HAW HAMBURG

Bachelorarbeit

Nadja El-Busta

Explorative Visualisierung dynamischer Kommunikationsstrukturen in einem Publish-Subscribe-System

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Nadja El-Busta

Explorative Visualisierung dynamischer Kommunikationsstrukturen in einem Publish-Subscribe-System

Bachelorarbeit eingereicht im Rahmen der Bachelorstudiengang
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Bettina Buth

Eingereicht am: 11.07.2022

Nadja El-Busta

Thema der Arbeit

Explorative Visualisierung dynamischer Kommunikationsstrukturen in einem Publish-Subscribe-System

Stichworte

Publish-Subscribe-System, Visualisierung, dynamische Kommunikationsstrukturen, Middleware, grafisches Monitoring

Kurzzusammenfassung

Diese Arbeit setzt sich mit dem Thema Visualisierung von dynamischen Kommunikationsstrukturen in Publish-Subscribe-Systemen auseinander. Im Rahmen dieser Arbeit wurde ein Prototyp entwickelt und evaluiert.
Monitoring in verteilten Systemen ist unter anderem durch die Komplexität dieser Systeme mit einer hohen Dichte an Informationen verbunden. Visualisierung soll große Datenmengen anschaulich und aussagekräftig darstellen. Neben der Komplexität von verteilten Systemen stellen weitere Faktoren wie Dynamik, Nebenläufigkeit und Offenheit Herausforderungen bezüglich der Überwachung solcher Systeme dar.
Die dynamischen Kommunikationsstrukturen in einem Publish-Subscribe-System abzubilden ist eine Möglichkeit, diesen Herausforderungen bei einem solchen Systemdesign zu begegnen. Eine Publish-Subscribe-Architektur beschreibt ein verteiltes System mit einer lose gekoppelten Kommunikationsform. Eine lose Kopplung schafft Unabhängigkeit zwischen den Systemkomponenten. Unabhängigkeit dieser Art hat zur Folge, dass Teilnehmer des Systems keine Information über das System oder andere Systemteilnehmer besitzen. Die Kommunikation geschieht indirekt und mit unbekannten Kommunikationspartnern. Eine Blackbox-Betrachtung der Kommunikation soll durch eine explorative Visualisierung der dynamischen Kommunikationsstrukturen aufgehoben werden. Auf diese Weise soll ein Verständnis der Systemstruktur erleichtert und Systemverhalten verifiziert werden. Die Ursache von nicht erwartungskonformen Kommunikationsverhalten kann auf diese Weise geprüft werden.

Nadja El-Busta

Title of Thesis

Explorative visualization of dynamic communication structures in a Publish-Subscribe-System

Keywords

Publish-Subscribe-System, visualization, dynamic communication structures, middleware, graphical monitoring

Abstract

This thesis deals with the topic of visualization of dynamic communication structures in publish-subscribe-systems. A prototype was developed and evaluated as part of this thesis.

Monitoring in distributed systems is associated with a high density of information due to the complexity of these systems. visualization should present large amounts of data in a clear and meaningful way. In addition to the complexity of distributed systems, other factors such as dynamics, concurrency and openness present challenges concerning monitoring such systems.

Mapping dynamic communication structures in a publish-subscribe-system is one way to encounter those challenges in such a system design. A publish-subscribe-architecture describes distributed system with a loosely coupled form of communication.

Loose coupling creates independence between system components. Such independence means, that system participants have no information about the system or other participants. Communication happens indirectly and with unknown communication partners.

A black-box view of communication should be set aside by an explorative visualization of the dynamic communication structures. In this way, an understanding of the system structure should be facilitated as well as system behaviour verified. The cause of unexpected communication behaviours can be examined in this way.

Inhaltsverzeichnis

1 Einleitung

In verteilten Systemen stellt das Erfassen des Verhaltens der beteiligten Komponenten im System und das aller Systemzustände zu einem bestimmten Zeitpunkt eine Herausforderung dar. Das Erfassen ist hilfreich, um Fehler zu identifizieren und ein korrektes Systemverhalten zu verifizieren. Verhaltens- und Zustandserfassung in verteilten Systemen können bislang nur mit Einschränkungen erfüllt werden. Früher oder später besteht in jedem Entwicklungsprozess ein Interesse, das Verhalten und die Zustände der Komponenten zu kennen und diese zur Laufzeit zu überwachen.

Anwendungen, die zur Überwachung monolithischer Programme existieren, sind auf isolierte Komponenten und deren seriellen Abläufe ausgerichtet [21].

Für den Entwickler[1] einer Anwendung oder eines Systems kann es hilfreich sein, aus der grafischen Darstellung der Kommunikationsstrukturen Informationen zu gewinnen, um aus diesen Informationen Schlussfolgerungen ziehen zu können. Der Gewinn von Laufzeitinformationen dient dem Finden von Fehlern, der Kontrolle, dem Verstehen von Abläufen und zur Demonstration der Kommunikationsstrukturen einer Anwendung.

In verteilten Systemen gibt es unterschiedliche Arten der Kopplung der einzelnen Subsysteme. Eines der aktuellen und losen Kopplungssysteme ist das Publish-Subscribe-System. Die Kommunikation zwischen den Komponenten zu verstehen und diese abzubilden, gestaltet sich in Publish-Subscribe-Systemen schwierig.

Bei Verteilungsmodellen wie dem Publish-Subscribe-System stehen die dynamischen Kommunikationsstrukturen stärker im Vordergrund. Die Visualisierung von Abläufen als Möglichkeit zur Laufzeitüberwachung in einem Publish-Subscribe-System für die Entwicklung und Pflege von verteilten Programmen mit physikalischer Verteilung der Komponenten und Parallelität der laufenden Prozesse erfordert dabei andere Vorgehensweisen, als Anwendungen zur Überwachung monolithischer Programme ermöglichen.

Die Anzahl der zu überwachenden Komponenten bedeutet eine inhärente Komplexität, so dass diese zur Darstellung und Wahrung der Übersichtlichkeit und dem intuitiven Verständnis zu beachten ist.

[1] Aus Gründen der besseren Lesbarkeit wird bei Personenbezeichnungen und personenbezogenen Hauptwörtern in dieser Arbeit die männliche Form verwendet. Entsprechende Begriffe gelten im Sinne der Gleichbehandlung grundsätzlich für alle Geschlechter und beinhalten keine Wertung.

