HELMUT SCHMIDT
UNIVERSITÄT
Universität der Bundeswehr Hamburg

Henry Winkel

Distributed Interactive Simulation (DIS)

Studienarbeit

Fakultät für Elektrotechnik

Studiengang: Elektrotechnik und Informationstechnik Matr.-Nr. 874650 / ET2014
Betreuer: Univ.-Prof. Dr. phil. nat. habil. Bernd Klauer
Weiterer Prüfer: Univ.-Prof. Dr.-Ing. habil. Udo Zölzer

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)

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

DIS	Distributed Interactive Simulation
IEEE	Institute of Electrical and Electronics Engineers
PDU	Protocol Data Units
IST	Institute for Simulation and Training
UCF	University of Central Florida
SimNet	Simulator Network
HLA	High Level Architecture
ALSP	Aggregate Level Simulation Protocol
DE	Directed Energy
WGS84	World Geodetic System 1984
BIH	Bureau International de l'Heure
NIMA	National Imagery and Mapping Angency
DoD	Department of Defense
MOVES	Modeling, Virtual Environments and Simulation
ESPDU	Entity State PDU
GPS	Global Positioning System
UDP	User Datagram Protocol

Inhaltsverzeichnis

1 Einleitung

bla bla DIS

2 DIS

2.1 Überblick

DIS steht für eine im Institute of Electrical and Electronics Engineers (IEEE) 1278 von 1993 definierten Standard zur Steuerung und Überwachung von Simulationen. Der Standard wird in militärischen sowie in zivilen Simulationen verwendet und ermöglicht das Erstellen von simmulierten Lagebildern sowie den Datenaustausch der Teilnehmer einer vernetzten Simulation in Echtzeit. Dabei können die Simulationsumgebungen der Teilnehmer unterschiedlich sein. Um dies zu ermöglichen, wird je eine standardisierte Protocol Data Units (PDU) für einen Teilnehmer erstellt und an die anderen Teilnehmer im Netzwerk verteilt. Dies ermöglicht die Kommunikation der Teilnehmer.
DIS wurde in einer Reihe von Workshop des Institute for Simulation and Training (IST) der University of Central Florida (UCF) entwickelt. Dabei baut DIS auf den in den 1980er entwickelten Simulator Network (SimNet) auf, der für Trainingssimulationen der US Militär geschaffen wurde. Im Jahr 1996 wurde High Level Architecture (HLA) entwickelt, dass aus der Verbindung von DIS und Aggregate Level Simulation Protocol (ALSP) besteht.
Über die Jahre wurde DIS verbessert und 2012 in der aktuellen Versionen 7 veröffentlicht. Diese Version wurde im IEEE Standard 1278.1-2012 beschrieben.

1. DIS PDU Version 1.0 - 1992
2. IEEE Standard 1278 - 1993
3. DIS PDU Version 2.0 - 1993
4. DIS PDU Version 2.0 - 1994
5. IEEE Standard 1278.1 - 1995
6. IEEE Standard 1278.1a - 1998
7. IEEE Standard 1278.1 - 2012

[8] [1] [6]

2.2 Protokoll

Um die einzelnen Simulationen miteinander zu verbinden, wird eine einheitliche „Sprache“ benötigt. Die einzelnen Simulationen erstellen PDUs und broadcasten sie ins Netzwerk, sodass alle anderen Teilnehmer sie empfangen und verarbeiten können. Hierfür wurden in DIS verschiedene Arten von PDUs implementiert, die in PDU-Familien zusammengefasst wurden. Dabei hat jede PDU eine andere Bedeutung und andere Attribute. Im Sinne von DIS kann eine Entity eine ortsfeste oder ortsveränderliche Einheit darstellen, die spezifische Attribute besitzt. PDUs enthalten kodierte Informationen, die im „Enumeration and Bit Encoded Values for Use with Protocols for Distributed Interactive Simulation Applications “ enthalten sind. Dabei werden die Informationen nummerisch dargestellt. Folgende PDU-Familien gibt es:

- Entity information/interaction
- Warfare
- Logistics
- Simulation Management
- Distributed Emission Regeneration
- Radio Communications
- Entity Management
- Minefield
- Synthetic Environment
- Simulation Management with Reliability
- Live Entity
- Non-Real Time protocol
- Information Operations

