Hochdynamische Graphstruktur für Situations- und Maßnahmenobjekte des Verkehrsmanagement- und Informationssystems TRAPH

Ein Projekt finanziert im Rahmen der Verkehrsinfrastrukturforschung 2019 (**VIF 2019**)

März 2021


Abbildungen: Überkopfanzeige und ASFINAG Verkehrsmanagement (C) ASFiNAG

Impressum:

Herausgeber und Programmverantwortung:
Bundesministerium für Klimaschutz
Abteilung Mobilitäts- und Verkehrstechnologien
Radetzkystraße 2
1030 Wien

ÖBB-Infrastruktur AG
Praterstern 3
1020 Wien

Autobahnen- und Schnellstraßen-Finanzierungs-Aktiengesellschaft
Rotenturmstraße 5-9
1010 Wien

Für den Inhalt verantwortlich:
RISC Software GmbH
Softwarepark 32a
4232 Hagenberg

Programmmanagement:
Österreichische Forschungsförderungsgesellschaft mbH
Thematische Programme
Sensengasse 1
1090 Wien

Hochdynamische Graphstruktur für Situations- und Maßnahmenobjekte des Verkehrsmanagement- und Informationssystems TRAPH

Ein Projekt finanziert im Rahmen der

Verkehrsinfrastrukturforschung

(VIF2019)

Autor*innen:

DI Paul HEINZLREITER

Mag. Stefanie KRITZINGER, PhD

Auftraggeber:

Bundesministerium für Klimaschutz

ÖBB-Infrastruktur AG

Autobahnen- und Schnellstraßen-Finanzierungs-Aktiengesellschaft

Auftragnehmer:

RISC Software GmbH

KURZFASSUNG

Als Autofahrer*in auf Österreichs Autobahnen und Schnellstraßen kennt man die Situation: Warum zeigt der Überkopfanzeiger gerade jetzt eine Geschwindigkeitsbeschränkung auf 60 km/h an? Um solche Anfragen auch für einen länger zurückliegenden Zeitpunkt zu beantworten – zum Beispiel bei Anfragen der Exekutive – soll eine effiziente Bearbeitung solcher Fragestellungen seitens der ASFiNAG durch ein modernes Datenhaltungssystem unterstützt werden.

Schaltungen von Überkopfanzeigern müssen dazu nachvollziehbar sein. Dafür werden die Zusammenhänge von den Sensordaten, welche an der Strecke aufgenommen werden, bis zu den Schaltentscheidungen der Überkopfanzeiger in einer Datenbank abgelegt. Beispiele für Situationen, welche Schaltungen auslösen können, sind die Erkennung eines Stauendes oder von Nässe auf der Fahrbahn, welche als Maßnahme zu einer dynamischen Beschränkung der zulässigen Geschwindigkeit führen. Zentrale Fragen sind auch welche Schaltungen auf bestimmten Abschnitten der Autobahnen am häufigsten vorkommen oder welche Situationen und Maßnahmen führen zu bestimmten Anzeigen. Ein weiterer Anwendungsfall ist die Auswertung der örtlichen Verteilung der Schaltungen. Des Weiteren ist das Verständnis der Zusammenhänge zwischen den Situationen und Maßnahmen für die Steuerung und Parametrierung des Verkehrsmanagement- und Informationssystems 2.0 (VMIS) essenziell. Nur durch die Analyse dieser Daten, hat die ASFiNAG das Wissen, ob VMIS 2.0 korrekt funktioniert und wie die Automatikprogramme zu parametrisieren sind.

1. EINLEITUNG

1.1 Problemstellung

Die ASFiNAG bearbeitet mit Projektpartnern das Projekt VMIS2, welches zum Ziel hat, die Verkehrsbeeinflussungsanlagen an den österreichischen Autobahnen weiterzuentwickeln. Ein Aspekt dabei ist die Vereinheitlichung der Betriebsmittel in Tunnel und am Freiland. Zusätzlich soll die aktuelle punktuelle Steuerung durch ein System basierend auf parametrierbaren Wirkungsbereichen ersetzt werden.

Parallel zu der Entwicklung dieses Projekts verfolgte die ASFiNAG im Rahmen des Projekts TRAPH das Ziel ein prototypisches System zur Verbesserung der Nachvollziehbarkeit von Schaltentscheidungen der Verkehrsbeeinflussungsanlagen zu entwickeln.

Grundsätzlich werden in VMIS2 die Schaltungen der Verkehrsbeeinflussungsanlagen aus Sensorwerten abgeleitet. Diese werden in einem mehrstufigen Prozess synchronisiert und plausibilisiert bevor Kennwerte zur Verkehrslagebestimmung berechnet werden. Aus diesen Daten werden Situationsobjekte abgeleitet, welche die Basis für Maßnahmenobjekte und die daraus folgenden Schaltungen für die Wirkungsbereiche bilden. Ein Beispiel dafür ist die Anzeige von Staubildung auf den Überkopfanzeigern.

Ziel des Projekts TRAPH war nun basierend auf den Daten, die im Laufe des Ableitungsprozesses anfallen, eine Graphdatenbank aufzubauen, um die Nachvollziehbarkeit der Schaltungen zu verbessern. Hierbei soll die Ableitung der Daten von den Sensorwerten bis zu den Anzeigen auf den Überkopfanzeigen in der Graphdatenbank abgebildet werden.

1.2 Projektziele

Das Projekt TRAPH hatte zum Ziel den Auftraggeber ASFiNAG bei der Auswahl sowie dem Aufbau einer Graphdatenbank zu unterstützen und diese prototypisch umzusetzen. Der Anwendungsbereich der Graphdaten war hierbei die Verbesserung der Nachvollziehbarkeit der Schaltungen auf den Überkopfanzeigern auf den österreichischen Autobahnen.

Folgende Ziele wurden mit dem Auftraggeber akkordiert:

- Auswahl einer Graphdatenbank-Technologie basierend auf den technischen Anforderungen des Auftraggebers
- Einarbeitung in das bestehende VMIS2-Datenmodell und Erstellung eines geeigneten Datenmodells für die Graphdatenbank
- Prototypische Installation und Konfiguration der Graphdatenbank
- Import der relevanten VMIS2-Daten aus verschiedenen Datenquellen in die Graphdatenbank sowie Speicherung gemäß dem entworfenen Graph-Datenmodell
- Beschleunigung des Datenimports sowie der Abfragen auf die Graphdatenbank
- Die Graphdatenbank soll im Endausbau Situations- und Maßnahmenobjekte, welche österreichweit im 15 Sekundentakt gebildet werden, abgelegen können.
- Das zu entwerfende Datenmodell für die Graphdatenbank soll auf der einen Seite einen hochdynamischen Graphen abbilden und auf der anderen Seite sehr viele Zusammenhänge aus Knoten und Kanten verwalten zu können.
- Die Abbildung einstellbarer zeitabhängige Wirkungsbereiche im Graphen soll möglich sein.

Um diese Ziele zu erreichen, wurde das Projekt bereits im Rahmen der Beantragung in entsprechende Arbeitspakete aufgeteilt, die mit ihren Ergebnissen auch die Struktur des Endberichts vorgeben.

Im ersten Schritt wurde dabei in Zusammenarbeit mit der ASFiNAG die Projektanforderungen abgeklärt, die sich durch den Fortschritt, des Rahmenprojekts VMIS2 im Vergleich zur Ausschreibung zwischen der Ausschreibung und dem Projektbeginn leicht verändert hatten. Zum Beispiel war die Speicherung der Binärobjekte beziehungsweise ihre Verknüpfung mit der Graphstruktur, kein zentrales Anliegen des Projekts TRAPH mehr, da dieses Problem bereits im Rahmenprojekt gelöst wurde. Andererseits konnten im Rahmen des Projekts durch eine geeignete Auswahl der Speichermethodik die Sensorwerte direkt mit in die Graphdatenbank übernommen werden, was eine integrierte Abfrage aller relevanten Daten aus der Graphdatenbank ermöglicht. Zusätzlich ist damit der Zugriff auf separat abgespeicherte Binärdaten nicht mehr notwendig, um die gewünschten Abfragen durchführen zu können.

1.3 Stand der Forschung

Der für das Forschungsprojekt TRAPH relevante State-of-the-Art (SotA) umfasst im Wesentlichen Themen der NoSQL- und Graphdatenbanken, Datenmodellentwurf sowie effiziente Datenaufbereitung sowie Datenabfrage. Der Einsatz von Graphdatenbanken sowie Big Data und NoSQL Technologien - wie zum Beispiel Hadoop [white15][1] und verwandte Projekte - zählt seit einigen Jahren zum SotA. TRAPH hat sich stark auf bereits existierende Ergebnisse gestützt, vor allem um eine geeignete Technologieauswahl für den angestrebten Prototyp zu treffen.

