Blue Team* und ändert den Namen. Dadurch wird derselbe Eintrag unter dem neuen Namen gespeichert. Will man nur etwas ändern, muss der ursprüngliche Eintrag zusätzlich gelöscht werden. In dem Beispielszenario wurde Team Grün (*Green Team*) hinzugefügt und im Anschluss eine Kopie der M1A1 Plattform erstellt. Diese wurde dem gerade erstellen Team Grün zugewiesen. Zu Demonstrationszwecken wurde die maximale Geschwindigkeit, sowie die Beschleunigung der Plattform modifiziert. Die genauen Bewegungs- und Lagedaten lassen sich während des Szenarios überwachen.

Unerwartet ist das Verhalten des Zerstörers im Bereich des Gewässers. Hier wurde provokanter Weise ebenfalls das Kommando `MOVE_TO_WAYPOINT` vorgegeben, wobei sich die Zielkoordinaten auf dem Festland befinden. Erwartet wurde, dass das Schiff an der Stelle im Gewässer stoppt, an der es nicht mehr weiterkommt. Stattdessen bewegt sich das Schiff während der Simulation weiterhin entlang der Luftlinie von den Start- zu den Zielkoordinaten. Die 3D-Ansicht in Abb. 3.18 verdeutlicht das realitätsferne Verhalten.

Abbildung 3.18: Zerstörer abseits des Gewässers.

In diesem Fall wäre es sinnvoll, automatisiert nach fehlerhaften Kommandos zu suchen. Eine Zielkoordinate zum Anfahren, welche sich nicht in einem Gewässer befindet, ergibt für ein Schiff keinen Sinn. Alternativ kann das Land als ein Hindernis gesehen werden. Eine genauere Untersuchung hat ergeben, dass für das kognitive Modell des Steuermannes die Option `Navigation points must be in water` vorhanden ist. Wird diese aktiviert, tritt das oben dargestellte Verhalten nicht auf. Allerdings werden auch danach festgelegte Wegpunkte von dem Zerstörer nicht angefahren, dieser bleibt demonstrativ an seiner Position. Eine Möglichkeit diese Option deaktivieren zu können, kann unterschiedlichste Gründe haben. Sie zeigt aber, dass es durchaus möglich ist, fehlerhaftes Verhalten zu korrigieren. Ein Einsatz einer bestimmten Einheit erfordert umfangreiche Kenntnis über dessen Zusammensetzung, was aber durch eine ausführliche Dokumentation ausgeglichen werden kann.

Mittig im südöstlichen Korridor befinden sich zwei Panzerverbände. Beide *Leader* haben den Befehl bekommen, feste Zielkoordinaten zu erreichen. Außerdem wurde das Kommando `ASSESS` zugewiesen. Dieses führt laut Beschreibung zu dem Verhalten, dass bei Feindkontakt geschossen wird. Das Gelände in diesem Abschnitt ist hügelig bis bergig. Erwartetes Verhalten ist hier eine realistische Gefechtsführung unter Berücksichtigung der natürlichen Hindernisse. Abb. 3.19 visualisiert die Situation.

Abbildung 3.19: Gefecht zwischen zwei Panzerverbänden.

Zu klären ist hierbei, ob die Detektion der Sensoren unter Berücksichtigung des Geländes korrekt verlief. Eine selbst programmierte Datenaufzeichnung könnte aufschlüsseln, wodurch die feindlichen Panzer aufgeklärt wurden. Denkbar ist auch eine Aufklärung durch Lufteinheiten, mit Weitergabe der Informationen durch die Kommunikationsmodule. Im Anschluss muss überprüft werden, ob die Bekämpfung in dem dargestellten Gelände durch die verfügbaren Waffensysteme möglich ist. Das Waffensystem der Panzer funktioniert anders, als das Waffensystem eines Jets. Aufgerufen wird die Klasse `FQWTurretGun`, welche Parameter zur Munitionsanzahl, Reichweite sowie zum Geschützturm fordert. Anstatt Munition abzufeuern, berechnet das Waffensystem unter Berücksichtigung einer Zeitverzögerung den Schaden am Ziel direkt. Laut Dokumentation wäre eine direkte Berechnung der Munition zu aufwändig. Ein eigenes Waffensystem im Nachhinein zu entwickeln und dabei ähnlich zu gestalten, ist wie bei den Kampfjets aber denkbar. Hier bietet es sich an, im Austausch mit der Ternion Corporation herauszufinden, wodurch eine direkte Modellierung und Berechnung der Panzermunition gescheitert ist. Neben den Details zur Munition sollte das Kampfverhalten der Panzer untersucht werden. Anstatt ein Panzergefecht zu führen, werden nur die jeweiligen aufgeklärten Panzer abgeschossen, während die anderen Panzer strikt ihrer Route weiter folgen.

