Dynamisches Eisenbahn System Modell
Modèle dynamique d'un système ferroviaire
Dynamic model of a railway system

Laax, 20.03.2014

DESM Middleware

Spezifikation v0.17

Autor & Referenzperson
Sebastian Straube

sebastian.straube@desm.ch
+41 (0) 79 48 18 444
http://www.desm.ch

1 Inhaltsverzeichnis

2 Kontaktliste

Name	Verankerung	Kontakt	Aufgaben / Hintergrund DESM
Jürg Suter	DESM Präsident	+41 31 931 3662	Präsident Verein DESM, Middleware Standardisierung
Sebastian Straube	DESM Vorstand IT	+41 79 48 18 444	Vorstand Verein DESM, Middleware Standardisierung, Implementierung für verknüpfte Systemkomponenten
Maximilian Haupt	Privat	mail@maximilianhaupt.com	Implementierung Middleware
Fabian Riesen	Cisco Systems	+41 79 448 4700	Dispatcher Implementierung Re 4/4
Hansjürg Rohrer	Fachhochschule Biel	+41 32 321 63 73	Eigentümer Simulation LOCSIM

3 Dokumentenhistorie

Version	Datum	Name	Änderung
0.12	06.02.2013	Sebastian Straube	Kapitel hinzugefügt: 2 Kontaktliste, 3 Dokumentenhistorie Kapitel entfernt: „Kontaktpersonen“ Kapitel erweitert: 7.7.5.3 Events Simulation Transition (Start, Aufbau und Ende) Beschreibung aktualisiert „setTrainPosition“
0.13	24.02.2013	Sebastian Straube	Kapitel erweitert: 2 Kontaktliste, 7.7.1 ToDo DLL Schnittstelle
0.14	02.04.2013 22.07.2013	Sebastian Straube	Kapitel 7.7.3 erweitert: stw_infoConnectionStatus Kapitel 6 angepasst Kapitel 7.8.3 angepasst
0.15	11.01.2013	Sebastian Straube	Kapitel 2 angepasst Kapitel 7.7.5 angepasst
0.16	19.01.2013	Sebastian Straube	Kapitel 7.7.5 angepasst
0.17	20.03.2014	Sebastian Straube	Kapitel 7.5 angepasst Kapitel 7.7.5 angepasst

4 Ausgangslage

Ausgangsprojekt
Innerhalb der Promotionsarbeit von Jürg Suter wurde ein Forschungslabor aufgebaut, welches zu einem offiziellen Verein mit dem Namen DESM institutionalisiert wird. Das Forschungslabor besteht momentan aus zwei Fahrsimulatoren der Loktypen Re 4/4 und Re 460. Der DESM Simulator des Loktyps Re 460 ist gegenwärtig in der Schweiz der einzige Vollsimulator dieser Art.. Für weitere Details zu der Promotionsarbeit verweise ich auf die Vereinsseite: http://www.desm.ch .

Unterprojekte
Alle Untersuchungen der Forschungsarbeit beziehen sich auf qualitative und quantitative Analysen und der Erarbeitung von neuen Methoden, die im Forschungslabor durchgeführt werden. Dafür ist es nötig ein System aufzubauen, in dem diese Methoden erarbeitet und die Ergebnisse wissenschaftlich analysiert werden können. Sie finden weitere Details auf die Vereinsseite: http://www.desm.ch.

Systemkomponenten Re 4/4
Der Simulator der Re 4/4 beinhaltet verschiedene Systemkomponenten, um dem Lokführer ein möglichst realitätsgetreues Interface und „feeling“ zu bieten. Die Führerstandkabine enthält alle Bedienelemente der Lok vom Typ Re 4/4. Ausserdem wird das Bremssystem mit Druckluft betrieben, so dass realitätsnahe Geräusche und mechanische Bewegungen zu hören sind. Für die Darstellung der Umwelt wird das Simulationsprogramm LOCSIM von der Berner Fachhochschule Biel eingesetzt. Diese Simulationssoftware verfolgt einen videobasierten Ansatz, um die Umwelt für den Lokführer realitätsgetreu abzubilden. Dabei wird für die Simulation eine bearbeitete Videoaufnahme abgespielt. Des Weiteren werden alle benötigten Simulationsvariablen durch die Simulationssoftware berechnet. Für weitere Details zu dem LOCSIM Simulator verweise ich auf die LOCSIM: http://www.locsim.ch

Systemansatz
Für die Forschungsarbeit wird es nötig sein, die Kommunikation zwischen bestimmten Systemelementen zu ermöglichen. Das heisst, es soll nicht nur der Lokführer in die Simulation eingebunden werden, sondern auch der Zugverkehrsleiter in der Betriebszentrale sowie die dazugehörigen Stellwerke auf der simulierten Strecke.