Alle Informationen die eine PDU enthalten soll, sind im IEEE Standard festgelegt. Alle PDUs enhalten einen PDU Header, der unter anderem die Protokollversion und den PDU Typ enthält. Weitere Informationen die der PDU Header beinhaltet, sind:

- Protocol Version Field

- Exercise Identifier Field
- PDU Type Field
- Protocol Family Field
- Time Stamp Field
- PDU Length Field
- Padding Field

Im folgenden Absatz wird die Entity information/interaction Familie und die Warfare Familie beschrieben. Diese Familien spielen eine große Rolle innerhalb einer militärischen Simulation.

2.2.1 Entity information/interaction

Die Entity information/interaction Familie besteht aus der ESPDU, der Entity State Update PDU und der Collision PDU. Die wichtigste PDU ist die ESPDU, diese enthält Informationen über den Zustand einer Entity, wie zum Beispiel eine eindeutige Entity ID. Eine weitere wichtige Information ist der Entity Typ, der angibt, welcher Nation und Kategorie die Entity angehört.

Field size	Entity State PDU fields	
64	Entity Type	Entity Kind - 8 Bit Domain - 8 Bit Country - 16 Bit Category - 8 Bit Subcategory - 8 Bit Specific - 8 Bit Extra - 8 Bit

Tabelle 2.1: Entity Type Field [9]

	Kind	Domain	Country	Category	Subcategory	Specific	Extra
Leopard - A6	1	1	78	1	3	2	
M1A2 - Abrams	1	1	225	1	1	3	
T-72B	1	1	222	1	2	6	
Mercedes - C240	1	1	78	81	43	3	
F220 - Hamburg	1	3	78	6	3	2	

Tabelle 2.2: Entity Type Examples

Wie in Tabelle 2.2.1 zu sehen ist, bietet die neuste „Reference for Enumerations for Simulation Interoperability“ in der Version 21 [7] viele verschiedene Beschreibungen für militärische und zivile Einheiten der unterschiedlichsten Länder.
Des Weiteren werden in einer ESPDU die Geschwindigkeit, die Beschleunigung und die Orientierung angegeben. Der Ursprung der drei Vektoren befindet sich im Massenschwerpunkt. Die Geschwindigkeit und die Beschleunigung setzen sich aus einem dreidimensionalen Vektor mit den Komponenten X,Y und Z zusammen. Dabei zeigt die X Komponente aus der Vorderseite der Entity. Die Richtung der verbleibenden Komponenten zeigen entsprechend eines rechtwinkligen Koordinatensystems aus der Entity heraus. Der Vektor der die Orientierung angibt, ist in eine Psi (ψ), eine Theta (θ) und eine Phi (ϕ) Komponente.
Neben den genannten Informationen die eine ESPDU enthält, gibt es noch weitere die im IEEE Standard beschrieben sind. Diese Informationen sind, wie in Tabelle 2.2.1 gezeigt, gegliedert. Jedoch haben sie ihren eigenen Inhalt.
Die Collision PDU wird von einer Entity erstellt, wenn diese eine Kollision mit einer anderen Entity oder einem Objekt feststellt. Die Collision PDU enthält die Entity IDs der Entity's die zusammenstoßen. Da diese PDU im weiteren Verlauf nicht weiter benutzt wird, wird an dieser Stelle davon abgesehen die Collusion PDU weiter zu beschreiben. Genaue Informationen können dem „IEEE Standard 1278.1-2012“ [9] entnommen werden.
[6] [9] [3]