Neben den dargestellten Situationen ist es außerdem möglich, den Wirkungsbereich von Sensoren einzelner Einheiten zu untersuchen. Betrachtet man den Sensor eines blauen Panzers aus dem in Abb. 3.19 dargestelltem Szenario, fällt direkt auf, dass es sich bei der Klasse `FQSGeometricSensor` um einen einfachen *field of view* Sensor handelt. Das bedeutet, dass der Panzer selbst nur das aufklären kann, was sich im seinem Sichtfeld befindet. In der Dokumentation ist hinterlegt, dass der Sensor keine Geländedaten oder andere Aspekte der Umgebung berücksichtigt, allerdings ist eine Funktion `Consider Terrain Masking` vorhanden. Einheiten sollten also nicht detektiert werden, wenn sich diese im Schutze des umliegenden Geländes, außerhalb des Sichtfeldes befinden. Mit Hilfe des *Sensor Coverage Tools* lässt sich die Erfassungsweite eines Sensors graphisch darstellen. Für die jeweiligen *Platoon Leader* des Panzergefechts ist dies in Abb. 3.20 zu sehen.

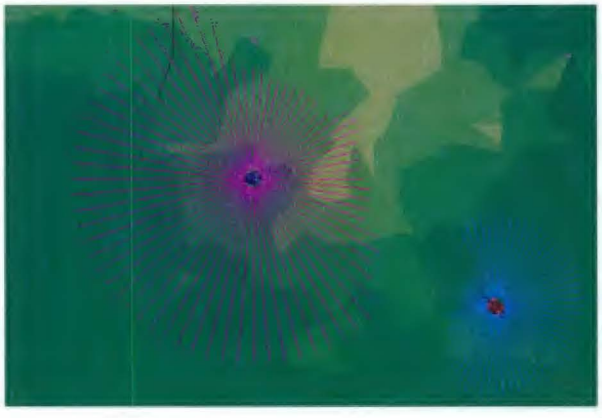

Abbildung 3.20: Sensorabdeckung im Panzergefecht.

Die dargestellte Situation lässt vermuten, dass die Geländedaten hinreichend berücksichtigt wurden und es sich bei der Situation aus Abb. 3.19 möglicherweise um einen Darstellungsfehler handelt.

Besonders hervorzuheben ist das Modell der Mi-24 Hubschrauber. Im Beispielszenario fliegt eine Gruppe aus drei roten Kampfhubschraubern, startend aus dem Westen, in Richtung des ATC von Team Blau. Dabei führt die Flugroute etwas verschoben unterhalb des Zerstörers entlang. Es wird folgendes Verhalten beobachtet: Die Hubschrauber lokalisieren korrekterweise den Zerstörer und greifen diesen an. Dabei werden diese durch einen Angriff von blauen Jets unterbrochen. Nachdem ein roter Hubschrauber und der angreifende blaue Jet zerstört wurden, führen die übrigen beiden Hubschrauber ihren Angriff auf den Zerstörer fort. Sobald dieser zerstört wurde, begeben sich die Hubschrauber in Richtung ihres ursprünglichen Ziels. Abb. 3.21 visualisiert das Verhalten. Der weiße Pfad stellt dabei die ursprüngliche Route dar.

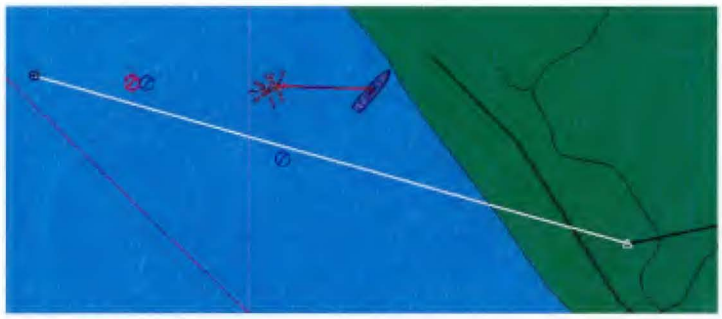

Abbildung 3.21: Verhalten der Kampfhubschrauber.

Um die Simulation teilweise zu validieren, erfordert es eine Beobachtung einer jeder einzelnen Einheit, da realistisches Verhalten, zum Beispiel im Falle der in der Luft stehenbleibenden Boeing und dem über Häuser fahrendem Panzer eine korrekte Programmierung erfordert. Gibt es keine Möglichkeit, realistisches Verhalten mit den vorhandenen Kommandos zu ermöglichen, müssen die kognitiven Modelle angepasst werden, wofür wiederum die *Development Suite* notwendig ist.

3.12 Data Farming-Fähigkeit

Um beim Planungsamt im Bereich M&S erfolgreich als Simulationstool eingesetzt werden zu können, müssen die erstellen Simulationsmodelle Data Farming-fähig sein. Dieser Begriff wurde im Grundlagenkapitel bereits geklärt. In diesem Kapitel wird die Umsetzung mit FLAMES beschrieben. Dazu wird die Analysis Option von FLAMES vorgestellt [19] und im Anschluss auf die Veröffentlichung von Johan Schubert zurückgegriffen [20], welcher dieses Verfahren bereits in der Praxis umgesetzt hat.