5 Ziele

Das Projekt Middleware verfolgt das langfristige Ziel, verschiedene Systemkomponenten einer Simulation über eine Kommunikationsarchitektur miteinander zu verbinden und somit die Integration einer Betriebszentrale in Bezug auf den Schienenverkehr zu ermöglichen. Dieses Vorhaben wird innerhalb des Vereins DESM umgesetzt. Die einzelnen Komponenten der Architektur sollen möglichst modular aufgebaut sein, um die Anwendung und Integration verschiedener Komponenten kurzfristig zu ermöglichen. Die Kommunikation soll anhand eines anerkannten Industriestandards umgesetzt werden.

6 ToDo DESM Middleware

- ✓ Analyse des bisherigen Simulators Re 420 (Fabian Riesen, Thomas Schneider)
- ✓ Integration LokSim 3D (Fabian Riesen)
- ✓ Integration Locsim FHS Biel (FHS Biel, Fabian Riesen)
- ✓ Aufbereitung der Infrastruktur in der Umgebung von Obermatt auf der Strecke Bern - Luzern (FHS Biel, Jürg Suter)
- ✓ Modellierung der Aussenanlagen in vergangene Epochen (Jürg Suter)
- ✓ Schnittstelle zu Aussenanlagen (Signale) definieren (DLL, TCP, IP)
- ✓ Middleware Netzwerkprotokoll Implementierung (weitere Finktionen – MESSAGE_TYPE_SET_KILOMETER_DIRECTION)
- ✓ Alle Funktionen in das Interface DispatcherPlugin / LocsimPlugin
- ✓ JSON Parser für config file
- ✓ Zugriff auf Signale im Locsim-Gelände
- ✓ Steuerung und Schnittstelle zu Stellwerken (Sebastian Straube, Jürg Suter)
- ✓ Zusammenführung Loksimulation und Stellwerksteuerungen (DESM)

7 LOCSIM - DESM Middleware

Die Middleware Software ist so modular aufgebaut, dass dort verschiedenste Systemkomponenten angeschlossen werden können. Allerdings wird für jede Komponente eine bestimmte Konfiguration für den Datenverkehr benötigt. In diesem Fall umfasst das System die Kommunikationsverbindung von folgenden Systemkomponenten:

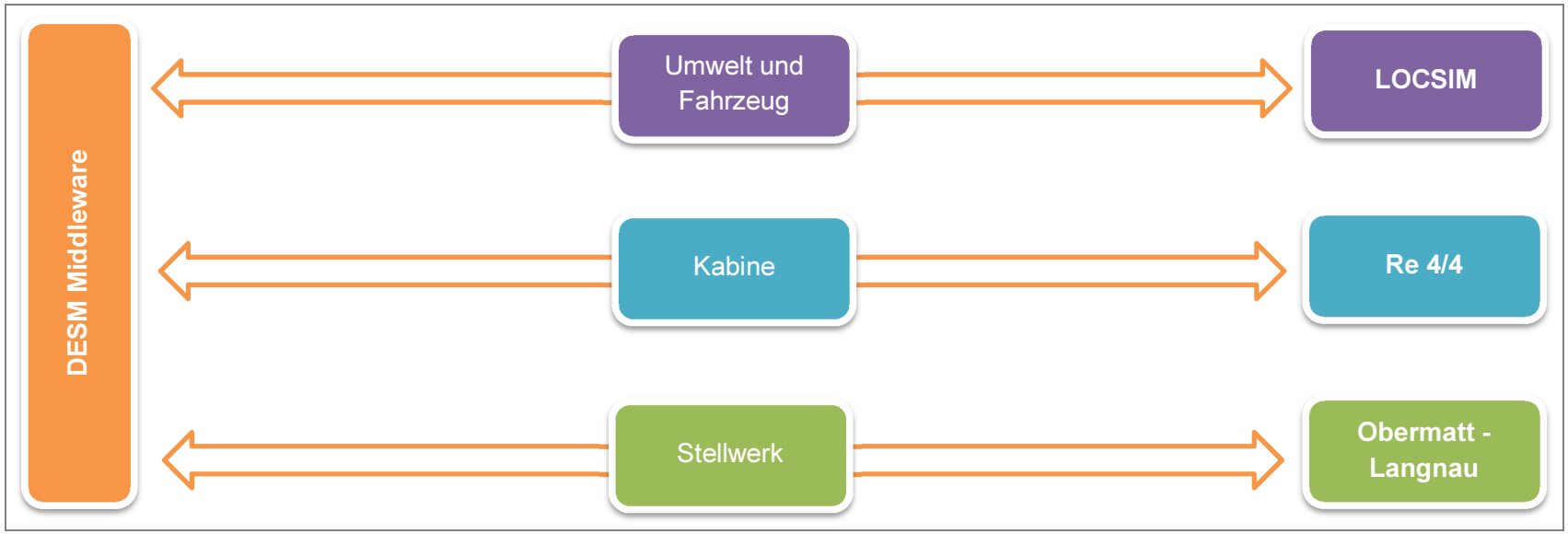

Abb. 1: Wichtigste Elemente der Zusammenführung von Loksimulatoren und Stellwerklogik am Beispiel des Simulators der Re 4/4 und des Stellwerks Obermatt.

7.1 ToDo Ressourcen

Status	Beschreibung
✓	PC und Projektion für den Simulator 420
✓	Software für Loksim 3D
✓	Server für LOCSIM der FHS Biel
✓	Software mit Lizenz für Locsim der FHS Biel und Gelände der Strecke Bern – Luzern

7.2 Systemkommunikation (TCP/IP Protokoll)

Die Verbindung zwischen der Middleware und dem LOCSIM wird per Ethernet hergestellt. Aus den gegebenen Anforderungen wird eine modulare Lösung und damit die Wiederverwendbarkeit der DLL angestrebt. Daher wird in der DLL ein Server und ein Proxy für die Datenübertragung bereitgestellt. Dadurch wird es ermöglicht, auf beiden Systemen mit der gleichen DLL eine Verbindung über das Netzwerk herzustellen.

7.3 Kommunikationsarchitektur

Es stehen verschiedene Techniken zur Verfügung, die Daten über eine Ethernet Verbindung auszutauschen.

Es wurden folgende Technologien bzw. Frameworks evaluiert.

Protokollname	Beschreibung	Vorteil	Nachteil
Winsocket (TCP)	Eine „Winsocket“ Client-Server Übertragung. Das Übertragungsprotokoll soll lediglich ByteStreams empfangen und senden können, weil nur XML Daten transferiert werden sollen und keine RAW Datentypen. Dafür muss ein Magic Packet (Initialisierung) definiert werden.	• schnell • Abstraktion durch XML Struktur	• Integration kompliziert • eigenes Übertragungsprotokoll nötig
HTTP (high level) & Json	verschiedene libraries sind u.A. boostASIO, libHTTP, libEvent, libOV	• High Level Integration • ermöglich Steuerung über	• Overhead gross • keine permanente Verbindung • zwei Webserver

		Webinterface	
Open Sound Protokoll		• library Unterstützung sehr umfangreich	• proprietäre Implementierung
ZeroMQ (0MQ)		• Framework Message Handling • Portierung auf andere Sprachen möglich	•

7.4 Übertragungsprotokoll

Es ist je nach gewählter Technik für die Umsetzung der Ethernet Schnittstelle ein sehr hoher Aufwand für die Entwicklung eines Protokolls nötig. Damit der Aufwand auf jeder Ebene gering gehalten wird, gibt es ein Dateiübertragungsprotokoll. Der Vorteil dieser Lösung ist ein geringerer Synchronisationsaufwand und die Implementierung und Nutzung von später benötigten Techniken. Für die Übertragung wird eine XML strukturierte Datei genutzt. Die Struktur stützt sich auf das railML Datenformat. Dafür wird ein Reader und Writer in die DLL implementiert. Das hat den Vorteil, dass diese Funktion von der Middleware und vom LOCSIM genutzt werden können.

7.5 Synchronisation der Systemkomponenten

Die Datenübertragung von LOCSIM zur Middleware und umgekehrt findet asynchron statt. Die Daten können vom LOCSIM sowie von der Middleware zu einem beliebigen Zeitpunkt übertragen und gelesen werden. Dabei werden die Daten in der DLL zwischengelagert.

Zitat: „Die Topologie einer Gleisanlage wird mit settrack und settrackconnection eindeutig übergeben. Die Signale werden mit setsignal mit Ihrer eindeutigen ID, ihrem Namen und ihrer Lage (Längsposition sowie seitliche Lage) auch eindeutig übergeben. Alle diese Informationen werden beim Aufstarten (vor der Simulation) übergeben.