In der ersten Projektphase wurde basierend auf den Anforderungen der ASFiNAG eine geeignete Graphdatenbanktechnologie ausgewählt. Im Zuge dessen wurde eine Erhebung des Stands der Entwicklung im Bereich der Graphdatenbanken durchgeführt. Dafür wurde einerseits eine Literaturrecherche durchgeführt sowie andererseits verschiedene Graphdatenbanksysteme evaluiert. Wie im Antrag beschrieben wurden die Anforderungen und die Kriterien zur Technologieauswahl gemeinsam mit dem Auftraggeber festgelegt. Daraus ergab sich das Ziel eine skalierbare Graphdatenbanktechnologie auszuwählen, bei der einerseits keine hohen Lizenzkosten anfallen und die andererseits gut in das geplante Hadoop Ökosystem des Auftraggebers integrierbar ist. Zusätzlich wurden durch die auftraggeberseitige Festlegung auf eine Graphdatenbank Datenbanksysteme in die engere Auswahl genommen, welche mit einem reinen Property-Graph Modell arbeiten. Neo4J[2] als verbreitetste Graphdatenbanklösung wurde einerseits wegen der höheren Lizenzkosten für einen Clusterbetrieb sowie der Anforderung einer guten Integrierbarkeit mit einer Hadoop-Umgebung ausgeschieden.

Aufgrund dieser Kriterien fiel die Wahl auf die Open-Source Graphdatenbank JanusGraph[3], die alle oben genannten Kriterien erfüllt. Um diese Auswahl abzusichern, wurde eine eingehende technische Analyse der Datenbank durchgeführt.

In den Bereichen Systemdesign und Umsetzung des Prototyps finden sich die Aspekte, welche über den SotA hinausgehen. Dies liegt vor allem daran, dass aufgrund der Anforderungen an die Leistungsparameter die zu entwerfende Datenarchitektur speziell auf

[1] http://hadoop.apache.org/
[2] https://www.neo4j.com/
[3] https://janusgraph.org/

den Anwendungsfall zugeschnitten werden muss und es sich daher um keine reine Umsetzung eines bekannten Konzepts handelt.

Die zentralen Forschungsaspekte des Projektes sind durch die Rahmenbedingung gegeben, dass neue Verkehrsdaten schnell in das Graphmodell einfließen müssen sowie die Graphdatenstruktur durch Veränderung von zeitabhängigen Wirkungsbereichen modifiziert werden kann. Dafür muss das zugrundeliegende Datenhaltungssystem schnelle Datenspeicherung und spezifische vorgegebene Abfragen ermöglichen und die Datenaufbereitungsalgorithmen entsprechend optimiert werden. Selbiges gilt für die Unterstützung von schnellen Modifikationen der Graphdatenstruktur.

2. METHODIK

Gemäß der im Antrag spezifizierten Arbeitspakete wurden zuerst in Zusammenarbeit mit der ASFINAG die Anforderungen an das System erarbeitet und auf deren Basis eine Technologieentscheidung für die Graphdatenbank getroffen. Da die horizontale Skalierbarkeit aufgrund der erwarteten Datenmenge eine zentrale Anforderung darstellte und ein Hadoop-Cluster als Systemumgebung geplant ist, fiel die Wahl der Graphdatentechnologie auf JanusGraph.

Im nächsten Schritt folgte die Einarbeitung in das VMIS2 Datenmodell sowie die Übernahme der Daten aus einem Kafka-Broker[4]. Parallel zur Entwicklung des Datenmodells für die Graphdatenbank wurde die Datenextraktion aus dem Kafka-Broker und der SQL-Datenbank umgesetzt. Dies schließt auch die Deserialisierung der Daten aus dem Google Protocol Buffers[5] Format ein.

Danach wurden die Daten gemäß dem entworfenen Datenmodell in der JanusGraph Datenbank gespeichert. Abschließend wurden durch das Anlegen von Indizes sowohl der Datenimport als auch die Abfragen gegen die Graphdatenbank beschleunigt. Mit diesen Tätigkeiten wurden einerseits die im Antrag formulierten Aufgaben abgearbeitet und andererseits die Erwartungen des Auftraggebers ASFINAG an das Projektergebnis erfüllt. In den folgenden Abschnitten werden die Tätigkeiten genauer dargelegt.

2.1 Analyse und Technologieauswahl

Die Tätigkeiten im Rahmen der Technologieanalyse adressierten im Wesentlichen das Projektziel der Auswahl einer Graphdatenbank-Technologie basierend auf den technischen Anforderungen des Auftraggebers. Basierend auf diesen wurden die Kriterien zur Technologieauswahl gemeinsam mit dem Auftraggeber festgelegt. Neben den funktionalen Anforderungen der Graphdatenbank selbst umfasst dies auf eine geeignete Integration der Graphdatenbank in die geplante gesamte Systemumgebung von VMIS2.

[4] https://kafka.apache.org/
[5] https://developers.google.com/protocol-buffers

2.1.1 Systemumgebung

Bei der Betrachtung der Systemumgebung des Projekts TRAPH muss zwischen der prototypischen Umgebung während der Projektlaufzeit sowie der Produktivumgebung im Endausbau des Rahmenprojekts VMIS2 unterschieden werden. Die Auswahl der eingesetzten Graphdatenbanktechnologie erfolgte nach den Anforderungen des Produktivbetriebs, vor allem im Hinblick auf die horizontale Skalierbarkeit der Datenhaltungslösung.

Die Systemumgebung für das geplante System VMIS2 der ASFiNAG stellt sich im Endausbau nach Projektabschluss von VMIS2 so dar, dass alle Daten, die durch Algorithmen in VMIS2 auf der Basis von Sensordaten errechnet werden, über einen Kafka-Broker zur Verfügung gestellt werden. Statische Daten in Bezug auf die Anlagen an den Autobahnen werden durch eine gRPC-Schnittstelle[6] zur Verfügung gestellt. Alle Quelldaten – sowohl die statischen Daten aus der gRPC-Schnitstelle als auch die dynamischen Daten aus dem Kafka-Broker werden im Google Protocol Buffers Format zur Verfügung gestellt. Die statischen Daten umfassen dabei unter anderem die vorhandenen Sensoren sowie Überkopfanzeiger mit ihren geographischen Positionen, sowie die vorhandenen Wirkungsbereiche, die zur logischen Einteilung der Streckenabschnitte auf den Autobahnen dienen. Die dynamischen Daten wiederum beinhalten neben den Sensordaten die daraus algorithmisch abgeleiteten Situations- und Maßnahmenobjekte sowie die Schaltungen der Verkehrsbeeinflussungsobjekte.

Alle Daten, die im Rahmen von VMIS2 anfallen, werden auf einem Hadoop-Cluster in der NoSQL Datenbank HBase[7] gespeichert werden. Damit war die in der Projektausschreibung erwähnte architektonische Entscheidung über die Speicherung der Binärdaten bereits außerhalb des Projekts TRAPH getroffen.

Während der Durchführung des Forschungsprojektes TRAPH befand sich die geplante Systemumgebung für den produktiven Einsatz der Graphdatenbank noch im Aufbau, im Besonderen stand der Hadoop-Cluster noch nicht zur Verfügung. Daher musste RISC als

[6] https://grpc.io/
[7] https://hbase.apache.org/

Auftragnehmer ein prototypisches Setup der Graphdatenbank und im Besonderen des Storage-Backends im Rahmen des Projekts vornehmen. Als weiterer Unterschied zu der geplanten Produktivumgebung konnten im Rahmen der Prototypentwicklung von TRAPH die dynamischen Daten neben dem Kafka-Broker auch von einem Microsoft SQL-Server bezogen werden.

Für die Entwicklung des Graphdatenbank-Prototypen wurden seitens der ASFiNAG drei Centos-Basierte Linux-Maschinen zur Verfügung gestellt, von denen aus die oben genannten Datenquellen wie gRPC, Kafka und der SQL-Server erreichbar waren.

2.1.2 Durchführung der Technologieauswahl

Die im Antrag spezifizierten Aufgaben im Rahmen der Technologieauswahl waren wie folgt:

- Abklärung der technischen Anforderungen: Diese Aufgabe wurde in enger Zusammenarbeit mit dem Auftraggeber erfüllt. Zuerst wurde seitens des Auftraggebers die geplante Systemumgebung sowie die Anforderungen an die Graphdatenbank spezifiziert, um die in der Ausschreibung angegebenen Projektziele zu konkretisieren. Genauso wurde ein Überblick über die zu importierenden Daten gegeben. Auf Basis dieser Information wurde von RISC eine Technologieevaluierung zur Auswahl der Graphdatenbank durchgeführt.
- Kriterien der Technologieauswahl: Im Rahmen dieser Aufgabe wurden von RISC gemeinsam mit dem Auftraggeber relevante Kriterien für die Technologieauswahl der Graphdatenbank erarbeitet. Diese umfassen unter anderem die Skalierbarkeit der Graphdatenbank sowie deren Integrierbarkeit in die geplante Systemumgebung. Überlegungen zu den Lizenzkosten der eingesetzten Datenbanklösung, im Besonderen im Hinblick auf die Skalierbarkeit der Datenbank wurden in die Technologieentscheidung ebenfalls einbezogen.
- Bewertung von Graphdatenbanken und Big-Data-Systemen: Bei dieser Aufgabe wurden die zur Auswahl stehenden Graphdatenbanken nach den zuvor aufgestellten Kriterien bewertet und basierend auf dem Ergebnis die geeignetste Lösung ausgewählt.