FLAMES Analysis Option

Laut der Ternion Corporation ist die FLAMES Analysis Option in der Lage, System Design Analysen und Monte Carlo Analysen durch die zusätzlichen Funktionen zu unterstützen.

Die *Enhanced Analysis Option* erlaubt es hierbei, Szenariovariablen zu definieren. Zuerst wird angegeben, um welchen Datentyp es sich konkret handeln soll. Im Anschluss kann für jeden Lauf ein Wert zugewiesen werden. Danach wird im Skript oder auch im interaktiven *Edit Unit* Fenster als Wert der Variablenname im Format `&"Name"` angegeben. Im Anschluss lässt sich in den Szenario Settings angeben, wie viele Durchläufe durchgeführt werden sollen. Dabei kann ein Lauf auch mehrfach durchgeführt werden, wobei dem Zufallszahlengenerator ein unterschiedlicher *Seed* übermittelt werden kann.

Um den Einfluss unterschiedlicher Parameter auswerten zu können, müssen Daten über die Szenarien bereitgestellt werden. FLAMES hat hierfür *data recorders*, die durch vorgefertigte Klassen eine Reihe an Informationen - während und am Ende der Simulation - in Dateien abgespeichert werden. In dem Beispielszenario werden folgende Daten herausgeschrieben:

- DETECTION: Sensorinformationen von freigeschalteten Sensoren.
- FUNCTION: Information zu aufgerufene Prozessmethoden, allerdings nur nachdem der Befehl `RECORD` manuell hinzugefügt wurde.
- KILLS: Informationen über die Ereignisse von eingesetzter Munition.
- PLAYBACK: Speichert die durchgeführten Simulationen in Wiedergabedateien, um diese später visualisiert darstellen zu können.
- TASK: Tasks, die eine *Unit* durchgeführt hat (dabei handelt es sich um externe Aufgaben).
- UNITS: Vorhandene *Units* des Szenarios.
- UNIT_STATUS: Statusinformationen zu den *Units* zu festen Zeitintervallen.

Das aufgebaute Szenario und die Einstellungen zur wiederholten Ausführung lassen sich per FIRE mittels Kommandozeilenbefehl starten. Zur Demonstration wurde das FLAMES Beispiel `EA0.fsc` geladen und alle verfügbaren *data recorders* ausgewählt. Gestartet werden die Simulationsdurchläufe durch den Befehl:

```
C:\Program Files (x86)\Ternion\FLAMES\v14.0_w32\bin\fire.exe EA0.fsc n 1 ne 12
```

Zusätzlich zum Dateipfad müssen die Nummern von dem ersten Lauf und dem letzten Lauf angegeben werden, wobei diese nicht außerhalb des in FORGE festgelegten Definitionsbereichs liegen dürfen. Szenariovariablen können auch aus einer externen Datei ausgelesen werden. Dafür wird zusätzlich der Dateipfad im Kommandozeilenbefehl angegeben.

In dem Beispiel fliegen Bomber von Team Rot von Norden nach Süden. Auf dessen Route befinden sich Flugabwehrsysteme von Team Blau. Für dieses Beispiel wurden die

Szenariovariablen „Höhe“ (1500, 750 und 500 Fuß) und „Geschwindigkeit“ (250, 450 und 800 Knoten) der Bomber erstellt. Dabei werden alle möglichen Kombinationen durchgeführt und jede Berechnung mit zwei unterschiedlichen *Seeds* wiederholt. In der Summe sind das 18 Simulationen, welche innerhalb von wenigen Sekunden durchgeführt werden. Die Ergebnisse, welche von den *data recorders* aufgezeichnet werden, haben eine fixe Struktur und einen fixen Inhalt und lassen sich deshalb nur in diese Richtung auswerten. Für detailliertere Untersuchungen sind eigene bzw. modifizierte *data recorders* notwendig. Außerdem wurde zu jedem Durchlauf eine `*.fpb` Datei angelegt. Dabei handelt es sich um die bereits angesprochenen Aufzeichnung der Simulation, welche bei Bedarf mit dem Programm FLASH abgespielt werden können, wenn das Ergebnis einer bestimmten Kombination der Szenariovariablen dies erfordert.

Durch dieses Beispiel lassen sich bereits vorsichtige Aussagen darüber treffen, welchen Einfluss unterschiedliche Flugparameter - die der Pilot frei wählen kann - auf das Überleben dieser Einheit haben. Außerdem zeigt sich beim Abspielen einer Aufzeichnung, dass die veränderten Rahmenbedingungen eine anderes Bekämpfungsverhalten der Flugabwehrsysteme, sowie eine angepasste Flugroutenwahl der Bomber verursachen.