Sie sollten damit in der Lage sein, einen Gleisplan samt den Signalen zu zeichnen.
Die Balisen und Loops haben bei uns keine ID, sondern sind durch ihre Position und Bezugssignale (setbalise und setloop) definiert.“

7.6 Übertragungsformat (Json)

Es wird für die Datenhaltung und Datenübertragung zwischen den Systemkomponenten ein standardisiertes gültiges Format benötigt. Dadurch wird die Anordnung von späteren Versuchsaufbauten erleichtert. Mit dieser Datenbasis werden Methoden angewandt, um verschiedene Simulatoren, Stellwerktypen und Sicherungsanlagen standardmässig integrieren zu können.

7.7 DLL Spezifikation

Die DLL wurde als „unmanaged“ Code in C++ geschrieben. Zum Laden der DLL wird kein MFC benötigt. Auf die Definition als COM Komponente wurde aus Vereinfachungsgründen verzichtet. Die Daten in der DLL werden in einem Cache gehalten und von dort weitergegeben oder abgeholt.

7.7.1 ToDo DLL Schnittstelle

Status	Beschreibung
✓	DLL Interface von Middleware nach LOCSIM spezifizieren
✓	Interface in unmanaged C++ implementieren
✓	Umsetzung erster Server Tests für den Datenaustauch der Middleware
✓	Testumgebung für Collaboration einrichten (GIT)
✓	Evaluierung Server/Client Architektur
✓	Server und Client in Testumgebung implementieren & einrichten
✓	Allgemeines Interface Format für Datenaustausch finden. {XML, OWL, railML}
✓	Synchronisationsstruktur für Asynchronen Datenaustausch aufbauen
➢	LOCSIM verantwortlichen Herrn Dr. Rohrer informieren und DLL übergeben (Iterativer Prozess)
✓	Threadsafe Implementierung des Server und des Clients in DLL
✓	Implementierung JSON Datei für Konfigurationshandling
✓	Implementierung Error-Handling
✓	Implementierung der Verifikation von Transferierten Nachrichten
~~➢~~	~~XML Dateistruktur für die Infrastrukturdaten in dem Anwendungsfall Signal Obermatt Langnau aufbauen~~
~~✓~~	~~XML Reader und Parser in die DLL implementieren~~

~~✓~~	~~Integration Stellwerklogik anhand einer RuleEngine~~
✓	Test und finale Zusammenführung mit den Komponenten des LOCSIM

7.7.2 Architektur

....

7.7.2.1 Präfix Definition

Es ist für programmatische Problemstellungen u.U. wichtig zu erkennen, in welcher Situation eine Funktion benutzt werden sollte um allen Anforderungen gerecht zu werden. Daher werden hier verschiedene Präfixe für Funktionen definiert, damit bereits vom Namen abgeleitet werden kann in welche Richtung der Kommunikationsweg vollzogen wird und ob z.B. der Cache beeinflusst wird. Die folgende Tabelle.

Präfix	Beschreibung
stw_set	Daten aus LOCSIM zum Stellwerk übertragen
stw_get	Daten aus Stellwerk zum LOCSIM übertragen
stw_on	Funktionsaufruf während bestimmter Events
stw_info	DLL Informationen

7.7.3 Events und Funktionsaufrufe LOCSIM

Die folgende Tabelle zeigt eine Übersicht bei welchem LOCSIM Event bestimmte Funktionen aufgerufen werden sollten. Die Übersicht beschränkt sich auf genau eine Instanz des LOCSIM.

Event	Funktionsaufruf Anzahl	Aktion
Start Programm	einmalig beliebig	stw_onStartProgramm stw_infoVersion stw_infoConnectionStatus
Lade Strecke (Lade neue Strecke)	beliebig beliebig beliebig beliebig beliebig	stw_onLoadStrecke stw_setTrack stw_setTrackConnection stw_setSignal stw_setBalise

	beliebig beliebig beliebig	stw_setIsolierstoss stw_setKilometerDirection stw_setLoop
Start Simulation	einmalig beliebig beliebig beliebig beliebig beliebig beliebig	stw_onStartSimulation stw_getEvents stw_getSignal stw_getBalise stw_getWeiche stw_getLoop stw_setTrainPosition
Stopp Simulation	Einmalig	stw_onStopSimulation
Stopp Programm	Einmalig	stw_onStopProgramm

7.7.4 DLL Error

Error Code	Beschreibung
0	ERROR_OK
1	ERROR_FATAL
2	ERROR_API_MISUSE
3	ERROR_UNKNOWN_ID

7.7.5 DLL Funktionen

Der Zeitpunkt des Zugriffs auf bestimmte DLL Funktionen ist durch gewisse Anwendungsstatus des LOCSIM gegeben. Die Kommunikation findet bidirektional statt.