2.2.2 Entity Warefare

Die Entity Warefare Familie besteht aus der Fire PDU, der Detonation PDU , der Directed Energy (DE) Fire PDU und der Entity Damage Status PDU. Die Fire PDU enthält Informationen über das Abfeuern von Waffen, dabei wird unterschieden, ob es sich um eine Waffe handelt oder eine an der Entity angebrachten Verbrauchsmaterial. In DIS gilt alles als Waffe, das auch als Waffe benutzt werden kann, das kann unter anderem konventionelle Munition, wie beispielsweise Bomben, Raketen, Projektile oder Energiewaffen. Als Verbrauchsmaterial gelten Gegenmaßnahmen gegen Munition wie zum Beispiel IR-Täuschkörpern und sonstiges Verbrauchsmaterial, wie beispielsweise Signalfackeln. Die Fire PDU enthält unter anderem die Entity ID, der Entity, die gefeuert hat, die beschossen wird, sowie den Abschussort, die verwendete Munition und ob es sich um das Abfeuern einer Waffe oder Verbrauchsmaterial handelt. Eine Fire PDU wird erstellt wenn den Teilnehmern das Abfeuern einer Waffe oder eines Verbrauchsmaterials mitgeteilt werden soll. Sie wird von der Simulation erstellt, die schießt. Sollen andere Entitys die Munition erkennen und eventuell darauf reagieren, muss die Simulation, die die Fire PDU erstellt, eine ESPDU erstellen, welche die Munition darstellt.

	Kind	Domain	Country	Category	Subcategory	Specific	Extra
AIM-9B Sidewinder	2	1	225	1	1	1	
Tomahawk	2	10	225	1	4		
5.56 mm	2	8	78	2	1		
120 mm HE	2	9	78	2	5	1	

Tabelle 2.3: Beispiel Munition

Da Raketen, wie die AIM-9B Sidewinder, Marschflugkörper, wie die Tomahawk, klassische Projektile wie die 5,56 mm Munition, für Sturmgewehre oder 120mm HE Geschosse, für zum Beispiel Panzer, als Munition gelten, können diese durch eine ESPDU mit den entsprechenden Attributen repräsentiert werden.
Die Detonation PDU enthält, ähnlich wie die Fire PDU, die Entity ID der Entity, die gefeuert hat und die, die beschossen wird. Es ist neben anderen Informationen auch der Ort der Detonation, sowie die Entity ID der Munition bzw. des Verbrauchsmaterials, welches abgefeuert wurde, enthalten. Eine Detonation PDU wird erstellt, wenn der Einschlag oder die Explosion einer Munition, das Umsetzen von Verbrauchsmaterial oder sonstigen Explosionen den Teilnehmern mitgeteilt werden soll. Bei der DetonationPDU

kommt neben der Munitions ID ein sogenannter „Burst Descriptor Record “zur Verwendung. Dieser enthält neben der Munition auch die Angaben des verwendeten Sprengkopfs, des Zünders, die Anzahl der Schüsse und die Feuerrate. Soll die Entity, bei der die Munition einschlägt und detoniert, als zerstört gelten, muss die ESPDU deaktiviert werden. Wird die Entity nicht zerstört, kann der ESPDU ein Schaden zugewiesen werden. Entstehen bei der Detonation Fragmente der ursprünglichen Entity, können diese Fragmente ebenfalls als ESPDUs dargestellt werden.
[6] [9] [3] [5]

2.2.3 Koordinaten System

Das Koordinatensystem, nach dem in DIS Positionsangaben getätigt werden, ist das World Geodetic System 1984 (WGS84). WGS84 ist ein in den 1980ern entwickeltes geodätisches Referenzsystem zur Standardisierung von Positionsangaben auf der Erde und im erdnahen Weltraum. WGS84 besteht aus mehreren Bestandteilen. Es beinhaltet einen Referenzellipsoiden, welcher der stark vereinfachten Oberfläche der Erde angepasst wurde und für Ortsangaben in Form von geografischer Länge und Breite genutzt wird. Des Weiteren beinhaltet WGS84 einen detailliertes Modell der Erde, welches stark von dem Referenzellipsoiden abweicht. Dieses Modell wird als Geoid bezeichnet.

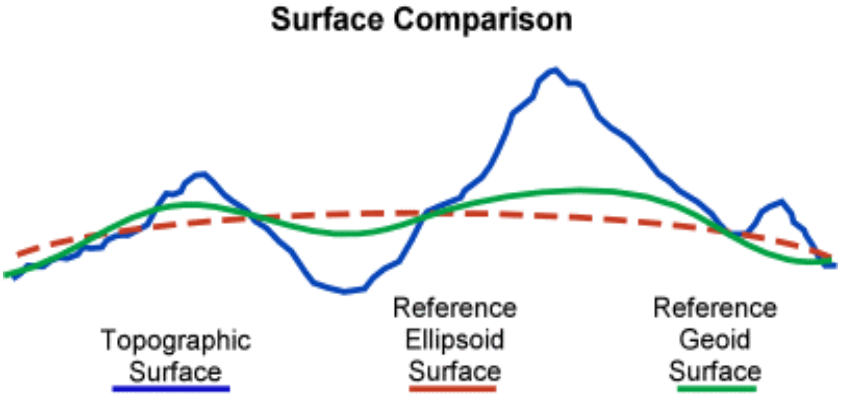

Bild 2.1: Unterschied Geoid, Ellisoid und realer Oberfläche [10]