Die Visualisierung der Kommunikationsstrukturen dient der Klärung von komplexen Sachverhalten, um Zusammenhänge und Abläufe darzustellen. Die verwendeten Daten stammen aus verschiedensten Datensätzen (zum Beispiel Logdateien oder Listen). Ohne entsprechende visuelle Aufbereitung ist es schwierig, aussagekräftige Zusammenhänge zu erkennen. Durch das Veranschaulichen dieser Datenflut werden mit grafischen Mitteln fehlende Übersicht und mangelnde Einblicke komplexer Systeme kompensiert. Des Weiteren ist durch selektives und exploratives Nutzerverhalten auch die Größe solcher Systeme weniger belastend, als es ohne grafische Hilfe der Fall wäre. Die Fähigkeit, Subsysteme ohne die Belastung irrelevanter Informationen zu inspizieren, kann ungeachtet der Systemgröße lesbare visuelle Ergebnisse ermöglichen.

1.1 Motivation

Die Motivation zu dieser Arbeit ist das Auffinden von Fehlern und das Verstehen komplexer Kommunikationsstrukturen mittels grafischer Darstellungen zu erleichtern. Die Visualisierung kann bei der Entwicklung von Projekten, der Prüfung und der Überwachung des Systems eingesetzt werden. Mit Hilfe von Darstellung der Kommunikationsstrukturen ist eine Laufzeitüberwachung möglich.

Die Nutzung von Visualisierung als Möglichkeit, in übersichtlicher Form eine große Menge an Informationen zu präsentieren, bietet sich durch die kognitiven und perzeptuellen Fähigkeiten des Menschen an.

Jarig Richter-Peill schreibt in „Visualisierung der Interaktion in Multiagentensystemen“: „Visualisierung kann als Externalisierung von Gedankenbildern begriffen werden.“ [18, Seite 6]

Die in verteilten Systemen zunehmende Komplexität mindert die Handhabbarkeit und das Verständnis der Strukturen. Durch Visualisierung der Strukturen sollen kognitive Bilder beim Betrachter erzeugt werden, die das Verstehen fördern und somit die Komplexität handhabbar werden lassen [18].

1.2 Ziele

Ziel ist eine sinnvolle Visualisierung von internen Abläufen beziehungsweise Kommunikationsstrukturen zu schaffen, wobei die Kommunikationsstrukturen in dieser Arbeit als Graphen aufgezeigt werden.

Sinnvoll meint den Einsatz von bedarfsorientierten Filtern zur Darstellung der für den Anwender relevanten Informationen. Wie beispielsweise die Kommunikation zwischen zwei Diensten anzuzeigen, ohne das gesamte verteilte System zu visualisieren.

Die Filter dienen dem Zweck der Übersichtlichkeit und Lesbarkeit des Graphen-Layouts und sollen so intuitive Interpretation und Nutzung ermöglichen.

In einem Publish-Subscribe-System existieren Gruppen, die von Teilnehmern, welche Agenten genannt werden, abonniert werden können (subscribe). In diesen Gruppen werden von weiteren Agenten Daten veröffentlicht (publish) und an die abonnierenden Agenten der Gruppe weitergegeben. Relevante Daten zur Kommunikation wie die Informationen, welcher Agent zu welcher Zeit mit welchen anderen Agenten über eine Gruppe kommuniziert, was kommuniziert wird, und welche Agenten zu der Zeit noch lebendig sind, sollten grafisch dargestellt werden.

Die relevanten Ereignisse der Kommunikation sollten aus dieser grafischen Darstellung hervorgehen beziehungsweise gefiltert werden können.

Die Visualisierung soll auf eine lesbare Art und Weise geschehen. Das bedeutet zum Beispiel, dass die Anordnung des Graphen-Layouts in der Visualisierung bei Änderungen, so weit möglich, erhalten bleiben sollte. Durch die gleichbleibende Anordnung soll das Wiederfinden von Elementen und das Verfolgen von Abläufen erleichtert werden. Zur übersichtlichen Darstellung ist es erforderlich abzuschätzen, welche Abläufe im System für die jeweilige Untersuchung beachtet werden sollen. Diese Abschätzung ist für die Lesbarkeit und für die Reduzierung der Häufigkeit des Renderns von Vorteil.

In diesem Publish-Subscribe-System ist es notwendig, folgende Entitäten abbilden zu können: Gruppen, Nachrichten und Agenten.

Das Ziel ist es, eine möglichst kreuzungsfreie Übersicht der Kommunikationsstrukturen mit den beteiligten Entitäten abzubilden. Wobei die effiziente Nutzung von Layout-Algorithmen zur Minimierung von Überschneidungen der Kanten, durch den Wunsch des Erhalts der Anordnung von Entitäten, schwer möglich ist und eine Herausforderung darstellt.

Die Entitäten müssen in jedem Fall klar optisch unterscheidbar sein. Anforderungen an die Darstellung sind dabei beispielsweise Kästchen für Agenten, Kringel für Gruppen, und für Nachrichten unidirektionale Pfeile, um „publisher“ (publizierender Agent) von „subscriber“ (abonnierender Agent) unterscheiden zu können.

1.3 Gliederung der Ausarbeitung

Im Grundlagenkapitel (siehe Kapitel 2) werden die Aspekte von verteilten Systemen betrachtet, die für das Verständnis der Funktion und des Aufbaus des zu untersuchenden Systems und seiner Herausforderungen bezüglich Debuggings und Laufzeitüberwachung von Interesse sind.

Des Weiteren werden Kommunikationsmodelle verteilter Systeme umrissen, insbesondere das Publish-Subscribe-Modell. Darauf folgend wird das Konzept von Agenten und Multiagentensystemen vorgestellt. Am Beispiel der verwendeten Middleware werden ihr Prinzip und ihre Funktion erklärt. Diese Middleware ist das Herzstück des zu untersuchenden Systems.

Dem Grundlagenkapitel folgt die Analyse (siehe Kapitel 3). Im Analysekapitel werden alle beteiligten Entitäten erläutert, Schnittstellen des Systems beschrieben und Funktionen und Eigenschaften des Visualisierungstools untersucht. Auf dieser Basis und durch die Beschreibung von Laborszenarien wird ermittelt, welche Anforderung an das Visualisierungstool bestehen.

Im Kapitel „Design und Realisierung“ (siehe Kapitel 4) wird die Umsetzung der Entwicklung des Prototyps unter Berücksichtigung der im Analysekapitel erarbeiteten Anforderungen beschrieben. Es folgt die Vorstellung und Rechtfertigung der Auswahl der verwendeten Sprachen, Bibliotheken, Frameworks, Algorithmen und Mechanismen. Die Aufbereitung und der Transfer der Daten, die Darstellung des Programmaufbaus und die Betrachtung der Programmkomponenten wie Frontend, Backend und Schnittstellen werden ebenfalls in diesem Kapitel behandelt.