7.7.5.1 Event DLL laden

Signatur	int stw_onStartProgramm (char* configPath)	
Funktionsbeschreibung	• beim Laden der DLL wird der Netzwerkserver gestartet	
seit Version	0.1	
Parameter	configPath	der Pfad zum gemeinsamen Konfigurationsverzeichnisses wo die Konfigurationsdatei abgelegt ist
Rückgabepointer		
Rückgabewert (OK)	0	

Rückgabewert (ERROR)	1	

Signatur	int stw_onStopProgramm (void)	
Funktionsbeschreibung	• Wenn LOCSIM beendet wird ist die Simulation gestoppt, die DLL wird entladen und der Netzwerkserver heruntergefahren. Es wird sichergestellt, dass die Verbindung zwischen Server und Client ordnungsgemäss getrennt wird.	
seit Version	0.1	
Parameter		
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)		

7.7.5.2 Event DLL ist geladen

Signatur	const int stw_infoVersion(char* versionBuf, int versionBufLen, int* versionStrLen)	
Funktionsbeschreibung	• Version der DLL, zur Behandlung von Versionskonflikten	
seit Version	0.1	
Parameter	int versionBufLen	Die Anzahl der möglichen Zeichen
Rückgabepointer	char* versionBuf int* versionStrLen	Pointer auf die Zeichenkette Pointer für die Länge der Zeichenkette
Rückgabewert (OK)	0	
Rückgabewert (ERROR)		

Signatur	const char* stw_infoConnectionStatus(char* statusBuf, int statusBufLen, int* statusStrLen)	
Funktionsbeschreibung	• gibt den Status der Netzwerkverbindung von Middleware Server und Client zurück	
seit Version	0.1	
Parameter	int statusBufLen	Die Anzahl der möglichen Zeichen
Rückgabepointer	char* statusBuf int* statusStrLen	Pointer auf die Zeichenkette Pointer für die Länge der Zeichenkette
Rückgabewert (OK)	0	
Rückgabewert (ERROR)		

Signatur	const char* stw_infoName(char* nameBuf, int nameBufLen, int* nameStrLen)	
Funktionsbeschreibung	• gibt die Bezeichnung der DLL zurück	
seit Version	0.1	
Parameter	infoNameLen	Die Anzahl der Zeichen vom Rückgabewert als Pointer
Rückgabepointer	char* nameBuf int* nameStrLen	Pointer auf die Zeichenkette Pointer für die Länge der Zeichenkette

Rückgabewert (OK)	0	
Rückgabewert (ERROR)		

Signatur	const char* stw_infoDescription(char* descBuf, int descBufLen, int* descStrLen)	
Funktionsbeschreibung	• gibt die Beschreibung der DLL zurück	
seit Version	0.1	
Parameter	descriptionLen	Die Anzahl der Zeichen vom Rückgabewert als Pointer
Rückgabepointer	char* descBuf int* descStrLen	Pointer auf die Zeichenkette Pointer für die Länge der Zeichenkette
Rückgabewert (OK)	0	
Rückgabewert (ERROR)		

7.7.5.3 Events Simulation Transition (Start, Aufbau und Ende)

Signatur	int stw_onStartSimulation (void)	
Funktionsbeschreibung	• markiert den Start der Simulation • die weitere Datenverarbeitung wird durch mögliche „sets“ ermöglicht • Nach dem Aufruf dieser Funktion ist das dirty flag sämtlicher Signale zu setzen, damit die Grundstellung von LOCSIM erkannt wird	
seit Version	0.1	
Parameter		
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_onStopSimulation (void)	
Funktionsbeschreibung	• markiert das Ende der Simulation • erwartet danach „get“ Funktionen oder die „setTrainPosition“ Funktion	
seit Version	0.1	
Parameter		
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_onLoadStrecke (void)
Funktionsbeschreibung	• beim erneuten Laden einer Strecke werden alle Transferdaten in der Stellwerk DLL gelöscht