Als dritte Komponente beinhaltet WGS84 eine dreidimensionales Koordinatensystem. Wie man in der Abbildung 2.3 sehen kann, handelt es sich hierbei um kartesisches Koordinatensystem, welches seinen Ursprung im Erdmittelpunkt hat. Die Orientierung des Koordinatensystems wurde durch das Bureau International de l'Heure (BIH) vorgegeben. Die Z-Achse geht vom Ursprung aus durch den Nordpol des Ellipsoiden. Sie dient dabei

als Rotationsachse. Die X-Achse verläuft vom Ursprung aus durch den Nullmeridian auf der Höhe des Äquators. Dabei steht sie senkrecht zur Z-Achse. Die Y-Achse verläuft senkrecht zu den beiden anderen Achsen auf der Höhe des Äquators und komplettiert das kartesische Koordinaten System.

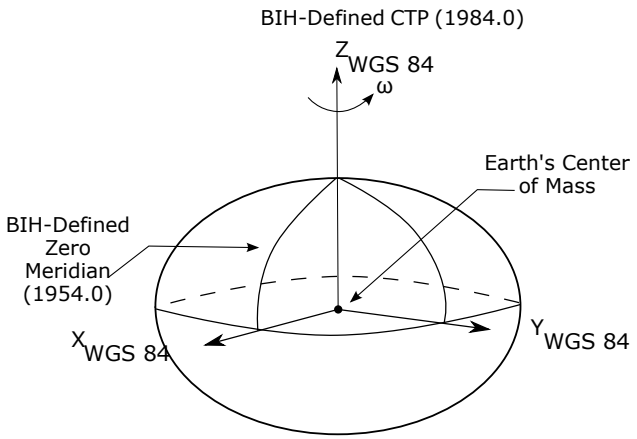

Bild 2.2: WGS 84 [4]

DIS nutzt das die dreidimensionale Angabe von Koordinaten als globales Koordinatensystem. Dies ermöglicht die genaue Positionierung von Entitys. Zusätzlich zu dem globalen Koordinatensystem hat jede Entity noch ihr eigenes lokales Koordinatensystem. Dieses besteht auch aus einem kartesischen Koordinatensystem mit den Komponenten X,Y und Z. Die Komponenten haben als gemeinsamen Ursprung den idealisierten Massenschwerpunkt der Entity. Die X Koordinate zeigt dabei aus der Front der Entity heraus. Die Y Koordinate zeigt im rechten Winkel zur X Achse an der rechten Seite der Entity heraus. Die Z Achse zeigt vom Ursprung nach unten. Die Größenordnung des Koordinatensystems ist wie beim WGS84 in Metern.

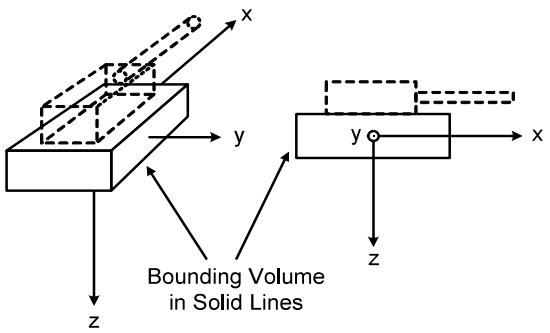

Bild 2.3: Entity Koordinatensystem [9]