Die Effizienz des Designs und die Erfüllung der Umsetzung der Anforderungen werden im Fazit des Kapitels „Design und Realisierung“ untersucht.

Das Schlusskapitel (siehe Kapitel 5) fasst diese Arbeit kurz zusammen und gibt einen Ausblick auf mögliche Erweiterungen dieser Arbeit.

2 Grundlagen

In diesem Kapitel werden Grundlagen und Begriffe erläutert, die zum Verständnis der Funktion und der Struktur des zu visualisierenden Systems benötigt werden. Das Verständnis soll die Nachvollziehbarkeit der resultierenden Anforderungen an das Visualisierungstool im folgenden Kapitel erleichtern. Zusätzlich werden Begriffe im Kontext des genutzten Publish-Subscribe-Systems erklärt. Als Visualisierungstool dient eine Webapplikation zur Graphenvisualisierung. Im Kontext dieser Arbeit sind die beiden Begriffe „Visualisierungstool“ und „Webapplikation“ als Synonym zu sehen.

2.1 Verteilte Systeme

In der Fachliteratur existieren verschiedene Definitionen von verteilten Systemen. Andrew S. Tanenbaum charakterisiert ein verteiltes System lose als eine Sammlung von autonomen Rechenelementen, die seinen Benutzern als ein einziges kohärentes System erscheint. Ein autonomes Rechenelement kann dabei ein Hardware-Gerät oder ein Software-Prozess sein. Jedes dieser Elemente ist in der Lage, unabhängig von den anderen zu agieren. Das System soll dabei für den Nutzer, sowohl Mensch als auch Anwendung, kohärent wirken. Das bedeutet, dass Nutzer glauben, es mit einem einzigen System zu tun zu haben.

Alle Elemente sind also autonom und agieren auch autonom. Um gemeinsam ein System zu bilden ist es erforderlich, dass die beteiligten Akteure zwecks Kollaboration interagieren. Durch die wie auch immer geartete Interaktion wird ebenfalls die Nutzung von gemeinsamen Ressourcen beziehungsweise der erleichterte Zugriff auf lokale und entfernte Ressourcen ermöglicht. Da in verteilten Systemen meistens kein gemeinsamer lokaler Speicher vorhanden ist, erfolgt die Kommunikation der agierenden Prozesse durch den Austausch von Nachrichten.

Neben den wünschenswerten Eigenschaften eines verteilten Systems wie loser Kopplung, Heterogenität und Transparenz, zeichnet es sich unter anderem auch durch Offenheit, Skalierbarkeit und Nebenläufigkeit aus [19].

Es existieren noch weitere Eigenschaften, die verteilte Systeme charakterisieren, die jedoch für das hier zu betrachtende System beziehungsweise die Visualisierung seiner Kommunikationsstrukturen zu vernachlässigen sind.

Die kohärente Wirkung eines verteilten Systems ist für die Visualisierung der Kommunikationsstrukturen wenig relevant. Von Interesse ist, dass die Akteure sich nicht zwingend bekannt sind, und dass die Interaktion der Akteure in verteilten Systemen via Nachrichtenaustausch erfolgt.

2.2 Kommunikationsmodelle

In verteilten Systemen wird von verschiedenen Kommunikationsmodellen ausgegangen. Es wird beispielsweise zwischen synchroner und asynchroner Kommunikation unterschieden.

Bei dem synchronen Kommunikationsmodell blockieren Sender und Empfänger beim Aufruf der Sende- beziehungsweise Empfangs-Operation. Sobald der Empfänger verfügbar ist, wartet er auf eine eintreffende Nachricht. Nach dem Senden der Nachricht wartet der Sender auf Antwort des Empfängers, der zuvor gesendeten Nachricht. Dieses einfach zu verstehende Modell ist eines mit hohen Abhängigkeiten, was insbesondere im Fehlerfall Nachteile mit sich bringt. Die synchrone Kommunikation bedeutet eine enge Kopplung zwischen Sender und Empfänger [23].

Das asynchrone Kommunikationsmodell ist nicht blockierend. Der Sender kann nach dem Senden einer Nachricht unmittelbar weiterarbeiten. Beim Empfänger werden Nachrichten gepuffert (je nach Implementierung), eine optionale Antwort kann vom Empfänger asynchron an den Sender geschickt werden. Diese meist effizientere Implementierung des Modells ist komplexer als die synchroner Kommunikation. Sender und Empfänger sind unabhängig voneinander, es ist nicht notwendig, dass der Empfänger bereit ist. Zwischen den Prozessen besteht eine lose Kopplung [23].

Ein Modell der asynchronen Kommunikation ist das Publish-Subscribe-Pattern, welches sich durch eine indirekte Kommunikation auszeichnet. Teilnehmer, die Informationen verbreiten beziehungsweise veröffentlichen möchten, melden sich in einer Gruppe an und veröffentlichen dort ihre Nachrichten. Die Gruppen können von anderen Agenten abonniert werden. Auf diese Weise bekommen alle Abonnenten der jeweiligen Gruppe alle Nachrichten, die in dieser Gruppe veröffentlicht werden. Dabei ist es nicht nötig, dass die Teilnehmer sich gegenseitig kennen oder über den Zustand anderer Bescheid wissen.

Durch dieses Kommunikationsmodell werden Heterogenität und eine lose Kopplung gewährleistet. Heterogenität einer Systemlandschaft zeichnet sich durch die Möglichkeit

der Interaktion von sich unterscheidenden Komponenten (Hard- und Software) aus. Um Nachrichten auszutauschen, muss die entsprechende Schnittstelle implementiert und im vorher vereinbarten Format kommuniziert werden. Die Kopplung bezeichnet den Grad der Abhängigkeit von Komponenten zueinander, dies schließt Hard- und Softwarekomponenten ein. Bei loser Kopplung eines Systems lassen sich Änderungen einzelner Komponenten einfacher durchführen, da die Änderungen nur lokale Auswirkungen haben [20] [19]. Eine Ausnahme bildet die Anpassung der Kommunikationsschnittstelle. Eine Anpassung dieser Art hat auf alle beteiligten Komponenten eine Auswirkung. Bei einer Anpassung der Schnittstelle müssen alle Komponenten angepasst werden, um eine Kommunikation weiter zu ermöglichen.