seit Version	0.1	
Attribute		
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_setTrack (int gleisId, double von, double bis, double abstand, char* name, int nameLen)	
Funktionsbeschreibung	• Definition von zu Hauptgleis parallelen Gleisabschnitten • das Hauptgleis ist die Standardfahrstrasse der Simulation	
seit Version	0.1	
Parameter	gleisId von bis abstand name nameLen	locsim-interne ID als Meterangabe als Meterangabe Abstand von Hauptgleis(in Meter) Name vom Track Die Anzahl der Zeichen vom Attribut name
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_getTrack (int gleisId, double* von, double* bis, double* abstand, char* nameBuf, int nameBufLen, int* nameStrLen)	
Funktionsbeschreibung	• Definition von zu Hauptgleis parallelen Gleisabschnitten • das Hauptgleis ist die Standardfahrstrasse der Simulation	
seit Version	0.1	
Parameter	gleisId von bis abstand nameBuf nameBufLen nameStrLen	locsim-interne ID als Meterangabe als Meterangabe Abstand von Hauptgleis(in Meter) positiv (rechts), negativ (links), Gleisnummer gemäss Sicherungsanlage, z.B. A1 (kann auch leer sein) Buffer Länge von nameBuf die Anzahl der Zeichen von der Variable nameBuf (exkl. Terminator String)
Rückgabepointer	double* von double* bis double *abstand char* nameBuf int* nameStrLen	
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_setTrackConnection (int gleis1Id, int gleis2Id, double von, double bis, char* name, int nameLen, int weiche1Id, int weiche2Id)	
Funktionsbeschreibung	• definiert die Verbindungen zwischen parallelen Gleisabschnitten • weiche1Id, weiche2Id = ID der Weiche • ID ist eine eindeutige LOCSIM-interne Nummerierung • wenn ID=0 dann keine Weiche (wenn z.B. connection2 in gleis2 übergeht) • pro Längsposition dürfen max. 20 verschiedene Gleise definiert sein (ID= 1…20)	
seit Version	0.1	
Parameter	gleis1Id gleis2Id von bis name nameLen weiche1Id weiche2Id	Eindeutige id vom Gleis1 Eindeutige id vom Gleis2 Positionsangabe in Meter Positionsangabe in Meter Name vom Gleis Die Anzahl der Zeichen vom Attribut name ID der ersten Weiche ID der zweiten Weiche
Rückgabepointer	char* name	
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_getTrackConnection (int gleis1Id, int gleis2Id, double* von, double* bis, char* nameBuf, int nameBufLen, int* nameStrLen, int* weiche1Id, int* weiche2Id)	
Funktionsbeschreibung	• definiert die Verbindungen zwischen parallelen Gleisabschnitten • weiche1Id, weiche2Id = ID der Weiche • ID ist eine eindeutige LOCSIM-interne Nummerierung • pro Längsposition dürfen max. 20 verschiedene Gleise definiert sein (ID= 1…20)	
seit Version	0.1	
Parameter	gleis1Id gleis2Id von bis nameBuf nameBufLen nameStrLen weiche1Id weiche2Id	Eindeutige id vom Gleis1 Eindeutige id vom Gleis2 Positionsangabe in metera Positionsangabe in metera Name der Gleisverbindung Grösse des Buffers zur Übertragung Die Anzahl der Zeichen vom Attribut name ID der ersten Weiche ID der zweiten Weiche
Rückgabepointer	von bis nameBuf nameStrLen weiche1Id weiche2Id	

Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_setSignal (int signalId, int gleisId, double position, int typ, double hoehe, double distanz, char* name, int nameLen, int stellung)	
Funktionsbeschreibung	• wird pro Signal aufgerufen	
seit Version	0.1	
Parameter	signalId gleisId position typ hoehe distanz name stellung	eindeutige Nummer (willkürlich) interne Gleisnummer Geoposition des Signals der Typ des Signals die Höhe des Signals die Distanz des Signals zum Gleis name des Signals positiv dann Richtung +1, negativ dann Richtung -1
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_setBalise (int baliseId, int gleisId, double position, int stellung, int beeinflussendeSignalId1, int beeinflussendeSignalId2)	
Funktionsbeschreibung	• definiert eine Balise: auf bestimmten Gleis, an bestimmter Position	
seit Version	0.1	
Parameter	baliseId gleisId position stellung beeinflussendeSignalId1 beeinflussendeSignalId2	eindeutige Balise ID eindeutige Gleis ID Position der Balise Stellung positiv dann Richtung +1 sonst negativ dann Richtung -1 Beeinflussende Signal 1 Beeinflussende Signal 2
Rückgabewert (Attribut)		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_setLoop (int baliseId, int gleisId, double von, double bis, int stellung, int beeinflussendeSignalId1, int beeinflussendeSignalId2)	
Funktionsbeschreibung	• Wozu wird diese funktion benötigt?	
seit Version	0.13	
Parameter	baliseId gleisId von	eindeutige Balise ID eindeutige Gleis ID Position von