2.3 Anforderungen

Das DIS Protokoll ist ein Mittel um verschiedene Simulationen mit einander zu verbinden. Jedoch müssen dafür einige Anforderungen eingehalten werden. Eine DIS Übung, also eine Verknüpfung von Simulationen, muss mindestens zwei Simulationen enthalten, die mit einander vernetzt sind. Sollen die vernetzten Simulationen Ereignisse oder Aktivitäten auslösen, ist das DIS Protokoll zu verwenden. Auch hierbei werden verschiedene Anforderungen gestellt. Um festzulegen, welche Simulationen an einer Übung teilnehmen, wird eine eindeutige Übungs-ID an die Simulationen verteilt, die an dieser teilnehmen. Jede Simulation muss durch ein „Simulation Address record“ identifiziert werden. Dieser Record ist im IEEE Standard festgelegt und enthält die „Site Number“ und die „Application Number“. Die „Site Number“ und die „Application Number“ stellen die Zugehörigkeit zu einer Übung dar, dabei dürfen zwei unterschiedliche Simulationen nicht den selben „Simulation Address record“ haben. Stellt eine Simulation eine Entity dar, muss eine Entity ID zugewiesen werden, mit der PDUs ausgegeben werden können die sich auf diese Spezielle Entity beziehen. Hier wird die „Site Number“ und die „Application Number“ im Header angegeben. Werden mehrere Entitys in einer Simulation erstellt, sind die „Site Number“ und die „Application Number“ gleich. Als weitere Anforderung gilt, dass alle Teilnehmer einer Simulation von einander wissen müssen. Das bedeutet, dass die Informationen über Entitys oder Ereignisse an alle Teilnehmer verteilt werden müssen, dies lässt sich als Broadcast realisieren.

3 Beispielsimmulation

Im folgenden Kapitel wird eine Beispiel Simulation beschrieben, die auf dem DIS-Protokoll beruht. Dabei wird eine DIS Library verwendet, die auf dem Standard IEEE-1278.1 beruht. Hierbei wird eine „open source“ Implementation mit dem Namen „Open-DIS“ verwendet, die vom Modeling, Virtual Environments and Simulation (MOVES)-Institute an der Naval Postgraduate School der US Navy in Monterey (Kalifornien) entwickelt wurde. „Open-DIS“ wurde in verschiedenen Sprachen implementiert. Dazu zählen Java, JavaScript, C++, Python und CSharp. Die im Rahmen dieser Arbeit verwendete Implementation wurde in der Programmiersprache C++ geschrieben. Diese Library enthält eine grundlegende Version von DIS. Dazu gehören die verschiedensten PDUs, mathematische Gebilde, die benötigt werden und Funktionen, mit denen PDUs Attribute und Eigenschaften hinzugefügt werden können. [2]

3.1 Ziel der Beispielsimulation

Das Ziel der Simulation war es, erste Erfahrungen mit der Library zu machen und die Möglichkeiten abzugrenzen, die „open source“ bietet. Des Weiteren sollte eine Einschätzung getroffen werden, ob „open source“ in Verbindung mit anderen Bibliotheken eine Plattform liefert, auf der Simulationen einfach zu realisieren sind.

3.2 Beispiel einer ESPDU

Um diese Fragen zu beantworten, wurde ein fiktives Szenario erstellt, in dem zwei Landeinheiten enthalten sind, die sich bewegen und schießen beziehungsweise explodieren können. Zu Beginn einer Simulation wird ein Zwischenspeicher benötigt, der dem DIS Format entspricht. In diesem Speicher werden die zu versendenden Daten gespeichert. Anschließend kann die erste ESPDU erstellt werden. In Quellcode 3.1 ist zu sehen, wie eine ESPDU erstellt wurde. Ihr wurde eine Protokollversion, eine ExerciseID, sowie die in der Entity ID enthaltenen Site, Application und Entity Nummer gegeben. Dabei muss

die Entity ID erst als eigenständiges Feld erstellt und befüllt werden, bevor man einer ESPDU eine Entity ID geben kann.

```
1  DIS::EntityStatePdu unit1;
2  unit1.setProtocolVersion(6);
3  unit1.setExerciseID(0);
4
5  DIS::EntityID unit1_entity_id;
6          unit1_entity_id.setSite( 0 );
7          unit1_entity_id.setApplication( 1 );
8          unit1_entity_id.setEntity( 1 );
9
10 unit1.setEntityID(unit1_entity_id);
```