Data Farming bei der FOI

Die FOI untersucht seit längerem den Einsatz eines *Decision Support Tools*, gestützt durch Data Farming Analysen. Ein möglicher Einsatz des Tools wird in [20] vorgestellt. Dort wird auch der Einsatz von FLAMES als durchführende Software angemerkt, dessen technische Modellierung aber in [21] erläutert wird.

Es wird ein Gefechtsszenario untersucht, in dem Einheiten von Team Blau einen Angriff auf Einheiten von Team Rot verteidigen sollen. Der Ort des Gefechts wird in Schweden simuliert. Dargestellt ist die Situation in Abb. 3.22 .

Abbildung 3.22: Darstellung des Szenarios [21, S. 5].

Die Autoren haben für ihre Untersuchung eigene Modelle entwickelt, um komplexere Fälle darzustellen. Der grobe Aufbau ist in Abb. 3.23 zu sehen. Diese Art der Modellierung

ist nicht ohne *Development Suite* möglich.

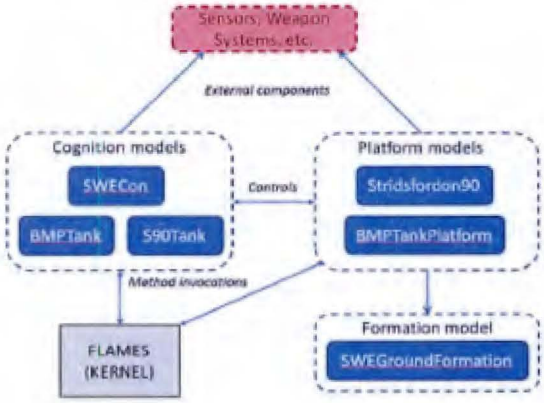

Abbildung 3.23: Cognition, platform, and formation models [21, S. 5].

Die *Development Suite* ermöglicht es dem Entwickler, Modelle zu schreiben, die auf den FLAMES Kernel zugreifen können. Das kognitive Modell `SWECon` wurde dafür entwickelt und greift während der Simulation auf folgende ausgewählte Daten zu und schreibt diese in eine Datei:

- Simulationsdauer,
- Gesamtanzahl der überlebenden Einheiten von Team Rot/Blau,
- Gesamtanzahl an Restmunition von Team Rot/Blau,
- Anzahl an Einheiten von Team Rot, welche die Verteidigungslinie durchbrochen haben,
- ausgewählte Route von Team Rot.

Das kognitive Modell `S90Tank` ist ein modifiziertes kognitives Modell der FLAMES Panzerklasse. Hier wurden unterschiedliches Angriffs- und Kommunikationsverhalten zusätzlich hinzugefügt. Die Plattform `Stridsfordon90` ist ein Modell, welches für die Definition einer neuen *Unit* benötigt wird. Es ermöglicht das von den Autoren vorgegebene Verhalten zur Bewegung und bestimmt, wie die Einwirkung von Munition - mit Hilfe einer Signatur - verarbeitet wird. Zusätzlich wurde eine eigenes *formation model* erstellt, welches die *wingmen* zufällig hinter dem *leader* platziert.

Die relevanten Parameter und möglichen Entscheidungen wurden durch ein Kriegsspiel (*war gaming*) ermittelt. Dadurch wurde ein Parameterraum aufgespannt, welcher bei dem Versuch, alle möglichen Kombinationen zu berücksichtigen, mit aktuellen Rechnern Jahre brauchen würde, um untersucht werden zu können. Deshalb werden 50.000 zufällig gewählte Kombinationen simuliert, wobei jedes der Szenarien 20 Mal mit unterschiedlichen *Seeds* simuliert wird. Daraus folgt, dass FLAMES genau 1.000.000 Simulationen durchführt, was gemäß der Referenz 10 Stunden auf einem Rechner mit 12

Kernen dauerte. Um diese Dauer zu erreichen, werden die Szenarios parallel berechnet. Neben der anfangs beschriebenen Möglichkeit, die Parameterräume in FLAMES festzulegen und die Kombinationen durch FIRE durchführen zu lassen, können die Parameter auch aus einer externen Datei eingelesen werden. Diese überschreibt zuvor im Programm festgelegte Werte. Die Autoren haben hierfür Programme erstellt, welche die zufälligen Parameterkombinationen erstellen und in eine Datei schreiben, welche im Anschluss von FIRE im *batch* Modus eingelesen und simuliert wird.

Mittels verschiedener Analysemethoden konnte hierbei letztlich ermittelt werden, dass der größte Einfluss auf den Erfolg bzw. Misserfolg der Einheiten von Team Rot abhängig von deren gewählter Route ist.