2.3 Agenten

Zum Agenten, auch häufig Software-Agenten genannt, existiert, wie auch bei verteilten Systemen, keine einheitliche Definition. Eine allgemeine Definition eines Agenten nach Wooldridge und Jennings beschreibt einen Agenten als Computersystem, welches in einer Umgebung situiert ist und in dieser autonom Aktionen ausführen kann, um seine Ziele zu erreichen. Dabei werden den Agenten verschiedene Eigenschaften wie Reaktivität, Proaktivität und Sozialfähigkeit zugeschrieben. Reaktive Agenten sind in der Lage ihre Umgebung wahrzunehmen und auf diese zu reagieren. Ein proaktiver Agent ist darüber hinaus fähig, Initiative zu ergreifen, um sein Ziel zu realisieren. Die Sozialfähigkeit von Agenten zeichnet sich durch die Interaktionen miteinander aus [24].

„Agenten können stationär in einer Systemumgebung beheimatet sein oder als mobile Agenten in andere Umgebungen migrieren. Kooperation und Lernfähigkeit sind weitere Eigenschaften, die mit Software-Agenten in Verbindung gebracht werden. Ein Multiagentensystem ist ein System aus gleichartigen oder unterschiedlichen Software-Agenten, die miteinander interagieren, um zum Beispiel kooperativ eine gemeinschaftliche Aufgabe zu bewältigen.“ [2, Seite 281]

2.4 Multiagentensysteme

Multiagentensysteme sind dynamische verteilte Systeme, die aus mehreren autonomen Agenten bestehen. Systeme dieser Art sind zumeist dezentral organisiert. Jeder Agent verwaltet seine Daten lokal und agiert autonom. Die Kommunikation erfolgt asynchron,

wodurch ein gefordertes Maß an Kontroll-Autonomie der Agenten gegeben ist. Dieses Verhalten hat Nebenläufigkeit zur Folge, da die Aktionen der verschiedenen Agenten parallel ablaufen. Die teilnehmenden Agenten besitzen nur unvollständige Informationen bezüglich des Gesamtsystems [15].

2.5 Middleware

Eine Möglichkeit, verteilte Systeme zu organisieren ist der Einsatz einer Middleware. Zur Entwicklung verteilter Anwendungen, werden verteilte Systeme häufig so organisiert, dass sie eine separate Softwareschicht aufweisen. Die Softwareschicht läuft logisch auf Hardware. Die Hardware ist Teil des verteilten Systems [19].

Eine Middleware ist eine im Hintergrund operierende Software-Komponente, die zwischen Betriebssystem und Anwendungen angeordnet ist. Sie stellt für alle Akteure die gleichen Schnittstellen bereit und ermöglicht auch bei Nutzung verschiedener Betriebssysteme oder heterogener Netze den Datenaustausch zwischen Anwendungsprogrammen.

Laut Oracle besteht der einfachste Weg zur Integration heterogener Komponenten darin, sie nicht als homogene Elemente neu zu erstellen, sondern eine Schicht bereitzustellen, die es ihnen ermöglicht, trotz ihrer Unterschiede miteinander zu kommunizieren. Diese als Middleware bezeichnete Schicht ermöglicht es Softwarekomponenten, die unabhängig voneinander entwickelt wurden und auf verschiedenen vernetzten Plattformen laufen, zu interagieren [17].

Die zu analysierende Kommunikation läuft vollständig über eine agentenbasierte Middleware. Das Nachrichtenformat, das die Middleware des zu untersuchenden Systems nutzt, ist JSON. Die hier beschriebene Middleware, deren Kommunikationsstrukturen dargestellt werden sollen, ist die Kernkomponente der Systemarchitektur der beiden Forschungslabore „Creative Space for Technical Innovations“ und „Living Place“ der Hochschule für Angewandte Wissenschaften Hamburg (siehe Kapitel 3.4). Die Middleware ist eine verteilte Server-Anwendung und bietet mit dem Publish-Subscribe-Dienst, der für die Gruppenkommunikation verantwortlich ist, eine Schnittstelle an, über die alle Teilnehmer miteinander kommunizieren.

„Teilnehmer“ werden im Folgenden synonym zu „Agenten“ behandelt.

Neben der Einrichtung der Kommunikation zwischen den Agenten stellen weitere Middleware-Komponenten weitere Funktionen zur Verfügung. Zu diesen Funktionen zählen die

Verwaltung und Überwachung von Agenten und die Kontrolle der Laufzeitumgebung [13][7].

Aus der Sicht eines Teilnehmers wirkt das System, mit dem er kommuniziert, homogen. Für den Teilnehmer ist nicht ersichtlich, ob die Systemlandschaft aus kompatibler Hardware besteht, oder ob die verschiedenen Softwarekomponenten in unterschiedlichen Sprachen geschrieben sind. Dieser Effekt ist für ein verteiltes System eine gewünschte Eigenschaft und fällt unter den Begriff Transparenz. Sie beschreibt den Umstand, dass für den Benutzer oder auch die Anwendung nicht merkbar ist, dass mit einem verteilten System interagiert oder auf diesem agiert wird. Das bedeutet, dass weder für den Benutzer, noch die Anwendung, relevant oder überhaupt ersichtlich ist, auf welche Art die Verteilung des Systems geschieht oder in welcher Sprache ein Programm geschrieben ist. Es wird dabei von einem transparenten Systemverhalten gesprochen [19].

Da das System aufgrund des Einsatzes in den Hochschullaboren oft für relativ kurze Arbeitsperioden genutzt wird, ist eine hohe Wiederverwendbarkeit aller im System befindlichen Komponenten und aller Schnittstellen (sowohl Soft- als auch Hardware) erstrebenswert. Eine lose Kopplung bedeutet unter anderem ein gekapseltes System, in dem lokale Modifizierungen einzelner Komponenten keinen Einfluss auf andere Komponenten im System haben, solange implementierte Schnittstellen unverändert bleiben. Lose Kopplung bedeutet auch, dass das System heterogen sein kann. Diese Eigenschaft ist für den Einsatzzweck notwendig, da die Projekt-, Abschluss- und Forschungsarbeiten, die in den Laboren entwickelt werden, frei von Einschränkungen sind. Die Interaktion mit den verschiedenen Sensoren sollte ungeachtet der Hardware in allen Kombinationen von den Agenten zu nutzen sein. Das System verlangt nach Offenheit, welche den Grad der Erweiterbarkeit eines Systems angibt. Komponenten sind aufgrund der Offenheit einfach hinzuzufügen, zu ändern oder zu entfernen. Aufgrund der Anforderungen ergibt sich, dass das System skalierbar sein muss, um bei steigender Anzahl von Nutzern eine konstante Funktionstüchtigkeit zu gewährleisten. Durch die nebenläufige Abarbeitung von Nachrichten der verschiedenen Agenten ergibt sich eine asynchrone Kommunikation (siehe Kapitel 2.2).

All diese Eigenschaften schaffen ein System mit dynamischen, schwer zu reproduzierenden Systemzuständen und sich stetig verändernden Kommunikationsstrukturen [7] [13] [8].