Aufgrund der abzuspeichernden Datenmenge von bis zu 100 TB wurde der Einsatz rein SQL-basierter Datenbanktechnologie unter anderem aus Lizenzkostengründen ausgeschlossen. Damit bewegt man sich im NoSQL Umfeld, womit der Frage des Datenmodells und der Auswahl der richtigen Datenbanktechnologie große Bedeutung zukommt. Dies gilt vor allem aufgrund der vorgegebenen zeitlichen Einschränkungen für die Datenverarbeitung.

Ein weiteres Kriterium in der Technologieevaluierung ist durch die Schreibgeschwindigkeit der Datenhaltungslösung gegeben, die ein zentrales Kriterium für die zeitgerechte Übernahme der Daten darstellt, welches in die Technologieauswahl sowie das Systemdesign einfloss. Den limitierenden Faktor stellt hierbei die Anforderung das, dass im Endausbau alle 15 Minuten neue Daten für ganz Österreich in die Datenbank übernommen werden müssen. Auch wenn keine harten Echtzeitanforderungen vorgegeben wurden, ist es doch notwendig, die Daten zeitnahe zu verarbeiten, da es sonst zu einem Rückstau bei der Datenübernahme kommt.

Allgemein wurde die Parallelisierung der Datenverarbeitung sowohl beim Speichern der übernommenen Datenobjekte sowie der Abfragen angestrebt. Die möglichen Parallelisierungsansätze sind allerdings durch die verwendete Technologie - im Big Data Bereich zum Beispiel Map-Reduce [dean04] oder Spark [chambers18] - vorgegeben. Daher stellen die Anforderungen an die Datenverarbeitung ein wesentliches Kriterium zur Technologieauswahl dar. Hierbei wurde die Frage untersucht, inwiefern sich eine Parallelisierung des Datenimports in und der Abfragen auf de Graphdatenbank umsetzen lässt, beziehungsweise ob die geplante Graphdatenbank Spark und Map-Reduce unterstützt.

Durch die auftraggeberseitige Festlegung auf eine Graphdatenbank wurden Datenbanksysteme in die engere Auswahl genommen, welche mit einem reinen Property-Graph Modell arbeiten. Neo4J als verbreitetste Graphdatenbanklösung wurde einerseits wegen der höheren Lizenzkosten für einen Clusterbetrieb sowie der Anforderung einer guten Integrierbarkeit mit einer Hadoop-Umgebung ausgeschieden.

Aufgrund der guten Integrierbarkeit mit einer Hadoop-Umgebung, der horizontalen Skalierbarkeit, sowie der Unterstützung von Map-Reduce und Spark fiel die Wahl auf die Open-Source Graphdatenbank Janusgraph, die alle oben genannten Kriterien erfüllt.

Die konkreten Tätigkeiten im Rahmen der Technologieauswahl umfassten hierbei eine umfangreiche Koordination mit dem Auftraggeber, um dessen Anforderungen und auch die Systemumgebung zu verstehen. Basierend darauf wurden die technischen Fähigkeiten und Limitierungen von verfügbaren Graphdatenbanklösungen verglichen. Praktische Evaluierungstätigkeiten von Graphdatenbanken wurden neben der Infrastruktur des Auftragnehmers auch auf der Amazon Public Cloud AWS durchgeführt, wofür Kosten im Antrag vorgesehen waren. Neben den technischen Möglichkeiten der Graphdatenbanken wurden auch die eingesetzten Abfragesprachen verglichen. Beispiele dafür sind die Abfragesprachen Cypher und Gremlin. Während Cypher primär von der Graphdatenbank Neo4J eingesetzt wird, stellt Gremlin eine Schnittstelle dar, welche von mehreren Graphdatenbanken implementiert wird. Bezüglich ihrer Mächtigkeit sind beide Sprachen äquivalent, und können auch automatisiert ineinander übergeführt werden.

2.1.3 JanusGraph

JanusGraph ist eine horizontal skalierbare Open-Source Graphdatenbank, die verschiedene NoSQL-Datenbanken als Storage Backend verwenden kann. Beispiele dafür sind Cassandra, HBase und Berkeley-DB. Ein JanusGraph Client kann entweder über eine Java Bibliothek und den Connection-String des Storage-Backends direkt auf die Daten zugreifen, wie in Abbildung 1 gezeigt, oder über einen Gremlin Server mit einer Gremlin-Query die Datenbank abfragen (Abbildung 2).

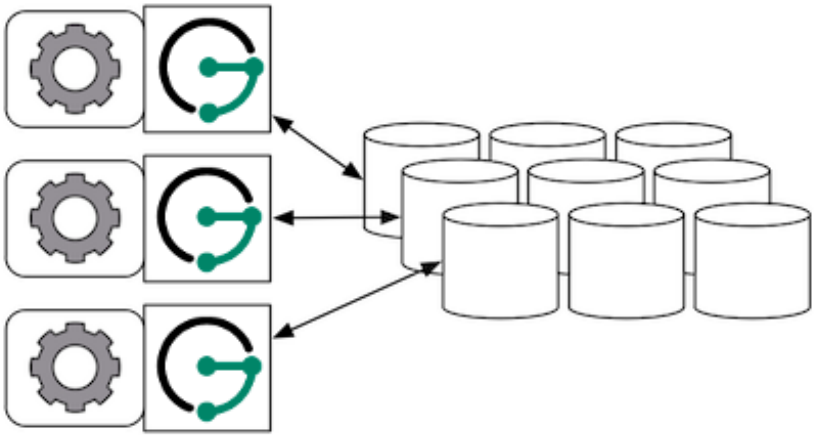

Abbildung 1: Direkter Zugriff auf das Storage-Backend

Im Rahmen des Projekts TRAPH wurde die erste Variante für Schreibzugriffe auf die Graphdatenbank verwendet, während die zweite Variante es ermöglicht, benutzerfreundliche interaktive Queries abzusetzen.

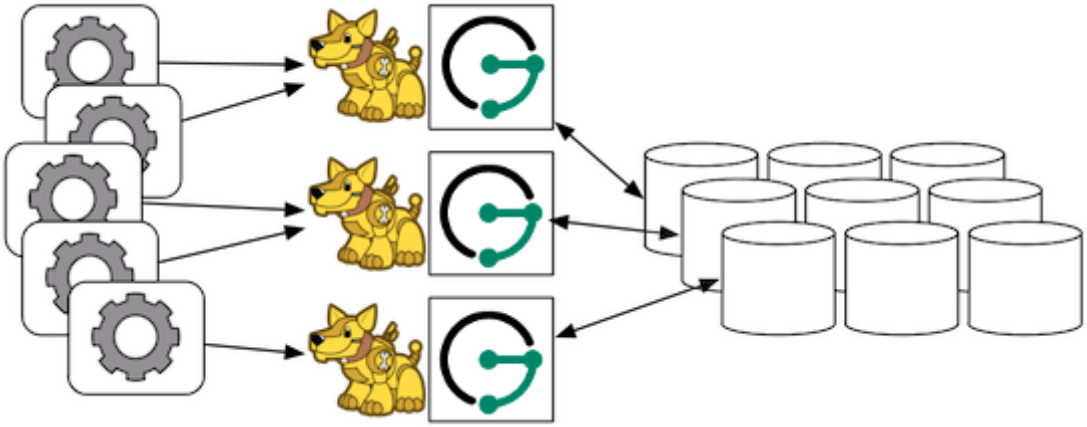

Abbildung 2: Zugriff über einen Gremlin-Server

2.1.4 Graph-Visualisierung

Im Zuge der Technologieanalyse wurden neben der Betrachtung von verschiedenen Graphdatenbanken auch Möglichkeiten der Graphvisualisierung untersucht, wobei nach Wunsch der ASFiNAG der Fokus auf webbasierte Visualisierungen gelegt wurde.

Im Zusammenhang mit dem Einsatz von JanusGraph als Graphdatenbank würde sich der Einsatz von GraphExp[8] als webbasiertes Visualisierungstool anbieten (vgl. Abbildung 3), da es direkt an JanusGraph angebunden werden kann.