Listing 3.1: Unit Header

Im Quellcode 3.2 wird dem Entity Typ, wie in Tabelle 2.2.1 zu sehen, bestimmte Attribute zugewiesen. In diesem Fall handelt es sich um einen Leopard 2 A6. Das Zuweisen des Typs erfolgt, wie bei der Entity ID, erst nach dem Erstellen des Typs.

```
1  DIS::EntityType leo2;
2          leo2.setCategory( 1 );
3          leo2.setCountry( 78 );
4          leo2.setDomain( 1 );
5          leo2.setEntityKind( 1 );
6          leo2.setExtra( 0 );
7          leo2.setSpecific( 2 );
8          leo2.setSubcategory( 3 );
9
10 unit1.setEntityType( leo2 );
```

Listing 3.2: Entity Typ

Neben den grundsätzlichen Informationen, die man einer Entity geben kann, können auch Informationen über zusätzliche Teile hinzugefügt werden. Diese Teile können beweglich oder fest angebracht sein. In dem Beispiel 3.3 wird dem Panzer ein Turm und eine Hauptkanone hinzugefügt. Dabei wird durch einen zusätzlichen Parameter dem Turm eine Drehgeschwindigkeit gegeben. Die Hauptkanone, also das Rohr, hat keine entsprechende Neigungsgeschwindigkeit, da diese Geschwindigkeit so groß ist, das sie

keinen entsprechenden Einfluss hat. Jedes Teil, das hinzugefügt werden soll, ist vom Grundaufbau gleich. Über den Wert „ParameterType “ kann die Art des Teils festgelegt werden. Alle möglichen Teile können der „Enumeration“ entnommen werden. Über den Wert „ PartAttachedTo“ kann festgelegt werden, wo das Zusatzteil angebracht ist. Eine „0“ gibt an, dass es direkt mit der Entity verbunden ist. Ein anderer Wert steht für den Index eines anderen Teils, an dem das zu spezifizierende Zusatzteil befestigt ist. Der „ParameterTypeDesignator“ gibt an, ob es sich um ein starres oder ein bewegliches Teil handelt. Um zum Beispiel der Kanone eine Neigung zu geben, wird die entsprechende Gradzahl im „ParameterValue“ angegeben. Alle Teile, dem man an der Entity anbringen will, werden erst in einen Vektor geschrieben, der dann der ESPDU angehängt wird.

```
1   ArticulationParameter turret_azimuth;
2  turret_azimuth.setParameterType( 4096 );
3  turret_azimuth.setPartAttachedTo( 0 );
4  turret_azimuth.setParameterTypeDesignator( 0 );
5  turret_azimuth.setParameterValue(0*PI/180);
6
7  ArticulationParameter turret_azimuth_rate;
8  turret_azimuth_rate.setParameterType( 4096 );
9  turret_azimuth_rate.setPartAttachedTo( 0 );
10 turret_azimuth_rate.setParameterTypeDesignator( 0 );
11 turret_azimuth_rate.setParameterValue( 0 );
12
13  ArticulationParameter turret_gun_elevation;
14 turret_gun_elevation.setParameterType( 4416 );
15 turret_gun_elevation.setPartAttachedTo( 1 );
16 turret_gun_elevation.setParameterTypeDesignator( 0 );
17 turret_gun_elevation.setParameterValue( 0 );
18
19
20
21 std::vector<DIS::ArticulationParameter> params;
22 params.clear();
23 params.resize(3);  // make default number of parameters
24 params[0] = turret_azimuth;
25 params[1] = turret_azimuth_rate;
```

```
26  params[2] = turret_gun_elevation;
27
28
29  unit1.setArticulationParameters(params);
```

Listing 3.3: Zusatz Teile

Um nun der ESPDU einen Standort zu geben, wird auf die gleiche Weise verfahren. Erst muss man einen Standort erstellen, dem man der ESPDU dann anschließend zuweisen kann. Wie ein Standort genau erstellt wird und welche Probleme dabei aufgetreten sind, wird in 3.3 erklärt. Wenn man nun die Entity in eine bestimmte Richtung bewegen will, wird zunächst ein Beschleunigungsvektor benötigt, der dann über eine Funktion der Entity hinzugefügt wird. Auf gleiche Weise wird auch die Orientierung der Entity hinzugefügt. In Quellcode 3.4 ist zu sehen, wie der Entity eine Geschwindigkeit von 1m/s, in X - Richtung und einer Richtung von 180°, vom eigenen Standpunkt aus gegeben wird.

```
1  DIS::Vector3Float veloUnit1;
2          veloUnit1.setX(1);
3          veloUnit1.setY(0);
4          veloUnit1.setZ(0);
5
6  DIS::Orientation headding_unit1;
7          headding_unit1.setTheta(0);
8          headding_unit1.setPsi(180);
9          headding_unit1.setPhi(0);
10
11 unit1.setEntityOrientation(headding_unit1);
12 unit1.setEntityLinearVelocity(veloUnit1);
```