Weiteres zu dem Thema Middleware sowie aktuelle Debatten findet sich unter anderem in der Konferenzreihe Middleware [3].

3 Analyse

Das Publish-Subscribe-Pattern nutzende Multiagentensystem, dessen dynamische Kommunikationsstrukturen visualisiert werden sollen, wurde als Infrastruktur für zwei Forschungslabore entwickelt: Zum einen für Smart Environments und zum anderen für Human-Computer Interaction [8]. Aus diesen Einsatzzwecken entstehen eine Vielzahl von Anforderungen an das System. Das umgesetzte Design beschreibt ein System, das für kurze Entwicklungszyklen (Semesterprojekte/Abschlussarbeiten) ausgelegt ist und eine hohe Wiederverwendbarkeit aufweist. Damit bedarf es Heterogenität und loser Koppelung. Diese Eigenschaften stellen bezüglich der Überwachung und Übersicht weitere zu beachtende Faktoren für die Anforderung an das Visualisierungstool zur Graphenvisualisierung dar. Die Faktoren werden in diesem Kapitel betrachtet. Dazu zählen Eigenschaften wie Nebenläufigkeit, dynamische Existenz von Teilnehmern und die dem Publish-Subscribe-Pattern zugrunde liegende lose Kopplung.

Beim Publish-Subscribe-Modell wird ohne Kenntnis über Existenz, Identität oder Lokalität potenzieller Abonnenten publiziert. Den abonnierenden Agenten ist nicht bekannt, wer publiziert oder von wo publiziert wird. Im Unterschied dazu liegen bei anderen lose gekoppelten Kommunikationsmodellen nebenläufiger Prozesse Informationen zu Strukturen vor. Beispielsweise existieren beim Remote-Procedure-Call Stub und Skeleton, es sind die Aufrufstruktur und die Parameter bekannt. Weitere Beispiele sind Webservices oder CORBA, bei denen bekannt ist, mit wem kommuniziert wird, auch wenn nicht bekannt ist, wo sich die potentiellen Kommunikationspartner befinden.

Das ab hier folgend analysierte Modell orientiert sich an der Arbeit von Tobias Eichler [7][13][8].

Von Relevanz bei diesem Publish-Subscribe-Modell ist das Topic, hier Gruppe genannt, welches abonniert wird, um auf diese Weise die Nachrichten, die für den jeweiligen Agenten von Interesse sind, erhalten zu können. Nicht relevant oder bekannt ist, von wem oder von wo die Nachricht stammt. Bekanntheitsgrade und Aufrufstrukturen sind hier nicht statisch festgelegt, und jegliche Interaktion findet durch Kommunikation auf Nachrichtenbasis statt. Autonom wird von einem Teilnehmer publiziert. Ob und wie ein anderer Teilnehmer darauf reagiert, ist nicht ersichtlich, weder für den publizierenden Agenten noch für andere Teilnehmer des Systems. Damit erreicht das Publish-Subscribe-Modell auch gegenüber anderen asynchronen Kommunikationsmodellen einen niedrigeren Grad

der Abhängigkeit und damit eine lose Kopplung. Unter diesen Bedingungen die Kommunikationsstrukturen zu verfolgen, stellt hohe Herausforderungen dar, da keine Kenntnis zu anderen Teilnehmern auf statischer Ebene existiert.

Um diesen Herausforderungen effizient zu begegnen ist Visualisierung als abstrakte Darstellung von Kommunikationsstrukturen und Informationen im Allgemeinen ein mögliches Werkzeug.

Colin Ware verdeutlicht in dem Buch „Information Visualization: Perception for Design“ anhand eines Beispiels eine Reihe von Vorteilen der Visualisierung [22, Seite 3-4, eigene Übersetzung]. Die beschriebenen Vorteile wurden wie folgt zusammengefasst:

- Unterstützung des Verstehens gewaltiger Informationsmengen
- Wahrnehmung von emergenten Eigenschaften, die nicht vorauszusehen sind
- Schnelles Entdecken von Fehlern in den visualisierten Daten, wodurch die Qualitätskontrolle unterstützt wird
- Förderung des Verständnisses des Charakters der Daten auf und zwischen allen Abstraktionsebenen
- Förderung der Hypothesengenerierung

3.1 Entitäten

Im Folgenden Abschnitt werden die am Kommunikationsprozess beteiligten Entitäten vorgestellt. Es wird analysiert, wie die Entitäten miteinander kommunizieren und wie die Entitäten in der Webapplikation dargestellt werden können.

- **Agenten:** Agenten sind Softwarekomponenten, die autonom agieren. Sie können sich an der Middleware registrieren. Mit Hilfe des Publish-Subscribe-Dienstes können die Agenten über die Middleware kommunizieren. Jeder registrierte Agent kann in Gruppen Nachrichten publizieren. In jeder abonnierten Gruppe kann ein Agent die dort veröffentlichten Nachrichten empfangen.
- **Gruppen:** Gruppen dienen der asynchronen Kommunikation zwischen Agenten. Durch in Gruppen publizierte Nachrichten kommunizieren Agenten indirekt. Der Erhalt dieser Nachrichten ist durch ein Abonnement der entsprechenden Gruppe

möglich. In eine Gruppe kann publiziert werden, auch wenn diese keine Abonnenten hat. Diese Gruppe gilt zu diesem Zeitpunkt als nicht existent.

Eine Gruppe und ihre Existenz definiert Tobias Eichler in der Abhandlung „Scalable Context-Aware Development Infrastructure for Interactive Systems in Smart Environments“ folgendermaßen: „Groups exist if there is at least one subscriber. Messages sent to non existing groups are discarded.“ [8, Seite 148].

- **Nachrichten:** Die Nachrichten werden via Publish-Subscribe-basierender Kommunikation in einer Gruppe ausgetauscht. Das genutzte Nachrichtenformat ist JSON. Dieses Format ist für Menschen lesbar. Aufgrund der für Menschen verständlichen Lesbarkeit wird das Debuggen erleichtert. Um eine Nachricht zu versenden, wird diese in einer Gruppe veröffentlicht. Zum Erhalt einer Nachricht wird die entsprechende Gruppe abonniert.
- **Knoten:** Die Middleware wird zwecks Skalierbarkeit und Ausfallsicherheit auf mehreren Knoten ausgeführt. Dieser Aufbau ermöglicht es, eine geringe Nachrichtenlatenz zu gewährleisten. Die Laufzeitumgebungen der unterschiedlichen Agenten können ebenfalls auf verschiedenen Knoten laufen und bilden so einen Teil des verteilten Systems.