3.13 Anpassbarkeit der Software

Wie anpassbar die Software ist und welche praktischen Anwendungen tatsächlich angewendet wurden, wird in diesem Kapitel behandelt. Dabei konnte nur auf nicht eingestufte und daher öffentlich zugängliche Materialien zurückgegriffen werden. Davon sind leider nur wenige Beispiele von erfolgreichen Implementierungen von FLAMES vorhanden, Nichtsdestotrotz sollten die folgenden Beispiele Anregungen für die eigene Umsetzung liefern können und veranschaulichen, welche Arten der Zusammenarbeit mit der Ternion Corporation möglich sind.

3.13.1 Beispiele der Eigenentwicklung

Die Beispiele sind auf der Website der Ternion Corporation zu finden [22] und demonstrieren erfolgreiche Implementierungen, welche nicht ohne Grund auf der Website präsentiert werden. Die Beispiele wurden von den Kunden selber erstellt, unter Einhaltung der durch die Ternion Corporation festgelegten Richtlinien.

Air Force Research Lab

Das Air Force Research Lab hat mit FLAMES eine Simulationsumgebung aufgebaut, um verschiedene Waffensysteme in der Konzeptphase schnell und einfach zu modellieren [23]. Dabei werden verschiedene Kommunikationsmodule und Sensoren sowie unterschiedliche Munition simuliert. Leider ist den Informationen kein Datum zu entnehmen, lediglich die Informationsdatei ist von 2010, was aber nicht auf den Zeitraum des Projekts schließen lässt. Ziel Implementierung ist es, während der Konzeptphase Information über den Einfluss einzelner Features von zukünftigen Waffensystemen zu erhalten, um eine erste Vorabbewertung vorzunehmen.

BAE Systems Battlespace Dgitization Demonstrator

Informationen zu dem Projekt mit BAE Systems sind leider nirgendwo anders zu finden, außer auf der Website von der Ternion Corporation [24]. Bei BAE Systems handelt es

sich um einen multinationalen Rüstung-, und Luftfahrtkonzern mit Sitz in Großbritannien. Die Ternion Corporation hat Ende der 90er Jahren einen *Battlespace Digitization Demonstrator* entwickelt, welcher den Einfluss von digitalisierter Kommunikation auf dem Gefechtsfeld simulieren und analysieren sollte. Das Interessante an der Fallstudie ist, dass viele der Elemente, die für das Projekt entwickelt wurden, zum Beispiel eine Visualisierung des Szenarios oder die Plattform- und Kommunikationsmodule, heute Standardbestandteile von FLAMES sind.

FOI Information Fusion Demonstrator

Die FOI stellt sich als ein intensiver Nutzer von FLAMES heraus. Neben einigen Veröffentlichungen, in denen FLAMES als Werkzeug benutzt wird, hat die FOI unter anderem zwei Fallstudien bereitgestellt. Das erste Paper [9] wurde bereits in Abschnitt 3.1 zum Teil behandelt. Dort wurde unter anderem die Auswahl eines geeigneten Simulationsframeworks behandelt. In diesem Abschnitt wird ein weiteres Paper [25] hierzu näher erläutert.

Auch diese Fallstudie befasst sich mit dem Thema *Information Fusion Research*. Allerdings geht es nun um konkrete Implementierung eines *Information Fusion Demonstrators* in FLAMES. Hierbei sei angemerkt, dass die Bundeswehr im Rahmen der „Vernetzten Operationsführung“ (NetOpFü) ähnliche Ansätze verfolgt. Für die Implementierung wurde in FLAMES ein Szenario aufgebaut, welches einzelne Truppenteile und diverse Überwachungssysteme beinhaltet. Diese bewegen sich in dem Szenario fort, wobei die Überwachungssysteme und Aufklärungseinheiten währenddessen Aufklärungsdaten sammeln. Ziel der Fallstudie war es dabei, die während der Simulation generierten Daten in einem *Fusion Center* zu sammeln und zu einem Lagebild zusammenzufügen. Um dies ausreichend zu modellieren, werden mittels FLAMES die Sensoren der jeweiligen Einheiten gebaut. Mit den selbst erstellten Sensormodellen in FLAMES hat das Team folgendes modellieren können:

- Die Art der Aufklärung, zum Beispiel optische, akustische oder seismische Aufklärung.
- Die Reichweite des Sensors, abhängig von dem Gelände, beispielsweise das Sichtfeld.
- Mobile Sensoren, zum Beispiel an abgesessene Kräfte, an Fahrzeugen oder Flugobjekten.
- Stationäre Sensoren, welche sich nicht verschieben können.

Zusätzlich wurden eigene kognitive Modelle entwickelt, welche es ermöglichen, dass die Truppenteile oder autonomen Flugobjekte die Wahl der Route selbstständig bestimmen. Die Entscheidung ist abhängig von der Umgebung, von der aus aufgeklärt wird und von der Qualität der Signale, welche der Sensor empfängt. Im Zentrum der Arbeit stand dabei, auf welchen Methoden und Algorithmen das kognitive Modell für das *Fusion Center* aufbauen kann, um die von den Sensoren gesammelten Daten zu aggregieren und Fehlerquellen zu eliminieren.