Listing 3.4: Beschleunigungs- und Orientierungsvektor

Um diese ESPDU über ein Netzwerk zu senden, müssen die Daten noch serialisiert werden.

```
1  DIS::DataStream buffer( DIS::BIG );
2
3  unit1.marshal(buffer);
4  /*code for sending buffer */
```

Listing 3.5: Serialisieren der Entity

3.3 Gelöste Probleme

Während die Beispielsimulation erstellt wurde, traten größere und kleinere Probleme auf. In diesem Abschnitt werden nur die größten gelösten Probleme beschrieben. Im Abschnitt 3.4 werden die nicht gelösten Probleme beschrieben.
Das größte Problem das auftrat, war das der Positionsberechnung. Da DIS als Koordinatensystem das WGS84 benutzt musste zunächst eine Möglichkeit gefunden werden Global Positioning System (GPS) Koordinaten, die In Breite- und Längengrad angegeben sind, in einen dreidimensionalen Vektor umzurechnen. Eine mögliche Lösung dafür bietet die „GeographicLib“. Die „GeographicLib“ bietet nicht nur eine Lösung für das Umrechnen von Koordinaten, mit ihrer Hilfe lassen sich auch der Abstand und die Richtung von Koordinaten berechnen. Des Weiteren kann man mit der Library Punke um eine bestimmte Entfernung in eine bestimmte Richtung versetzen. Diese Funktion ermöglicht eine Konstante Positionsänderung, also das Berechnen der Bewegung einer Entity.
Um Funktionen der „GeographicLib“ zu nutzen, muss man zunächst die Referenz-ellipsoiden und den Referenz-geoid erstellen. Dies ist durch einen Funktionsaufruf, wie im Codebeispiel 3.6 zusehen, möglich.

```
1 Geocentric earth(Constants::WGS84_a(), Constants::WGS84_f());
2 Geodesic geod(Constants::WGS84_a(), Constants::WGS84_f());
```

Listing 3.6: Referenz-Ellipsoid und Geoid

Im Beispiel 3.7 sieht man den Aufruf der Funktion die von einer GPS Koordinate in eine geozentrische Koordinate umrechnet und der Funktion die wieder zurückrechnet. Als Input benötigen die Funktionen jeweils das Quellsystem und das Zielsystem in das Umgerechnet werden soll.

```
1 // from GPS to XYZ
2 earth.Forward(Latitude,Longitude,Height_above_geoid,X,Y,Z);
3
4 //from XYZ to GPS
5 earth.Reverse(X,Y,Z,Latitude,Longitude,Height_above_geoid);
```

Listing 3.7: Umrechnung der Koordinaten

In 3.8 wird gezeigt, wie Berechnungen innerhalb eines Koordinatensystems durchgeführt werden. Im als erstes wird eine Position berechnet die in einer bestimmten Entfernung

und Richtung ausgehend von einer Startposition liegt. Die zweite Funktion berechnet den Abstand und die Richtungen der beiden Positionen zueinander.

```
1 //move from position A to a new position
2 geod.Direct(Latitude,Longitude,Direction,Length,
3             Latitude_new,Longitude_new);
4
5 //distance and direction from position A to position B
6 geod.Inverse(Latitude0,Longitude0,Latitude1,Longitude1,
7              Length,Direction0,Direction1);
```