[8] https://github.com/bricaud/graphexp

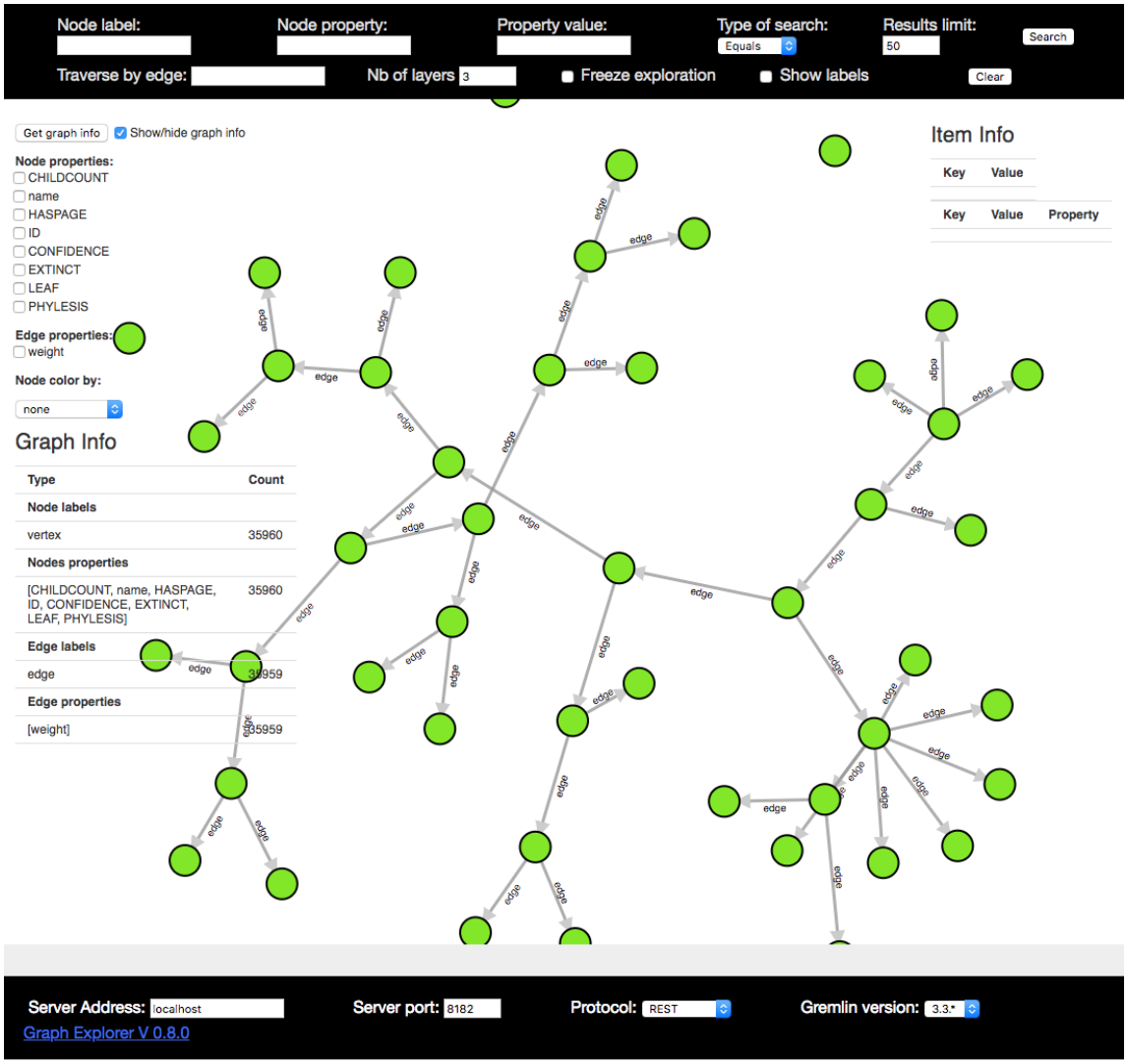

Abbildung 3: Beispielvisualisierung mit GraphExp

Abgesehen von der technischen Umsetzung der Visualisierung ist es aber unklar wie ein Visualisierungs- und Datenexplorationskonzept für große Graphen – wie beispielsweise in TRAPH – aussehen könnte. Dieses Problem müsste in einem eigenen Forschungsvorhaben untersucht werden, vor allem da mögliche Lösungen für dieses Problem wohl auch eine starke Anwendungsabhängigkeit aufweisen werden.

2.2 Design und Datenmodelle

Die Ziele umfassten hier den Entwurf des Datenmodells der Graphdatenbank beziehungsweise die Konfiguration der Graphdatenbanklösung entsprechend den Projektanforderungen.

2.2.1 Anforderungen an das Datenmodell

Hierzu wurden seitens des Auftraggebers beispielhafte Abfragen definiert, die als Leitlinien für den weiteren Entwurf des Datenmodells der Graphdatenbank dienten. Dieser lieferte eine sprachliche Beschreibung von beispielhaften geplanten Abfragen. Auf dieser Basis konnte RISC die konkreten Abfragen in der Abfragesprache Gremlin formulieren.

In Kombination mit der Technologieentscheidung, die Datenbank JanusGraph einzusetzen, wurde für die spezifizierten Abfragen ein geeignetes Datenmodell entworfen.

Folgende Abfragen wurden definiert, die jeweils für ein gewisses Zeitfenster durchgeführt werden sollen:

- Gib alle aktiven Situationen und/oder Maßnahmen zurück.
- Mit Angabe von Maßnahmen, gib alle Schaltanforderungen zurück.
- Welche Schaltanforderungen wurden durch welche Maßnahmen ausgelöst.
- Ausgehend von bestimmten Maßnahmen, welche Situationen liegen darunter. Falls bestimmte Situationen mit weiteren Maßnahmen zusammenhängen, gib auch diese Maßnahmen zurück.
- Bilde zumindest mit Zeiger ab, welche Daten zu welchen Situationen geführt haben. Mit Zeiger wird hier gemeint, dass nicht direkt Sensorinformationen im Graphen abgebildet werden müssen, aber es soll zumindest eine Information zu der Quelle (Datenart und Zeitfenster) geben.
- Welche Maßnahmen führen am häufigsten zu der gleichen Schaltanforderung.
- Welche Schaltanforderungen kommen am häufigsten vor auf einem Abschnitt der Autobahn.
- Wie lange bleiben Situationen und Maßnahmen aktiv.

Eine zentrale Herausforderung bei dem Entwurf des Datenmodells war die bereits in der Ausschreibung angegebene Anforderung der Abbildung zeitabhängiger Wirkungsbereiche, die durch eine Trennung des Graphdatenmodells in einen zeitabhängigen und eine nicht-zeitabhängigen Teil erreicht wurde. Im nicht-zeitabhängigen Teil wurden die Elemente abgebildet, die im Wesentlichen der Hardware an der Strecke entsprechen, wie zum Beispiel Sensoren oder Überkopfanzeiger. An die jeweiligen Knoten der Hardwareelemente wurde ein Baum als Subgraph angehängt, der die Messwerte oder die dargestellten Schaltungen nach Jahr, Monat und Tag des Auftretens filtert. Somit kann eine Abfrage auch innerhalb einer einfachen Graph-Traversierung zeitlich eingeschränkt werden.

Eine weitere Herausforderung war die Abbildung der Zeitbereiche, für die gewisse Situationen und Maßnahmen gültig sind. Hierzu wurde in den Knoten der Startzeitpunkt, sowie sobald verfügbar, der Endzeitpunkt erfasst. Dies erforderte das schnelle Wiederauffinden bereits gespeicherter Knoten, was einerseits durch einen eindeutigen Index und andererseits durch die Erstellung von Hashwerten für den Vergleich der Knoten ermöglicht wurde.

2.2.2 Analyse der Quelldaten

Die Quelldaten für den Import in die Graphdatenbank wurden über gRPC und Kafka übernommen und aus dem Protobuf-Format deserialisiert. Da sich das Rahmenprojekt VMIS2 noch in Entwicklung befand, mussten mehrere Versionen des VMIS2-Datenmodells eingebunden werden, was durch die Kompilierung des Datenmodells in korrespondierende Java-Deserialisierungsklassen umgesetzt wurde.

Grundsätzlich deckt das Datenmodell von VMIS2 eine große und komplexe Anwendungsdomäne ab, die den gesamten Datenfluss am österreichischen Autobahnsystem umfasst. Daher stellte die Analyse sowie das überblicksartige Verständnis des Gesamtdatensystems den ersten Schritt zum Aufbau des Datenmodells für die Graphdatenbank dar.

Neben einem groben Verständnis des Gesamtdatenmodells war es essenziell, die für die Nachvollziehbarkeit der Schaltentscheidungen der Verkehrsbeeinflussungsobjekte relevanten Datenobjekte zu identifizieren und ihre Zusammenhänge zu extrahieren. Dies

wurde einerseits durch die Unterstützung der ASFiNAG sowie ihrer externen Entwicklungspartner aber auch durch eigene Recherche des TRAPH-Projektteams ermöglicht.

Neben dem allgemeinen Verständnis des Datenmodells war es für die Modellierung der Nachvollziehbarkeit der Schaltentscheidungen von zentraler Bedeutung eine durchgehende Kausalkette von den Sensorinformationen bis zur Ansteuerung der Verkehrsbeeinflussungsanlagen zu finden.

Die Ermittlung dieser Zusammenhänge wurde oft noch von der parallellaufenden Weiterentwicklung des Datenmodells sowie der oft beschränkten Menge an verfügbaren Daten erschwert.

Im Datenmodell von VMIS2 wurden kausale Verknüpfungen einerseits durch Aggregation von abhängigen Datenelementen abgebildet, andererseits erfolgten manche Verknüpfungen über externe Identifier.