3.13.2 Outsourcing/ Co-Development

Die Ternion Corporation selbst bietet außerdem an, eine angepasste Simulationssoftware für den Kunden zu entwickeln. Dies kann der Bundeswehr die Möglichkeit geben, den Entwicklungsprozess von auf dem FLAMES Simulationsframework basierende Softwarelösungen in Kooperation mit der Ternion Corporation zu beschleunigen. Hier stellt sich die Frage, inwieweit die Fähigkeiten der Ternion Corporation ausreichen mögen, um den Anforderungen der Bundeswehr gerecht zu werden. Hierfür böte sich die Möglichkeit eines Erfahrungsaustauschs mit den U.S. Streitkräften an, um aufkommende Fragen zu klären und ggf. vorhanden Abstimmungsschwierigkeiten frühzeitig zu erkennen, respektive diesen entgegenzuwirken.

Auf der Website der Ternion Corporation finden sich zwei öffentlich zugängliche Beispiele über erfolgreiche Implementierung der durch die Firma umgesetzten Software.

Zum einen ist die Ternion Corporation an der Entwicklung des Integrated Training Capability (ITC)-Tools der NATO beteiligt gewesen [26]. Dieses Tool dient als Simulations- und Trainingsumgebung, um mit der Integrated Command and Control Software for Air Operations (ICC) realitätsnah und ressourcensparend zu üben. Die ICC -Umgebung wird von der NATO zum Informationsmanagement und als *Decision Support Tool* verwendet [27]. Durch die Kopplung mit dem ITC - Tool wird der vollständige Ablauf einer Gefechtssituation simuliert und der ICC -Bediener am echten System trainiert. Die Ternion Corporation will mit dem Beispiel demonstrieren, wie günstig und schnell sich derartige Simulationen durch den Grundbaustein FLAMES entwickeln lassen.

Zum anderen hat die Ternion Corporation ein Command and Control Weapon System Part Task Trainer (C2WSPTT) mitentwickelt. Dieser wird von der US-amerikanischen Luftwaffe genutzt und sei eine preiswerte M&S Lösung für Training, Testen & Experimentieren und Analyse. Das C2WSPTT tauscht Informationen mit anderen Systemen aus und ermöglicht eine interaktive Kontrolle des Szenarios. Wie bei dem von der NATO genutzten System ermöglicht FLAMES das Ausbilden von Operateuren an dem realen System mithilfe simulierter Daten. Ein besonderer Nebenaspekt dieses Beispiels ist dabei, dass das System an einem einzigen Arbeitsplatz läuft und die Operateure dadurch zeit- und kostensparend inhouse ausgebildet werden können. Weitere Informationen zu dem System sind in [28] zu finden. Es existiert weiterhin eine kompakte Dokumentation über das entwickelte System, welche auf dem *Discovery Channel* ausgestrahlt wurde [29].

4 Fazit und Ausblick

Diese Arbeit ermöglichte eine Grundsatzbetrachtung des FLAMES Simulationsframeworks, sowie den dazugehörigen Komponenten. Es wurden Möglichkeiten aufgezeigt, eine Simulationsumgebung aufzubauen und die Simulation mit anschließend auswertbaren Daten durchzuführen. Für die vorliegende *Runtime Suite* waren vorhandene Grenzen sehr schnell aufgezeigt. Viele wichtige Einblicke waren nur eingeschränkt möglich und notwendige Erkenntnisse stark abhängig von der Software-eigenen Dokumentation.

Im nächsten Schritt empfiehlt es sich daher, an die Ternion Corporation heranzutreten, um eine kostenlose Testversion der *Development Suite* zu erhalten. Hierbei sollten wenigstens alle zusätzlichen Optionen vorhanden sein. Nur so lässt sich umfassend testen, ob ein Einsatz der Softwareumgebung als Werkzeug zur Erstbewertung von M&S-Fragestellungen, bzw. als Data Farming-fähige Analysesoftware taugt. Für eine bessere Bewertung der Fähigkeiten und Möglichkeiten, sowie insbesondere der Grenzen von FLAMES ist eine intensive Begutachtung des Quellcodes notwendig. Erst danach kann bestätigt werden, ob eine generelle Validierung und Verifizierung der aufgebauten Modelle möglich ist. Außerdem lassen sich erst mit der *Development Suite* eigene Plug-Ins und Anwendungen zum Szenariomanagement, zum Datenexport und -import, sowie eigene Simulatoren und Visualisierungssysteme entwickeln. Insbesondere die Modellierung eigener kognitiven Modelle, Sensoren, Fahrzeugklassen oder auch Schnittstellen zu bereits vorhandenen Systemen ist ohne die Entwicklerversion nicht möglich.