	bis stellung beeinflussendeSignalId1 beeinflussendeSignalId2	Position bis Stellung positiv dann Richtung +1 sonst negativ dann Richtung -1 Beeinflussende Signal 1 Beeinflussende Signal 2
Rückgabewert (Attribut)		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_setIsolierstoss (int isolierstossId, int gleisId, double position)	
Funktionsbeschreibung	• Ist in den locsim-Streckendaten bis jetzt nicht drin, könnte aber hinzugefügt werden	
seit Version	0.1	
Parameter	isolierstossId gleisId position	Eindeutige Isolierstoss id Eindeutige Gleis id Position vom Isolierstoss
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_getIsolierstoss (int isolierstossId, int* gleisId, double* position)	
Funktionsbeschreibung	• Ist in den locsim-Streckendaten bis jetzt nicht drin, könnte aber hinzugefügt werden	
seit Version	0.15	
Parameter	isolierstossId gleisId position	Eindeutige Isolierstoss id Eindeutige Gleis id Position vom Isolierstoss
Rückgabepointer	Int* gleisId double* position	
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_setKilometerDirection (int richtung)	
Funktionsbeschreibung	• Gibt an ob die Kilometer inkrementiert oder dekrementiert werden	
seit Version	0.13	
Parameter	richtung	bei einem positiven Wert wird inkrementiert (+1) bei einem negativen Wert wird dekrementiert (-1)
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_getKilometerDirection (int* richtung)

Funktionsbeschreibung	• Gibt an ob die Kilometer inkrementiert oder dekrementiert werden	
seit Version	0.15	
Parameter	richtung	bei einem positiven Wert wird inkrementiert (+1) bei einem negativen Wert wird dekrementiert (-1)
Rückgabepointer	int* richtung	
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

7.7.5.4 Event Simulation gestartet

Signatur	int stw_getEvents(int* anzahlEvents, int** typeList, int*** paramList)	
Funktionsbeschreibung	LOCSIM fragt DLL, welche events vom Stellwerk ausgelöst wurden Beispiel aufruf: anzahlEvents = 3; typeList = {1,2}; paramList = {{54, 62,12}, {5, 23}}; Wenn bekannt ist das 3 Events gesammelt wurden, wie im obigen Beispiel. Dann können danach die korrespondierenden Funktionen aufgerufen werden. Die Art der Events wird in der typeList fetgehalten z.B. typeList[0] = 1 = Signal Event, dann kann die Id vom Signal anhand der Parameter Liste abgerufen werden, z.B typeList[1] = 2 = ein Balise Event (stw_getBalise), die dazugehörige paramList[1][0] = 5 = baliseId, dann kann die Balise mit stw_getBalise(baliseId) abgerufen werden.	
seit Version	0.1	
Parameter	anzahlEvents typeList paramList	Anzahl Events (Arraygrösse): 0 = keine Events vorhanden types = 1 (Signal Event), 2 (Balise Event), 3 (Weiche Event) Eine Liste mit IDs zu der korrespondierenden typeList, siehe Funktionsbeschreibung
Rückgabepointer	Int* anzahlEvents Int** typeList Int*** paramList	Beinhaltet die Anzahl der Events (Buffer länge)
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	1 2	

Signatur	int stw_getSignal (int signalId, int* gleisId, double* position, int* typ, double* hoehe, double* distanz, char** nameBuf, int nameBufLen, int* nameStrLen, int* stellung)	
Funktionsbeschreibung	• gibt die Stellung eines bestimmten Signals zurück	
seit Version	0.1	
Parameter	signalId gleisId position typ hoehe distanz nameBuf	ID des Signals