3.2 Dynamik der Existenz von teilnehmenden Agenten und Fehlerzuständen

Durch die in dem Multiagentensystem relevanten Eigenschaften besteht ein dynamisches Systemverhalten im verteilten System. Die Dynamik des Systems erfordert nicht nur die Betrachtung einer Momentaufnahme, sondern des Verhaltens über die Zeit. Die Middleware selbst ist dynamisch, unter anderem bezüglich ihrer Verteilung und Ausmaße. Die Middleware gewährleistet Offenheit, Nebenläufigkeit und eine Skalierung ohne relevante Performanceverluste. Das tut sie, in dem sie bei Bedarf neue Knoten akquiriert oder die Verteilung der Agenten neu arrangiert [7]. Da auch Transparenz gewährleistet wird, ist keinem der Teilnehmer bekannt wo er oder von ihm genutzte Dienste lokalisiert sind. Durch die Verteilung der Middleware auf verschiedene Knoten wird Ausfallsicherheit gewährleistet. Ein weiteres Kriterium ist die Fehlertoleranz, welches die Middleware zu einem gewissen Grad erfüllt. Durch die lose Kopplung ist ein Weiterarbeiten von Agenten möglich, wenn andere Agenten nicht arbeiten. Zur Skalierung oder für die Einrichtung der Kommunikation zwischen Agenten muss die Middleware über den Zustand der im

System vorhandenen Agenten Bescheid wissen. Mit Zustand gemeint ist, ob ein Agent noch existent ist. Es werden Heartbeat-Nachrichten verwendet, um zu prüfen, ob ein Agent noch existiert. Bei Bedarf kann die Middleware benötigte Agenten erneut starten [7]. Die Möglichkeit, Fehler zu maskieren beziehungsweise Agenten nach Ausfällen wieder herzustellen, steigert die Dynamik der Agenten zusätzlich. Die Agenten und ihre Umwelt verändern sich und ihren Zustand laufend. Eine Fehlersuche kann durch Visualisierung der dynamischen Strukturen erleichtert werden.

3.3 Unterstützung der Systementwicklung durch Visualisierung der Kommunikationsstrukturen

Die für die Darstellung der dynamischen Kommunikationsstrukturen notwendigen Elemente, Funktionen und Eigenschaften sind im Folgenden aufgeführt und erklärt. Die Dynamik der Existenz von teilnehmenden Agenten und Fehlerzuständen werden dabei berücksichtigt.

Die Wahl von Visualisierung zur Vermittlung komplexer Inhalte und Systemstrukturen als effizientes Mittel begründet Colin Ware folgendermaßen:

„The eye and the visual cortex of the brain form a massively parallel processor that provides the highest bandwidth channel into human cognitive centers. At higher levels of processing, perception and cognition are closely interrelated, which is the reason why the words ‚understanding' and ‚seeing' are synonymous." [22, Seite XVI]

Aufgrund der von Colin Ware beschrieben Fähigkeit des menschlichen Gehirns, visualisierte Daten besser verarbeiten zu können, bietet sich die Darstellung der Kommunikation von Systemkomponenten in einem Graphen-Layout an.

3.3.1 Analyse der Entitäten

- **Agenten:** Es ist wichtig für die Visualisierung der Kommunikationsstrukturen, die beteiligten Agenten der Gruppenkommunikation darzustellen. Zur lesbaren Darstellung bei steigender Anzahl der Teilnehmer sollen nur die zur betrachtenden Situation zugehörigen Agenten abgebildet werden. Die Agenten müssen optisch klar unterscheidbar von anderen Entitäten wie Gruppen oder Nachrichten sein.

Das System beziehungsweise Subsystem soll bezüglich seiner dynamischen Kommunikationsstrukturen zum aktuellen Zeitpunkt und darüber hinaus im aktuellem Zeitrahmen betrachten werden können. Ein Auswahl- oder Konfigurationsmenü für den Entwickler ist notwendig, um die zu fokussierenden Elemente zu filtern. Die Abhängigkeiten zwischen den explizit ausgewählten Agenten und Gruppen werden durch die Anwendung ermittelt.

- **Gruppen:** Eine Gruppe ist die Entität, über die kommuniziert wird. Sie muss in ihrer Darstellung optisch von den Agenten zu unterscheiden sein.

  Die Beziehungen zwischen Gruppen und Agenten soll durch unidirektionale Pfeile symbolisiert werden.

  Ein unidirektionaler Pfeil mit Ursprung in einer Gruppe und einem Agenten als Ziel kennzeichnet, dass dieser Agent diese Gruppe abonniert. Das Veröffentlichen von Nachrichten in einer Gruppe wird durch einen unidirektionalen Pfeil in die entgegengesetzte Richtung, vom Agenten zur Gruppe, dargestellt.

  Die Darstellung einer Gruppe hängt nicht von ihrer Existenz, sondern von ihrer Relevanz in der zu betrachtenden Situation ab. Wird eine Gruppe explizit in den Voreinstellungen ausgewählt, soll die Gruppe dargestellt werden. Bei relativer Betrachtung eines einzelnen Agenten sind genau die Gruppen für diesen Agenten von Relevanz, in die der Agent publiziert oder solche, die der Agent abonniert. Des Weiteren ist zu beachten, wie weit indirekte Interaktionspartner in der Darstellung berücksichtigt werden sollen. Hat eine Gruppe indirekt Einfluss auf das Verhalten des betrachteten Agenten, kann durch explorative Erweiterung des Graphen die Komplexität des Graphen gesteigert werden. Für dabei ausgewählte Gruppen und Agenten gilt das Gleiche für die Relevanz der darzustellenden Gruppen wie bei der initialen Auswahl der Entitäten. Den Erhalt einer lesbaren Übersicht trotz der Darstellung der notwendigen Komplexität der zu untersuchenden Situation soll durch optionale Manipulation des Graphen ermöglicht werden.

  Die Darstellung einer Gruppe ist nicht abhängig von ihrer Existenz (siehe Kapitel 3.1), wenn ein Agent dort Nachrichten veröffentlicht. Die Darstellung nicht empfangener oder ungehörter Nachrichten ist von Nutzen für die Systemanalyse und kann notwendig zum Finden von Fehlern sein. Eine „nicht existente Gruppe“ in die publiziert wird soll dargestellt werden, wenn die Gruppe von Relevanz im zu betrachtenden Kontext ist.

- **Nachrichten:** Nachrichten sollen als unidirektionale Pfeile dargestellt werden, deren Richtung Auskunft darüber gibt, ob eine Nachricht versendet oder empfangen wird. Die Intensität des Nachrichtenflusses soll via zunehmender Strichstärke und Index am publizierenden Pfeil dargestellt werden. Der Index gibt dabei die Anzahl der vom Agenten veröffentlichten Nachrichten innerhalb eines vordefinierten Zeitintervalls an.