Es hat sich nach der Untersuchung diverser Fallstudien gezeigt, dass die Ternion Corporation durchaus bereit ist, die Software zu erweitern, um auch neueren Anforderungen gerecht zu werden. Auch durch den Einsatz der Software bei diversen militärischen und nicht-militärischen Nutzern profitiert FLAMES von der regelmäßigen Integration neuer Standardfunktionalitäten.

Zusätzlich sollte ein Kontakt zur FOI - wenn nicht schon vorhanden - aufgebaut werden. In diesem Fall bietet es sich an herauszufinden, wie aktiv die Schweden die Software noch einsetzen und welche Erfahrungen sie gemacht zu haben. Vorteile eines Frameworks ergeben sich erst mittel- bis langfristig, hier sollten erste Erfahrungswerte bei den Schweden vorliegen. Zusätzlich sollte ein Kontakt zu den anderen deutschen Einheiten aufgebaut werden, welche im Abschnitt 3.3 genannt wurden. Möglicherweise wird die Software noch genutzt. Falls nicht, können Gründe für das Einstellen der Nutzung genannt werden. Ebenfalls ist es empfehlenswert, den Kontakt zum HQ AIRNORTH aufzubauen. Zwar verfolgt der Einsatz von FLAMES dort andere Ziele, dennoch sind jegliche Dokumentationen und eigene bereits entwickelte Modelle hilfreich. Die Kontakte sind nicht nur wertvoll um herauszufinden, ob der Einsatz von FLAMES einen Mehrwert bringt, sondern können durch den Austausch von Templates und Modellen, Zeit und Geld einsparen.

In zukünftigen Untersuchungen können die vorhandenen Beispielmodelle im Rahmen

der Möglichkeiten zerlegt und deren Interaktion mit der Umwelt untersucht werden. Im Anschluss daran können die vorhandenen kognitiven Modelle begutachtet werden, sofern die *Development Suite* vorhanden ist. Realistisches Verhalten, wie es eigentlich bei der - in der Luft stehengebliebenen - Boeing und dem - über Häuser fahrendem - Panzer erwarte wurde, erfordert eine korrekte Programmierung. Hierbei ist es anzunehmen, dass eine Anpassung der zugehörigen kognitiven Modelle notwendig ist. Weiterhin gilt die Frage zu klären, ob FLAMES in der Lage ist, ein von einer Einheit aktuell ausführendes Verhalten mittendrin unterbrechen kann - sofern dies gefordert wird. Dafür liefert die Dokumentation lediglich den Hinweis, dass eine Handlungsanweisung in die Warteschlange gestellt werden kann oder sofort ausgeführt wird, was wiederum abhängig von den Eigenschaften der Prozessmethode ist. Der Zusammenhang ist mit der vorliegenden Version nicht überprüfbar. Ebenso sollte das Verhalten einer Einheit untersucht werden, wenn diese einen Befehl erhält, der nicht von ihr verarbeitet werden kann.

Weiterhin können darauf aufbauend eigene Modelle, sowie *Pattern* erstellt werden und durch die Integration der eigener Geoinformationsdaten erste eigene Lagebilder und Szenarien erstellt werden. In diesem Zusammenhang gilt es dazu herauszufinden, inwieweit die vom GeoInfoDBw bereitgestellten Geländedaten genutzt werden können. Dazu kann entweder für die vorhandene Version der fehlende FACT Importer beschafft, oder mithilfe der *Development Suite* ermittelt werden, ob sich ein eigener Importer ohne großen Aufwand selber entwickeln lässt. Als letzte Möglichkeit sollte die Notwendigkeit einer zusätzlichen Lizenz der Software *Terra Vista* überprüft werden. Möglicherweise wird eine solche Anschaffung durch den GeoInfoDBw unterbunden, denn dieser ist alleinig für die Versorgung von Daten solcher Art zuständig. Gerade auf Fragestellungen im Zusammenhang mit Geländedaten sollten die Erfahrungen anderer Nutzer schnelle Antworten liefern können.

Literatur

[1] Abb. 1.1URL: http://www.planungsamt.bundeswehr.de/resource/resource/MzEzNTM4MmUzMzMyMmUzMTM1MzMyZTM2MzIzMDMwMzAzMDMwMzAzMDY5NmMzMjM0NmE2Nzc2MzkyMDIwMjAyMDIw/image_original-456x480.jpg (besucht am 08.10.2016).

[2] H.-J. Bungartz, S. Zimmer, M. Buchholz und D. Pflüger. *Modellbildung und Simulation - Eine anwendungsorientierte Einführung*. Springer-Verlag, 2013.

[3] Bundesministerium der Verteidigung. *Konzeption der Bundeswehr*. URL: http://www.planungsamt.bundeswehr.de/portal/a/plgabw/!ut/p/c4/04_SB8K8xLLM9MSSzPy8xBz9CP3I5EyrpHK9gpz0RCCVX1Sal5KTmJ6ap5edn1eVWgBSF5-SWhSfBJRILS5PzSjSL8h2VAQAXHv_5A!!/ (besucht am 19.09.2016).