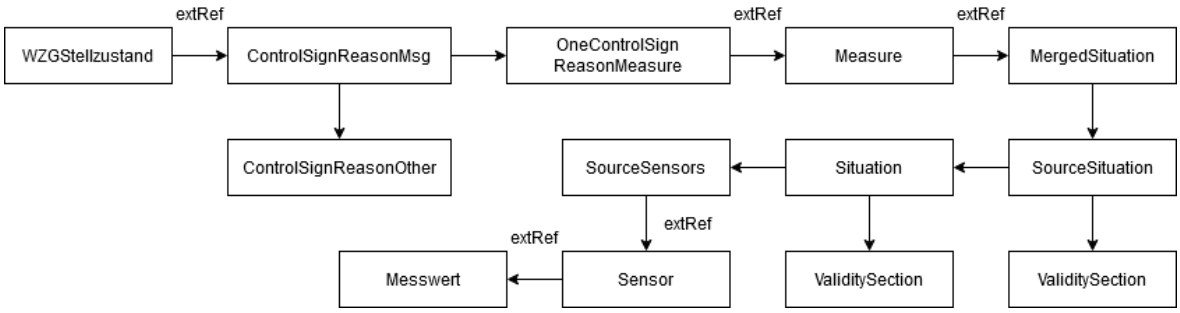

Abbildung 4: Relevante VMIS2 Datenelemente mit kausalen Verknüpfungen

In Abbildung 4 werden die für die Nachvollziehbarkeit relevanten VMIS2-Datenobjekte mit ihren Verknüpfungen gezeigt. Während sich Pfeile ohne Bezeichnung auf eine direkte Einbettung der Daten in das referenzierende Objekt beziehen, bezeichnen Pfeile mit *extRef* die Verknüpfung über eine externe Referenz.

Das Datenobjekt *WZGStellzustand* bezeichnet eine geplante Schaltung eines Verkehrsbeeinflussungsobjekts. Der Grund für diese Schaltung ist wiederum in einer *ControlSignReasonMsg* abgelegt, welche entweder eine Maßnahme (*OneControlSignReasonMsg*) oder ein anderer Grund (*ControlSignReasonOther*) sein

kann. Ein solch anderer Grund ist beispielsweise Gewährleistung der Anzeige der gleichen Geschwindigkeitsbeschränkung für alle Fahrstreifen einer Fahrtrichtung. In diesem Fall wurde die Ursache nicht mehr weiterverfolgt.

Im Fall einer begründenden Maßnahme, wie beispielsweise einer Geschwindigkeitsbeschränkung, referenziert diese wiederum eine zusammengeführte Situation (*MergedSituation*), die wiederum aus Quellsituationen (*SourceSituations*) folgt, und eine Verkehrssituation wie beispielweise eine Staubildung beschreibt.

Die Situationen werden wiederum aus Quellsensoren (*SourceSensors*) und ihren Messwerten errechnet.

2.2.3 Entwurf des Datenmodells

Die vom Auftraggeber formulierten Abfragen bildeten genauso die Basis für den Entwurf des Datenmodells, welches so geplant wurde, dass die spezifizierten Abfragen effizient durchgeführt werden konnten. Im Besonderen wurde dabei die Frage der bereits in der Ausschreibung des Projekts erwähnten zeitabhängigen Wirkungsbereiche berücksichtigt.

Basierend auf der aus dem VMIS-Datenmodell extrahierten Kausalkette wurde im nächsten Schritt das Datenmodell für die Graphdatenbank entwickelt. Hierbei musste auf die Spezifika von Graphdatenbankabfragen Rücksicht genommen werden wie beispielsweise der Fokus auf das Verfolgen von Verknüpfungen zwischen den Knoten des Graphen.

Ein weiterer kritischer Aspekt beim Entwurf des Datenmodells war durch die bereits in der Projektausschreibung spezifizierte Einbindung der Zeitkomponente gegeben. Im Besonderen die oben erwähnten Maßnahmen sowie Situationen weisen eine zeitliche Ausdehnung auf, und können durchaus zum Zeitpunkt einer Abfrage auf der Graphdatenbank noch andauern. Daher muss das Datenmodell diese Aspekte sowie die Möglichkeit einer einfachen zeitabhängigen Filterung von Situationen und Maßnahmen ermöglichen.

Das Ziel war daher, zeitabhängige Elemente, welche in der obigen Kausalkette auftreten, gemeinsam mit zeitunabhängigen statischen Elementen in einem Gesamtgraph

abzubilden. Die statischen Elemente umfassen hierbei wie oben bereits erwähnt, die Hardware an der Strecke wie beispielsweise Sensoren oder Überkopfanzeiger. Die Daten hierzu werden bei einem Importerlauf in die Graphdatenbank von einer gRPC-Schnittstelle bezogen.

Diese liefert die vordefinierten Wirkungsbereiche an der Strecke mit der entsprechenden Autobahn und Kilometrierung mit zusätzlich den GPS-Koordinaten. Weiters werden über die gRPC-Schnittstelle die Positionen der Überkopfanzeiger sowie der Sensoren an der Strecke bezogen. Weitere Elemente sind Mess- und Anzeigequerschnitte, welche Sensoren beziehungsweise Verkehrsbeeinflussungsanlagen örtlich zusammenfassen.

2.2.4 Graph-Datenmodell

Grundsätzlich stellt ein Knoten in Abbildung 4 eine ganze Klasse von Knoten dar. Der oberste Knoten *Road* wird bei einer konkreten Ausprägung des Graphen durch die Menge der Knoten für alle Autobahnen ersetzt. Analog gilt dies auch für andere Knoten. Die Darstellung ermöglicht so allerdings eine vereinfachte Darstellung des Graphen und unterstützt damit die Lesbarkeit.

Grundsätzlich lässt sich das Datenmodell für den Nachvollziehbarkeitsgraphen in einen statischen sowie einen dynamischen Teil unterteilen. Der statische Teil beschreibt die Hardware an der Strecke, währen der dynamische Teil die Kausalkette zwischen Sensorwerten und Schaltungen der Überkopfanzeiger abbildet.

Der statische Teil umfasst im Modell in Abbildung 5 den oberen Teil des Graphen bis einschließlich der Knotentypen *VdeSensor*, *UdeSensor* und *Panel*. In diesem Teil des Graphen sind die Daten abgebildet, welche beim Import aus den gRPC-Daten gebildet werden. Diese umfassen in der obersten Ebene die Autobahnen selbst (*Road*) die wiederum alle GPS-Positionen (*GPSPosition*) und Gültigkeitsbereiche (*ValiditySectionPosition*) referenzieren, die auf ihnen auftreten. Die Messquerschnitte (*MeasurementSection*) und Anzeigequerschnitte (*DisplaySection*) darunter referenzieren wiederum die korrespondierenden Sensoren und Anzeigepaneele.