Listing 3.8: Berechnungen innerhalb eines Systems

Alle dargestellten Beispielfunktionen der „GeographicLib“ gehören zu Klassen von Funktionen die in der Dokumentation der Library ausführlicher erklärt werden.
Da für das Senden und Empfangen von DIS Paketen User Datagram Protocol (UDP) verwendet wird ist stellt das versenden der Pakete nicht das Problem dar, jedoch war das verarbeiten von empfangen Paketen ein Problem. Da der Client, der ein Paket in DIS Format bekommt, nicht weiß um was es sich bei diesem Paket handelt, musst eine Möglichkeit gefunden werden dies heraus zu finden. Das Extrahieren der Klasse stellte im Großen und Ganzen nicht das Problem dar, denn im Header einer PDU steht, um was es sich bei dieser PDU handelt. Jedoch musst diese Grundlegende-PDU noch in die entsprechende ESPDU, Fire PDU, Detonation PDU oder in eine Andere umgewandelt werden. Um aus den empfangenen Daten eine PDU zu machen, muss eine „PDU-Factory“ verwendet werden. Diese erstellt aus den empfangen Daten, die als ein Vektor vorliegen, eine PDU.

```
1 DIS::PduFactory pf;
2
3 DIS::Pdu *recvPDU;
4 recvPDU = pf.createPdu(DATASTREAM);
```

Listing 3.9: PDU Factory

Da der Empfänger nicht weiß, wie viele PDUs er bekommt, wurde als Zwischenspeicher eine „map“ verwendet. In der „map“ wird die PDU mit der Hilfe eines KEYs gespeichert. Dieser Key setzt sich aus der Entity ID zusammen. Im Codebeispiel 3.10 ist gezeigt,

wie durch das Abfragen des PDU Typs eine Entsprechende PDU erstellt wird. Das dargestellte Beispiel ist nicht vollständig, jedoch sind die anderen Cases in ähnlicher weise Aufgebaut.

```
1  std::map<int,DIS::EntityStatePdu> unitmap;
2
3  int pdutype = (int)(*recvPDU).getPduType();
4
5  switch (pdutype) {
6  case 0x1:
7          EntityStatePdu *helppdu;
8          helppdu = (DIS::EntityStatePdu *)pf.createPdu(help);
9          key = keymaker((*helppdu).getEntityID());
10         unitmap[key] = *helppdu;
11 break;
12 .
13 .
14 .
```

Listing 3.10: PDU-Typ kovertieren

Neben den genannten gelösten Problemen, traten noch weitere kleine Probleme auf die jedoch keine speziellen Lösungen erstellt wurden.

3.4 Offene Probleme

Im laufe der Erstellung der Beispielsimulation traten auch Probleme auf, für noch keine, oder keine zufriedenstellende Lösung gefunden wurde. Eines dieser Probleme ist die Darstellung der Entitys auf einer Karte.

3.5 Fähigkeiten der Beispielsimulation

3.6 Ausblick

4 Fazit

5 Literaturverzeichnis

[1] Distributed interactive simulation. https://de.wikipedia.org/w/index.php?oldid=175148570. [Online, Stand 16. April 2018].

[2] About open-dis. http://open-dis.org/, 13.10.2017. [Online, Stand 02. Mai 2018].

[3] Joe Brann. Enumeration and bit-encoded values for use with ieee 1278.1-1994, standard for distributed interactive simulation–application protocols. [Stand 5. Mai 2003].

[4] Defense Mapping Agency. Official diagram of the wgs 84 reference frame. https://en.wikipedia.org/wiki/File:WGS_84_reference_frame_(vector_graphic).svg. [Online, Stand 25. April 2018].

[5] JDBE at Ft. Huachuca, Arizona. Dis data dictionary: Detonation pdu. http://faculty.nps.edu/brutzman/vrtp/mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/84.htm, 13.10.2003. [Online, Stand 24. April 2018].

[6] Mark McCall. Distributed interactive simulation (dis) 101. https://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=36474. [Online, Stand 17. April 2018].

[7] Graham Shanks. Enumeration and bit encoded values for dis. https://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=42916. [Stand 17. März 2015].

[8] Simulation Interoperability Standards Organization, Inc. Siso-ref-020-draft. https://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=29302. [Online, Stand 17. April 2018].

[9] SISO Standards Activity Committee of the IEEE Computer Society. Ieee std 1278.1-2012, ieee standard for distributed interactive simulation—application protocols. [Stand 12. Dezember 2012].

[10] UNAVCO. Tutorial: The geoid and receiver measurements. http://www.unavco.org/education/resources/tutorials-and-handouts/tutorials/geoid-gps-receivers.html, 22.03.2018. [Online, Stand 26. April 2018].