[4] ITIS GmbH. *Simulationsgestützte Analysen in der Bundeswehr*. 2013.

[5] G. Horne und K.-P. Schwierz. „Summary of Data Farming“. In: *Axioms* 5.1 (2016), S. 8.

[6] G. Horne und S. Seichter. *Data Farming Support to NATO: A summary of MSG-088 Work*. 2013.

[7] R. G. Sargent. „Verification and Validation of Simulation Models“. In: *Proceedings of the 37th Conference on Winter Simulation*. 2005, S. 130–143.

[8] Ternion Corporation. Abb. 3.1URL: http://www.ternion.com/live/wp-content/uploads/2011/10/FLAMES-Uses.png (besucht am 30.08.2016).

[9] P. Horling, V. Mojtahed, P. Svensson und B. Spearing. „Adapting a commercial simulation framework to the needs of information fusion research“. In: *Information Fusion, 2002. Proceedings of the Fifth International Conference on*. Bd. 1. 2002, 220–227 vol.1.

[10] Ternion Corporation. Abb. 3.2URL: http://www.ternion.com/live/wp-content/uploads/FLAMES-RS-FORGE.png (besucht am 30.08.2016).

[11] Ternion Corporation. *Customers*. URL: http://www.ternion.com/customers/ (besucht am 30.08.2016).

[12] AIRCOM - Allied Air Command. *History of NATO's Air Headquarters at Ramstein Air Base*. URL: https://www.airn.nato.int/history (besucht am 30.08.2016).

[13] *Airbus Group: Zwischen Hoffen und Bangen*. URL: http://www.bundeswehr-journal.de/2014/airbus-group-zwischen-hoffen-und-bangen/#more-2608 (besucht am 30.08.2016).

[14] J. Schubert. *About me: Johan Schubert.* URL: https://www.kth.se/profile/johsch/ (besucht am 30. 08. 2016).

[15] NATO STO. *Data Farming in Support of NATO (MSG-088).* URL: https://www.cso.nato.int/ACTIVITY_META.asp?ACT=1960 (besucht am 30. 08. 2016).

[16] NATO STO. *Developing Actionable Data Farming Decision Support for NATO (MSG-124).* URL: https://www.cso.nato.int/ACTIVITY_META.asp?ACT=2802 (besucht am 30. 08. 2016).

[17] *PERFORMANCE SPECIFICATION DIGITAL TERRAIN ELEVATION DATA (DTED).* URL: https://dds.cr.usgs.gov/srtm/version2_1/Documentation/MIL-PDF-89020B.pdf (besucht am 08. 09. 2016).

[18] Ternion Corporation. Abb. 3.4URL: http://www.ternion.com/live/wp-content/uploads/2011/10/3d-simulation-database.gif (besucht am 30. 08. 2016).

[19] Ternion Corporation. *Enhanced Analysis.* URL: http://www.ternion.com/analysis/ (besucht am 30. 08. 2016).

[20] J. Schubert und P. Hörling. *Decision Support for Simulation-Based Operation Planning.* Swedish Defence Research Agency, 2016.

[21] F. Moradi und J. Schubert. *Simulation-based Defense Planning.* Swedish Defence Research Agency, 2014.

[22] Ternion Corporation. *Build Your Own Custom Simulations.* URL: http://www.ternion.com/build-your-own/ (besucht am 09. 10. 2016).

[23] Ternion Corporation. *Analysis Customer Success - Air Force Research Lab Munitions Directorate Uses FLAMES.* URL: http://www.ternion.com/print/FLAMES-ARA-AFRL-Case-Study.pdf (besucht am 30. 08. 2016).

[24] Ternion Corporation. *BAE - Battlespace Digitization Demonstrator.* URL: http://www.ternion.com/bae-battlespace-digitization-demonstrator/ (besucht am 08. 09. 2016).

[25] P. Svensson und P. Horling. „Building an information fusion demonstrator". In: *Information Fusion, 2003. Proceedings of the Sixth International Conference of.* Bd. 2. 2003, S. 1316–1323.

[26] Ternion Corporation. *NATO CAOC Training.* URL: http://www.ternion.com/nato-caoc-integrated-training-capability/ (besucht am 05. 09. 2016).

[27] NCI Agency. *Integrated Command and Control Software for Air Operations (ICC).* URL: https://npc.ncia.nato.int/Pages/NATO-wide-Integrated-Command-and-Control-Software-for-Air-Operations-(ICC).aspx (besucht am 05. 09. 2016).

[28] Ternion Corporation. *USAF AOC Training.* URL: http://www.ternion.com/usaf-aoc-training/ (besucht am 05. 09. 2016).

[29] Ternion Corporation. *Ternion® Featured on Discovery Channe.* URL: `http://www.ternion.com/ternion-featured-on-discovery-channel` (besucht am 05. 09. 2016).