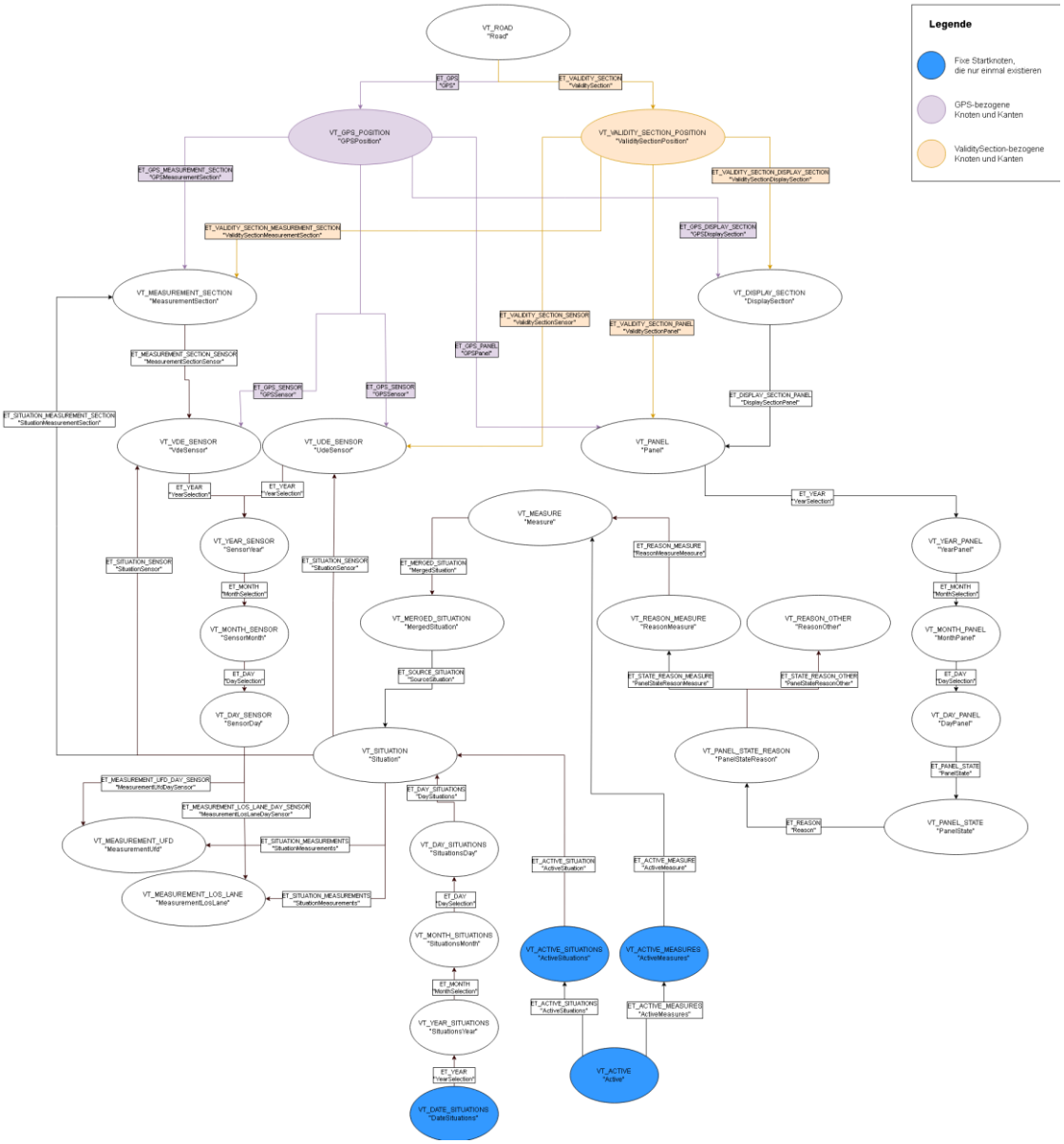

Abbildung 5: Graph-Datenmodell

Unterhalb der Knoten für Sensoren und Paneele ist für jeden Sensor und jedes Paneel ein Unterbaum aufgehängt, der die Messwerte oder Anzeigen nach deren Start- und Endzeitpunkten gliedert. Jeder Unterbaum unterteilt sich in die Ebenen Jahr, Monat, Tag und ermöglicht damit ein schnelles Auffinden der Ereignisse in einem vorgegebenen Zeitbereich.

Ausgehend von einem PanelState unterhalb eines Panels kann nun die oben beschriebene Kausalkette bis zu den auslösenden Sensorwerten unterhalb der Sensoren verfolgt werden.

Da innerhalb einer Abfrage der Graph in beide Richtungen traversiert werden kann, ermöglicht dieses Datenmodell sehr flexible Abfragen.

Im speziellen seien noch die Knoten für die aktuell gültigen Situationen (*Active Situations*) und Maßnahmen (*Active Measures*) erwähnt, die einen sehr schnellen Zugriff auf gerade laufende Ereignisse ermöglichen.

2.2.5 Graph-Schema

Nach dem Entwurf des Datenmodells wurde als erster Umsetzungsschritt das Schema für die Graphdatenbank definiert, analog zum Datenmodell im relationalen Fall. Dies hat den Vorteil, das schemaverletzende Einfügungen direkt von der Datenbanksoftware zurückgewiesen werden.

Das Schema des Graphen beinhaltet die Typinformation für alle zulässigen Knoten und Kanten des Graphen. Für jeden Knotentypen sind seine Attribut-Felder vermerkt. Die Typen der Kanten schränken wiederum die Typen der damit verbindbaren Knoten ein.

Es wurde für jede Knotenart, die in Abbildung 5 jeweils mit dem Präfix VT bezeichnet ist, ein Typ mit den zulässigen Attributen definiert. Zu den Attributen gehören typischerweise ein Identifier, ein Name und zusätzliche relevante Attribute wie Start- und Endzeitpunkte. Die Typen für die Graphkanten wurden mit dem Präfix ET versehen und beinhalten keine weiteren Attribute, da sie im Datenmodell von TRAPH nur der logischen Verknüpfung von Knoten dienen.

2.2.6 Formulierung der ASFiNAG-Abfragen in Gremlin

In diesem Abschnitt werden die von der ASfiNAG beispielhaft genannten Abfragen für das Graph-Modell in Gremlin übertragen. Grundsätzlich sind Gremlin-Abfragen so aufgebaut, dass sie seiner Graph-Traversierung entsprechen, die entweder bei einem einzelnen Knoten oder parallel bei mehreren Knoten, die einem Kriterium entsprechen, startet.

In den folgenden Abfragen stellt das Objekt *at* die Quelle der Graphtraversierung dar, die auch durch Kriterien wie Knoten-Ids oder Vergleiche auf Attribute noch weiter eingeschränkt werden kann.

Die Abfragen beziehen sich auf das in Abbildung 5 dargestellte Graph-Modell.

2.2.6.1. Aktive Situationen und Maßnahmen

Gib alle aktuell aktiven Situationen und/oder Maßnahmen zurück

```
at.V().has("Active", "hash", 0).
out("ActiveSituations").out("ActiveSituation").limit(10).valueMap(true).fold()

at.V().has("Active", "hash",
0).out("ActiveMeasures").out("ActiveMeasure").limit(10).valueMap(true).fold()
```

Bei dieser Abfrage wird ausgenutzt, dass das Datenmodell einen dedizierten Startknoten für alle aktiven Knoten vorsieht, der über seinen Knotentypen *Active* referenziert wird. Davon ausgehende Kanten werden dann über *out()* mit dem jeweiligen Kantentypen als Parameter referenziert. Damit wird der Graph bis zu den aktiven Situationen traversiert bevor mit *valueMap()* eine Map der Attribute des Knoten ausgegeben wird, die durch *fold()* in eine Liste umgewandelt wird.

Die zweite Abfrage gilt analog für die aktiven Maßnahmen.

2.2.6.2. Schaltanforderungen für gegebene Maßnahmen

Mit Angabe von Maßnahmen, gib alle Schaltanforderungen zurück, die mit den abgefragten Maßnahmen verknüpft sind.

```
at.V(439218368).in("ReasonMeasureMeasure").in("PanelStateReasonMeasure")
.in("Reason").valueMap(true)
```

In dieser Abfrage wird von einer konkreten Maßnahme ausgegangen, die hier über ihre JanusGraph-Id referenziert wird, und die Kanten bis zum Zustand des Überkopfanzeigers in umgekehrter Richtung zur Kantenrichtung traversiert. Das Ergebnis wird wieder als *ValueMap* ausgegeben.

2.2.6.3 Maßnahmen für gegebene Schaltanforderungen

Welche Schaltanforderungen wurden durch welche Maßnahmen ausgelöst. Es werden alle Maßnahmen zurückgegeben, die mit den abgefragten Schaltanforderungen verknüpft sind.

```
at.V(31334592).out("Reason").out("PanelStateReasonMeasure").out("ReasonMeasure
Measure").valueMap(true)
```

Diese Abfrage ist analog zur der in Abschnitt 2.2.6.2, allerdings in umgekehrter Richtung.

2.2.6.4 Situationen für gegebene Maßnahmen

Ausgehend von bestimmten Maßnahmen, welche Situationen liegen darunter. Falls bestimmte Situationen mit weiteren Maßnahmen zusammenhängen, gib auch diese Maßnahmen zurück

```
at.V(414654656).hasLabel("Situation").in("SourceSituation").in("MergedSituation")
.valueMap(true)
```

Hier werden ausgehend von einem Situationsknoten die verbundenen Maßnahmenknoten traversiert.

2.2.6.5 Maßnahmen für gegebene Schaltanforderungen

Welche Maßnahmen führen innerhalb des abgefragten Zeitbereich am häufigsten zu der gleichen Schaltanforderung.

```
at.V().has("YearPanel", "year", 2021).out("MonthSelection").has("MonthPanel", "month",
2).out("DaySelection").has("DayPanel", "day", 10).out("PanelState").out("Reason").
out("PanelStateReasonMeasure").out("ReasonMeasureMeasure")
.group().by("objectId").by(__.in("ReasonMeasureMeasure")
.in("PanelStateReasonMeasure").in("Reason").groupCount().by("id").
order(local).by(values, Order.decr).limit(local, 1))
.unfold().filter(select(values).where(count(local).is(gt(0))))
.project("maß", "wzg", "anz").by(keys).by(select(values).unfold()
.select(keys).unfold()).by(select(values).unfold()
.select(values).unfold()).order().by(select("anz"),Order.decr)
```

Hier werden zuerst die *PanelStates* für einen spezifischen Tag über die Kantenverknüpfung Jahr – Monat – Tag abgefragt. Weiter werden die begründenden Maßnahmen extrahiert und nach ihrer Id gruppiert. Diese Gruppen werden dann weiter nach ihrer Größe sortiert.