  Heartbeat-Nachrichten: Eine Heartbeat-Nachricht wird genutzt, um die Existenz eines Agenten zu überwachen oder das Existenzende beim Ausbleiben einer solchen Benachrichtigung festzustellen. Heartbeat-Nachrichten, die zwischen Middleware und den an der Middleware registrierten Agenten ausgetauscht werden, sollen nicht dargestellt werden. Auch wenn die Middleware eine entsprechende Schnittstelle anbieten würde, wäre eine direkte Darstellung der Heartbeat-Nachrichten nicht von Interesse. Misserfolg und Erfolg des Austauschs von Heartbeat-Nachrichten soll durch das Verschwinden oder die Anwesenheit von Teilnehmern indirekt dargestellt werden. Sonstige Betrachtungen dieser Art der Kommunikation zwischen Middleware und Agenten sind irrelevant, da ein verstorbener Agent nicht mehr in der Liste der registrierten Agenten auftaucht, die von der Middleware auf Anfrage geliefert werden kann. Ein weiterhin lebendiger oder auch ein von der Middleware neu gestarteter Agent bleibt in der Liste erhalten. Ist ein betrachteter Agent auch im nächsten Plot des Graphen noch vorhanden, war der Austausch der Heartbeat-Nachrichten erfolgreich oder der Agent wurde, bei Misserfolg und entsprechendem Bedarf, von der Middleware im Rahmen der Fehlermaskierung neu gestartet. Ist der Austausch von Heartbeat-Nachrichten erfolglos und ein erneutes Starten des betroffenen Agenten von der Middleware ist nicht vorgesehen, sollen die entsprechenden Agenten aus dem Graphen und dem Konfigurationsmenü verschwinden.

- **Knoten:** In der Webapplikation zur Graphenvisualisierung sollen die Knoten nicht mit dargestellt werden. Die Darstellung der Knoten würde zum einen der Lesbarkeit schaden, und zum anderen ist die Betrachtung der Verteilung der Middleware, Agenten und Gruppen auf unterschiedliche Knoten nicht relevant für die Betrachtung der dynamischen Kommunikationsstrukturen.

3.3.2 Analyse der Funktionen

- **Filter:** In komplexen Systemdarstellungen dienen Filter der Einschränkung von Elementen, die in der Graphendarstellung visualisiert werden sollen. Durch die Einschränkungen der zu visualisierenden Entitäten wird der Fokus auf die zu untersuchenden Elemente gerichtet. Neben dem Einsatz von Filtern stehen auch andere Mittel zur Verfügung, die eine spezifizierte Ansicht von Subsystemen ermöglichen. Alternative Ansätze wie beispielsweise Clustering werden hier nicht weiter betrachtet.

  Durch Systemeigenschaften wie Dynamik und Transparenz entsteht ein Blackbox-Effekt. Für den Entwickler ist nicht ersichtlich, welche Entitäten Teil einer Situation waren, sind oder sein werden. Eine Selektion aller an einer Situation beteiligten Entitäten ist ihm somit nicht möglich. Der Entwickler sollte eine Auswahl der Agenten oder Gruppen über die Filter selektieren, die er konkret untersuchen möchte.

  Die Auswahl der implizit dargestellten Agenten und Gruppen ist Aufgabe der Webapplikation. Es sollte auch innerhalb des geplotteten Graphen möglich sein, zu filtern und zu modifizieren. Innerhalb eines bestehenden Graphen-Layouts sollte, unabhängig von den Graphen-Updates, ein Entfernen oder Hinzufügen von Agenten und Gruppen durch den Nutzer möglich sein. Diese Anforderung stellt die explorative Komponente des Visualisierungstools dar. Eine weitere Anforderung dieses Typs könnte das Hineinzoomen in Teilgraphen sein.

  Die Filterfunktion soll genutzt werden, um die Eigenschaften Lesbarkeit und Übersichtlichkeit zu schaffen beziehungsweise zu optimieren und explorative Untersuchungen des Systems zu ermöglichen.

- **Lauschen:** Auslesen von Nachrichten, also Inspizieren der Kommunikationsinhalte, nicht der Struktur der Kommunikation und ihrer Dynamik. Diese Anforderung ist keine Anforderung an die Visualisierung selbst, sie ist vielmehr ein optionales Feature, welches Einblicke in Details ermöglichen könnte, die nichts mit den dynamischen Strukturen der Kommunikation zu tun haben.

  Das verwendete Nachrichtenformat JSON, bringt neben großer Bekanntheit bei Studierenden und niedriger Lernkurve auch menschenfreundliche Lesbarkeit mit, und der Overhead kann klein gehalten werden. Die Möglichkeit, den Nachrichteninhalt auf Anfrage ausgeben zu lassen, könnte gegebenenfalls eine Erweiterung der

Visualisierung sein, da durch die Art des Nachrichtenaufbaus eine nachvollziehbare Inhaltsvermittlung möglich ist.

Von der Middleware verworfene Nachrichten können nicht ausgelesen werden, da deren Inhalt zum Analysezeitpunkt nicht mehr existent ist.

3.3.3 Analyse der Eigenschaften

- **Überschneidungsfreiheit:** Zur Erstellung und Darstellung des Graphen ist ein Algorithmus nötig, der maximale Überschneidungsfreiheit ermöglicht. Diese sollte auch bei Updates des Graphen gewährleistet werden. Eine geringe Anzahl sich kreuzender Kanten soll den Graphen übersichtlich und infolgedessen lesbar machen.

  Nur drei der vorgestellten Entitäten sollen auch visualisiert werden. Knoten mit im Graphen darzustellen hieße, dass zur Anordnung der Entitäten berücksichtigt werden muss, welche Gruppen und Agenten auf welchen Knoten liegen. Es wäre von Nachteil, Entitäten aufgrund ihrer physikalischen Verteilung an bestimmten Positionen im Graphen darstellen zu müssen. Dieser Umstand würde zu weiteren Einschränkungen bezüglich der möglichst kreuzungsfreien Anordnung des Graphen führen. Hinzu kommt der Aspekt der Skalierung. Da zwecks Skalierung Agenten und Gruppen jeder Zeit den Knoten wechseln können, ist das maximale Erhalten eines Graphen-Layouts von einem Plot zum nächsten sehr unwahrscheinlich und der zu erwartende Wiedererkennungswert minimal. Da der Darstellung von Knoten keine für die Kommunikationsstrukturen relevanten Informationen entnommen werden können, aber Übersichtlichkeit und Lesbarkeit durch die Visualisierung der Knoten beeinträchtigt wären, ist es Teil der Anforderung, die Knoten zu vernachlässigen.