	nameBufLen nameStrLen stellung	
Rückgabepointer	Int* gleisId double* position int* typ double* hoehe double* distanz char* nameBuf int* nameStrLen int* stellung	
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_setWeiche (int weicheId, int gleisId)	
Funktionsbeschreibung	• gibt die Stellung einer bestimmten Weiche zurück	
seit Version	0.1	
Parameter	weicheId gleisId	ID gemäss set_trackConnection Gleisnummer der Stellung (im stumpfen Bereich)
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_getWeiche (int weicheId, int* gleisId)	
Funktionsbeschreibung	• gibt die Stellung einer bestimmten Weiche zurück	
seit Version	0.1	
Parameter	weicheId gleisId	ID gemäss set_trackConnection Gleisnummer der Stellung (im stumpfen Bereich)
Rückgabepointer	int* gleisId	ist rückgabewert, als Integer Pointer
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_setTrainPosition (int trainTyp, int direction, double* positionList, int positionListLen, int* gleisList, int gleisListLen)
Funktionsbeschreibung	• Übergibt pos1 – gleis1 – pos2 – gleis2 – pos3 - … des Zuges • Zug an DLL wenn die Zugspitze oder der Zugschluss einen Isolierstoss überfährt • Was bedeutet in diesem Fall die Position? • Anzahl pos = Anzahl gleis +1
seit Version	0.1

Parameter	trainTyp direction positionList poitionListLen gleisList gleisListLen	0=simulierter Zug, 1 und weitere=andere Züge 1=vorwärts, -1=rückwärts Position „von, bis“ als Objekt von „setTrack“ Gleisnummern „gleisId“ als Objekt von „setTrack“ Anzahl der Elemente in gleisList
Rückgabepointer		
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_getTrainPosition (int trainTyp, int* direction, double** positionList, int* positionListLen, int** gleisList, int* gleisListLen)	
Funktionsbeschreibung	• Übergibt pos1 – gleis1 – pos2 – gleis2 – pos3 - … des Zuges • Zug an DLL wenn die Zugspitze oder der Zugschluss einen Isolierstoss überfährt • Was bedeutet in diesem Fall die Position? • Anzahl pos = Anzahl gleis +1	
seit Version	0.1	
Parameter	trainTyp direction positionList positionListLen gleisList gleisListLen	0=simulierter Zug, 1 und weitere=andere Züge 1=vorwärts, -1=rückwärts Position „von, bis“ als Objekt von „setTrack“ Gleisnummern „gleisId“ als Objekt von „setTrack“
Rückgabepointer	int* direction double** positionList int* positionListLen int** gleisList int* gleisListLen	
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_getLoop int baliseId, int* gleisId, double* von, double* bis, int* stellung, int* beeinflussendeSignalId1, int* beeinflussendeSignalId2)	
Funktionsbeschreibung	• Wozu wird diese funktion benötigt?	
seit Version	0.15	
Parameter	baliseId gleisId von bis stellung beeinflussendeSignalId1	eindeutige Balise ID eindeutige Gleis ID Position von Position bis Stellung positiv dann Richtung +1 sonst negativ dann Richtung -1 Beeinflussende Signal 1

	beeinflussendeSignalId2	Beeinflussende Signal 2
Rückgabepointer	int* gleisId double* von double* bis int* stellung int* beeinflussendeSignalId1 int* beeinflussendeSignalId2	
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

Signatur	int stw_getBalise (int baliseId, int* gleisId, double* position, int* stellung, int* beeinflussendeSignalId1, int* beeinflussendeSignalId2)	
Funktionsbeschreibung	gibt eine Stellung und das Protokoll einer bestimmten Balise zurück	
seit Version	0.1	
Parameter	baliseId gleisId position stellung beeinflussendeSignalId1 beeinflussendeSignalId2	eindeutige Balise ID eindeutige Gleis ID Position der Balise Stellung positiv dann Richtung +1 sonst negativ dann Richtung -1 Beeinflussende Signal 1 Beeinflussende Signal 2
Rückgabepointer	int* gleisId double* position int* stellung int* beeinflussendeSignalId1 int* beeinflussendeSignalId2	
Rückgabewert (OK)	0	
Rückgabewert (ERROR)	2	

7.8 Konfigurationsdatei

Die Library beinhaltet verschiedene Komponenten. Für die Einstellung dieser Komponenten, wird eine Konfigurationsdatei benötigt. Die Eigenschaften der Konfiguration werden wie folgt festgelegt.

Name: desm-middleware_config.xml

7.8.1 Validierung (XSD)

7.8.2 Struktur

7.8.3 Properties & Values

Connection Timeout
Mode {Client, Server}
Host (optional im Server mode)
Port

8 Fehlerbeschreibung

Wenn die DLL in der DLL ein Fehler abgefangen wird, ….

8.1 Logfile

8.2 Syntax Beschreibung

9 Installationsprozedur

9.1 .NET Framework

9.2 Middleware

9.3 DLL

9.3.1 Konfiguration