2.2.6.6 Schaltanforderungen auf einem Autobahnabschnitt

Welche Schaltanforderungen kommen am häufigsten vor auf einem Abschnitt der Autobahn

```
at.V(430128).out("ValiditySectionPanel").out("YearSelection").has("YearPanel", "year",
2021).out("MonthSelection").has("MonthPanel", "month", 2).out("DaySelection")
.has("DayPanel", "day", 10).out("PanelState").groupCount().by("id")
.order(local).by(values).unfold().project("schaltanforderung", "anz").by(keys).by(values)
```

Ausgehend von einer ValiditySection als Abschnitt der Autobahn, die hier über die Knoten-Id angegeben wird, wird wieder ein Tag als Zeitbereich ausgewählt, dann werden die Anzeigezustände nach Wert gruppiert und sortiert

2.2.6.7 Dauer von Situationen und Maßnahmen

Wie lange bleiben Situationen und Maßnahmen aktiv

```
at.V().hasLabel("Measure").as("v").where("v",  gte("v")).by("endTime").by("startTime")
.project("st", "et").by("startTime").by("endTime").math("(et - st)/1000/60").mean()

at.V().hasLabel("Situation").as("v").where("v",
gte("v")).by("endTime").by("startTime").project("st", "et")
.by("startTime").by("endTime").math("(et - st)/1000/60").mean()
```

Diese Abfrage für Situationen und Maßnahmen zeigt die Möglichkeit Attributwerte von Knoten, hier Start- und Endzeitpunkt für Berechnungen heranzuziehen, indem der Mittelwert über alle erfassten Zeitdauern von Situationen und Maßnahmen gebildet wird.

2.3 Prototypische Implementierung

Nach der Durchführung der Technologieanalyse und dem Entwurf des Datenmodells wurde im nächsten Schritt die prototypische Implementierung der Graphdatenbank und des Datenimports umgesetzt.

Im ersten Schritt der Implementierung wurde der Fokus auf einen korrekten und vollständigen Datenimport sowie eine den Anforderungen entsprechend konfigurierte Datenbank gelegt. Arbeiten zur Verbesserung der Performanz des Datenimports wurden in einem späteren Schritt durchgeführt.

Die im Antrag spezifizierten Aufgaben für die Implementierung des Prototyps waren wie folgt:

- Implementierung einer prototypischen Datenhaltungslösung: Da als Ergebnis der Technologieevaluierung die Graphdatenbank JanusGraph gewählt wurde, und in der Systemumgebung Kafka und der Microsoft SQL-Server als Datenquellen vorgegeben waren, wurde ein java-basierter Datenimporter umgesetzt, der aus Kafka und dem SQL Server die Binärdaten extrahierte und diese unter Verwendung von Google Protobuf deserialisierte. Die Daten wurden dann in das entworfene Datenmodell abgebildet und in JanusGraph gespeichert.
- Test der benötigten Datentransformationen: Ausgiebige Tests waren Teil der Implementierungsphase, genauso wie Überlegungen zur Beschleunigung des

Datenimports. Für den Datenimport war hierbei vor allem die Abbildung des Datenmodells VMIS2, welches vom Kafka-Broker geliefert wurde, in das Graphdatenmodell zentral. Hierbei wurden die benötigten Verknüpfungen, die im VMIS2 Modell teilweise nur implizit vorlagen, extrahiert und gemeinsam mit den Knotendaten in JanusGraph gespeichert.

Zusätzlich wurde im Rahmen der prototypischen Implementierung auch noch die Testumgebung, im Speziellen eine JanusGraph-Instanz aufgebaut.

2.3.1 Setup der Graphdatenbank JanusGraph

Seitens der ASFiNAG wurden für das Testdeployment der Graphdatenbank drei virtuelle Maschinen mit CentOS Linux als Betriebssystem zur Verfügung gestellt. Auf diesen wurden die benötigten Dienste in Docker-Containern betrieben.

JanusGraph ermöglicht den Einsatz verschiedener Storage Backends, unter anderem Cassandra und HBase. Für das Deployment auf den ASFiNAG Servern wurde Cassandra gewählt, da dies bezogen auf die Konfiguration und die benötigten Abhängigkeiten, die am schnellsten umsetzbare Lösung war.

Die Unterschiede, die sich aus der Wahl des Storage-Backends ergeben, beschränken sich im Fall von JanusGraph auf die Konfiguration der Graphdatenbank und sind für den implementierten Importercode transparent. Daher wurde zwar im Rahmen der Technologieanalyse auch der Einsatz von HBase als Storage-Backend bewertet, als Testumgebung für den Importercode dann aber Cassandra eingesetzt.

Aufgrund der Architektur ist JanusGraph von dem eingesetzten Storage Backend entkoppelt. Damit ist es möglich, die JanusGraph Gremlin-Server zum Absetzen der Gremlin Queries nach Bedarf zu starten, diese greifen über das Netzwerk auf das Storage Backend zu. Analog dazu verwendet der java-basierte Importer direkt die JanusGraph-Bibliothek um über Netzwerkzugriff direkt ins Storage-Backend zu schreiben.

Der Einsatz der JanusGraph-Bibliothek ermöglicht es dabei, innerhalb von Transaktionen direkt Knoten und Kanten zu speichern., während diese im Storage-Backend pro Knoten

zeilenweise abgelegt werden. Jede Zeile beinhaltet weiters die Kanten, die mit dem jeweiligen Knoten verknüpft sind. Diese Methode der Speicherung ermöglicht eine schnelle Traversierung des Graphen, da über die Knoten-Id die korrespondierende Zeile direkt aus dem Storage Backend geholt werden kann.

2.3.2 Datenextraktion

Als erster Schritt für die Protoypimplementierung wurde die Datenextraktion aus den Datenquellen Apache Kafka sowie eines Microsoft SQL-Servers, die beide vom Auftraggeber zur Verfügung gestellt wurden und somit Teil der Systemumgebung waren, umgesetzt. Diese Aufgabe umfasste einerseits die Datenextraktion und andererseits die Deserialisierung aus dem Google Protocol Buffers Format sowie die Speicherung in der JanusGraph-Datenbank gemäß dem entworfenen Datenmodell.

Da die Ausgangsdaten einerseits von einem Kafka-Broker sowie andererseits von einer SQL-Datenbank geliefert wurden, wurde die Datenquelle als abstrakte Klasse implementiert, welche Binärfelder an den Deserialisierungsteil übergibt. Weiters wurde die Zuordnung zwischen Kafka-Topics, MS-SQL Tabellen sowie den Protobuf-Klassen zur Deserialisierung der Topics in einer HashMap abgelegt, womit alle Aspekte der Datenextraktion von der konkreten Datenquelle entkoppelt wurden.

Parallel dazu wurden die statischen Daten, welche von der gRPC-Schnittstelle geliefert wurden, mit Hilfe korrespondierender generierter Protobuf-Klassen deserialisiert.

Die Deserialisierung selbst wurde durch Deserialisierungsklassen erledigt, die durch den Protobuf-Compiler aus den Protobuf-Schemas, die von der ASFiNAG zur Verfügung gestellt wurden, erzeugt wurden.

Da der SQL-Server nach Fertigstellung des Entwicklungsprojekts VMIS2 nicht mehr weiterbetrieben wird, wurde der Import neuer Daten konzeptionell auf den Kafka-Broker ausgerichtet. Dazu gehört unter anderem, das der Kafka-Broker mitführt, welche Daten bereits vom Importer ausgelesen wurden. Dies führt automatisch zu einer verringerten Menge an Daten, die in einem Importerdurchlauf importiert werden müssen, und reduziert ebenfalls die Menge der auftretenden Duplikate.

Generell setzt sich eine Importerdurchlauf aus zwei Teilen zusammen, zuerst dem Import der statischen Daten aus den gRPC-Schnittstellen und danach die dynamischen Daten vom Kafka-Broker. Beim Zugriff auf den Kafka-Broker werden dabei alle Daten, die noch neu hinzugekommen sind, als Block abgerufen und gleich unter Einsatz von Protobuf deserialisiert. Die deserialisierten Protobuf-Objekte werden dann im weiteren Importablauf in den Graphen integriert.

Für die Erzeugung der korrespondierenden Graphknoten wurde zunächst Hashmaps aus den Prootbuf-Objekten erzeugt, die dann mit einer generellen Methode direkt in JanusGraph-Knoten umgewandelt wurden.

Der Import wurde dabei nach Knotentypen gruppiert, wobei das Graphmodell von den Straßen ausgehend abgearbeitet wurde.

2.3.3 Entfernen duplizierter Knoten

Eine Herausforderung, welche sich während der Umsetzung der Implementierung ergab war das Herausfiltern von duplizierten Knoten, da Datenelemente vom Kafka-Broker oft mehrfach geliefert wurden, im Besonderen, wenn Maßnahmen oder Situationen länger andauerten und somit mehrfach mit unterschiedlichen Zeitstempeln übertragen wurden. Daher war es notwendig, effizient zu überprüfen, ob der aktuell einzufügende Knoten bereits im Graph vorliegt. In diesem Fall müssen Objekte wie Maßnahmen und Situationen in der Graphdatenbank angepasst werden, anstatt die neuen Werte als neues Objekt einzufügen. Diese Vorgehensweise ist vor allem für die Berücksichtigung des Zeitaspekts bei länger laufenden Situationen und Maßnahmen zentral. Hierbei stellt das Auffinden bereits vorhandener Duplikate einen wesentlichen Flaschenhals für die Import-Performanz dar, weil eine Traversierung des gesamten Graphen im Bereich von Terrabytes an Daten nicht mehr in Frage kommt. Zusätzlich stellt auch der attributweise Vergleich von Knoten selbst einen nicht unwesentlichen Zeitaufwand dar.

Aus diesen Gründen wurden mehrere Überlegungen zur Beschleunigung des Datenimports eingesetzt. Im ersten Schritt wurden die Daten, die in einem Importerdurchlauf importiert wurden, in HashMaps oder HashSets gespeichert, wodurch schon die Java-Laufzeitumgebung garantiert, das keine Duplikate mehr erhalten bleiben. Darüber hinaus

muss allerdings auch der bereits bestehende Graph auf Duplikate getestet werden. Dies wurde je nach dem Bereich der Graph unterschiedlich gelöst.

Hierzu wurden mehrere Methoden eingesetzt, abhängig vom Traversierungsaufwand über die Geschwisterknoten sowie des Vorhandenseins eines eindeutigen Index auf dem einzufügenden Knotentyp. Zentral war dabei der Einsatz von Composite Indices auf Knotenmengen desselben Typs, welche ein schnelles Auffinden von Graphknoten mit vorgegebenen Attributwerten ermöglichen.

Für Knotentypen, für die solcher ein eindeutiger Index definiert ist, wurde das Anlegen von neuen Knoten derart gekapselt, dass im Fall eines bereits existierenden Knotens die entsprechende Ausnahme gefangen wird und der bereits existierende Knoten zurückgeliefert wird. Dieser kann in weiterer Folge angepasst werden, wie zum Beispiel durch das Hinzufügen eines Enddatums zu einer aktuell laufenden Situation.

Diese Ausnahme entsteht dadurch, dass für eine im Graph-Schema definierte Knotenart ein *unique Index* definiert wird, und ein Wiedereinfügen desselben Knoten damit eine Schemaverletzung verursacht, welch vom JanusGraph-Laufzeitsystem zurückgewiesen wird.

Sofern ausgehend von einem konkreten Knoten die Menge der Kindknoten beschränkt war, war auch eine direkte Traversierung der Knoten möglich. Zum Zwecke der Beschleunigung des Knotenvergleichs wurde bei der Erzeugung von Knoten ein Hashwert über die relevanten Attribute gebildet, womit sich der Aufwand des Attributvergleichs auf ein einziges Datenfeld pro Knoten reduzieren ließ.

Eine zentrale Herausforderung bei dem Entwurf des Datenmodells war die bereits in der Ausschreibung angegebene Anforderung der Abbildung zeitabhängiger Wirkungsbereiche, die durch eine Trennung des Graphdatenmodells in einen zeitabhängigen und eine nicht-zeitabhängigen Teil erreicht wurde. Im nicht-zeitabhängigen Teil wurden die Elemente abgebildet, die im Wesentlichen der Hardware an der Strecke entsprechen, wie zum Beispiel Sensoren oder Überkopfanzeiger. An die jeweiligen Knoten der Hardwareelemente wurde ein Baum als Subgraph angehängt, der die Messwerte oder die dargestellten Schaltungen nach Jahr, Monat und Tag des Auftretens filtert. Somit kann eine Abfrage auch innerhalb einer einfachen Graph-Traversierung zeitlich eingeschränkt werden.

Eine weitere Herausforderung war die Abbildung der Zeitbereiche, für die gewisse Situationen und Maßnahmen gültig sind. Hierzu wurde in den Knoten der Beginnzeitpunkt, sowie sobald verfügbar, der Endzeitpunkt erfasst. Dies erforderte das schnelle Wiederauffinden bereits gespeicherter Knoten.

2.4 Skalierbarkeit und Performance

Ein zentrales Ziel von TRAPH war die Skalierbarkeit der umgesetzten Graphdatenbanklösung zu gewährleisten. Dies wurde primär durch die Technologieentscheidung für JanusGraph gewährleistet, es erforderte aber auch einige Anpassungen des Importprozesses, um die Daten mit der erforderlichen Datenrate in die Graphdatenbank zu übertragen.

Dies umfassten vor allem den Aufbau und das Management von Vertex-Indices, die zur Beschleunigung von Abfragen sowie des Imports selbst dienen. Da die Erstellung beziehungsweise die Anpassung der Indizes an neu eingefügte Knoten gerade in verteilten Janusgraph Datenbanken einen berechnungsaufwendigen Prozess darstellt, wurde die Neuberechnung der Indizes asynchron durchgeführt, um sie mit dem Hinzufügen von Knoten zeitlich überlappen zu können. Darüber hinaus wurden die Indizes nur dann neu aufgebaut, wenn dies durch das Hinzufügen neuer Knoten notwendig geworden war. Blockierende Operationen wie das Warten auf die Fertigstellung eines Index wurde nur an jenen Stellen im Code eingefügt, an denen der entsprechende Index benötigt wurde, um die Datenkonsistenz nach dem Einfügen neuer Knoten zu gewährleisten. Die Finalisierung von Indizes, welche nur für die Gremlin-basierten Abfragen des Graphmodells benötigt werden, wurden erst am Ende des Imports abgewartet.

Weitere Untersuchungen zur Skalierbarkeit unter Realbedingungen beziehungsweise diesbezügliche Anpassungen des Importercodes und der Datenbankkonfiguration wurden im Antrag geplant, aber aufgrund der geänderten Anforderungen beziehungsweise der Nichtverfügbarkeit von Testdaten im entsprechenden Ausmaß in Abstimmung mit dem Auftraggeber nicht mehr durchgeführt.

Dies bezieht sich vor allem auf die Durchführung von Lasttests unter Realbedingungen, welcher aufgrund der eingeschränkten Verfügbarkeit von Testdaten nur beschränkt möglich war.

In Bezug auf Lasttests unter simulierten Realbedingungen lässt sich sagen, dass sich ein solcher Test aufgrund des Status des übergeordneten Projekts des Auftraggebers, welches als Datenlieferant diente, im Rahmen der Projektlaufzeit nicht umsetzen ließ. Sehr wohl wurde eine Beurteilung der Performance der umgesetzten Implementierung durchgeführt. Aufgrund des Ergebnisses dieser Beurteilung wurden zur Beschleunigung der Abfragen und des Datenimports zahlreiche Indizes zum Graphdatenmodell hinzugefügt, die auch zu einer wesentlichen Beschleunigung des Datenimports bis zu einem Faktor 40 führten.

3. ZUSAMMENFASSUNG

Das Projekt TRAPH hatte zum Ziel den Einsatz einer Graphdatenbanklösung für die Verbesserung der Nachvollziehbarkeit der Schaltungen der Verkehrsbeeinflussungsanlagen auf den Autobahnen der ASFiNAG zu untersuchen. Dieses Ziel wurde schrittweise umgesetzt, indem zuerst auf Basis einer Technologieanalyse eine geeignete Datenlösung ausgewählt wurde und dies dann in einer Testumgebung aufgesetzte wurde.

Weiters wurde ein geeignetes Datenmodell zur Abbildung der relevanten Objekte aus dem VMIS2-Datenmodell entworfen. Mit der Implementierung eines java-basierten Importers gelang es, den Nachweis zu erbringen, dass die gewählte Graphdatenbank in der Lage ist die Anforderungen der ASFiNAG in Bezug auf Skalierbarkeit und Query-Performance zu erfüllen.

Die Dokumentation der technischen Projektergebnisse wurde durchgeführt und an den Auftraggeber übergeben. Dies umfasst einerseits die Code-Dokumentation mittels Javadoc sowie weiterer erläuternder Kommentare genauso wie die Dokumentation der Konfiguration der Graphdatenbank.

3.1 Projektfazit

Die Projektergebnisse von TRAPH ermöglichen es somit der ASFiNAG, die Einbindung einer Graphdatenbank in die Produktivumgebung von VMIS2 als gut geeignete Option in Betracht zu ziehen. Zusätzlich können einerseits der entwickelte Importercode sowie andererseits die im Rahmen des Projekts eingesetzte Konfiguration der Datenbank JanusGraph als solide Basis für die Weiterentwicklung der Graphdatenbnak zu einem produktiv einsetzbaren System dienen.

3.2 Weiterer zukünftiger Entwicklungsbedarf

Um die prototypische Graphdatenbank sowie den Importercode für den Einsatz in einer Produktivumgebung vorzubereiten, sind noch einige Adaptierungen notwendig. Diese

umfassen im Wesentlichen den Einsatz von Hadoop und Spark, um die horizontale Skalierbarkeit zu verbessern.

Weitere Anpassungen sind auch aufgrund der Weiterentwicklung des Rahmenprojekts VMIS2 zu erwarten, wie zum Beispiel Anpassungen des Datenmodells oder der Anbindung an den Kafka-Broker.

LITERATURVERZEICHNIS

[dean04] Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp. 137-150

[chambers18] Bill Chambers, Mattei Zacharia: Spark: The Definitive Guide: Big data processing made simple, O'Reilly, 2018

[white15] Tom White: Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale, O'Reilly, 2015