- **Positionserhalt:** Der Erhalt der Positionen von Gruppen und Agenten im Graphen ist für die Wiedererkennung der Kommunikationsstruktur von Plot zu Plot wichtig. Bei einem Graphen, der in regelmäßigen Intervallen neu gezeichnet wird um die Darstellung der Kommunikationsstrukturen aktuell zu halten, ist ein Wiedererkennungswert der Graphentopologie von Plot zu Plot anzustreben. Eine nachvollziehbare Entwicklung der Topologie des Graphen ist Voraussetzung für eine Lesbarkeit, die Aufschluss über die Situation und ihren Verlauf gibt und somit Schlussfolgerungen zulässt.

Der gewählte Algorithmus zur Vermeidung sich kreuzender Kanten sollte auch die Anordnung der Elemente soweit möglich bewahren. Soweit möglich meint, den Graphen trotzdem maximal überschneidungsfrei zu halten und bei zu großem Ungleichgewicht der Bildaufteilung, die durch die Dynamik des Systems entstehen kann, neu anzuordnen.

Verschwindet ein Agent im Bild, sollte sich der Rest des Graphen im Layout nach dem Update nicht von dem vorangegangenen Plot unterscheiden. Taucht derselbe Agent ein paar Zyklen später wieder auf, wird entschieden, wo er positioniert wird. Bei Wiedererscheinen an der Stelle, an der er verschwunden ist, müsste geprüft werden, ob sein aktueller Einsatzzweck im Kontext der Situation an der bisherigen Position noch sinnvoll wäre. Sollte der ursprüngliche Kontext bestehen, hätte das Wiedererhalten der letzten Position den Vorteil eines höheren Wiedererkennungswertes des Agenten-Knotens. Die notwendigen Maßnahmen um dieses Verhalten umzusetzen würden dauerhaft Plätze blockieren, was den Algorithmus beeinträchtigen kann. Die Tatsache, ob ein Agent im selben Kontext wieder auftaucht, ist nicht sicherzustellen. Aufgrund der Dynamik des Graphen ist es selbst beim kontextbewahrenden, erneuten Erscheinen eines Agenten möglich, dass ein Wiederauftauchen an seiner bisherigen Position visuelles Rauschen durch zu viele Überschneidungen verursacht. Dieser Effekt, wie auch die Beeinträchtigung des Algorithmus, könnte einen größeren Verlust der Übersichtlichkeit zur Folge haben als das Wiederauftauchen eines Knotens an anderer Stelle. Das Wiederauftauchen eines Knotens mit neuer Position hat gegebenenfalls einen geringeren Wiedererkennungswert der Entität zur Folge. Der Positionserhalt beeinträchtigt gegebenenfalls den Algorithmus und führt zu höherem Ressourcenverbrauch.

- **Erweiterbarkeit:** Explorative Analysen von Subsystemen werden ermöglicht, in dem der bestehende Graph modifiziert wird. Die Gruppen und Agenten, die im Konfigurationsmenü initial ausgewählt werden, sind explizit ausgewählte Entitäten. Explizite Entitäten sollen optisch unterscheidbar von impliziten Entitäten sein. Die im Fokus der Analyse stehenden explizit gewählten Gruppen oder Agenten werden mit ihren direkten Kommunikationspartnern (Agenten mit Gruppen und umgekehrt) dargestellt. Die impliziten Entitäten werden aufgrund der expliziten Auswahl ermittelt. Durch Anklicken der impliziten Kommunikationspartner sollen diese explizit gesetzt werden. Die Auswahl einer Entität als explizit soll ebenfalls über des Konfigurationsmenü möglich sein. Ein Wechsel des Wertes einer Gruppe oder eines Agenten von implizit zu explizit hat die Darstellung aller direkten

Kommunikationspartner dieser Entität zur Folge. Auf die gleiche Art kann der dargestellte Graph auch reduziert werden.

- **Übersichtlichkeit und Lesbarkeit:** Die Eigenschaften Kreuzungsfreiheit und Positionserhalt sind Voraussetzung für einen übersichtlichen Graphen. Übersichtlichkeit ist wiederum die Voraussetzung für Lesbarkeit des Graphen. Ein Wiedererkennungswert des Graphen von Plot zu Plot, klare optische Unterscheidung, aussagekräftige Symbole und gegebenenfalls eine Legende sollen die Lesbarkeit eines übersichtlichen Graphen ermöglichen.
- **Plattformunabhängigkeit:** Eine Webapplikation bietet die Möglichkeit, Informationen plattformunabhängig zur Verfügung zu stellen. Mit Hilfe der Webapplikation ist es nicht nötig zu wissen, auf welchem System die Informationen in Zukunft von den Benutzergruppen abgerufen werden.

3.3.4 Resultierender exemplarischer Entwurf

Die Grafik (Abbildung 1) zeigt einen ersten exemplarischen Entwurf der Webapplikation zur Visualisierung von Graphen. Dieser Entwurf soll das Zusammenspiel der beschriebenen Elemente und das Konfigurationsmenü unter Berücksichtigung der Anforderungen aufzeigen. Gruppen werden in diesem Entwurf als Hexagon dargestellt. Die kreisförmigen Objekte stellen in diesem Entwurf die Agenten dar. Der Nachrichtenfluss wird durch Pfeile zwischen Gruppen (Hexagone) und Agenten (Kreise) verdeutlicht. Die Richtung der unidirektionalen Pfeile gibt an, ob eine Nachricht in einer Gruppe veröffentlicht oder aus einer Gruppe versendet wird. Der Index an einem Pfeil kennzeichnet die Anzahl der Nachrichten, die in den letzten 5 Sekunden von dem Agenten in die Gruppe publiziert wurden. Das Konfigurationsmenü zeigt zwei Dropdown-Menüs, in denen Agenten und Gruppen angezeigt und ausgewählt werden können, wenn diese für die Middleware existent sind. Der Tracing-Slider wäre eine Möglichkeit, im Falle von Tracing innerhalb der Aufnahme einen bestimmten Zeit- oder Startpunkt auszuwählen. Der *Pause*-Button und der *Autoreload*-Schalter sollen ermöglichen, den Updatezyklus zu unterbrechen. Für das Feature Lauschen wäre bei Umsetzung ebenfalls eine Form von Eingabefeld notwendig, um das Feature zu aktivieren. In diesem Entwurf wird das Eingabefeld als *Listen*-Schalter gezeigt.

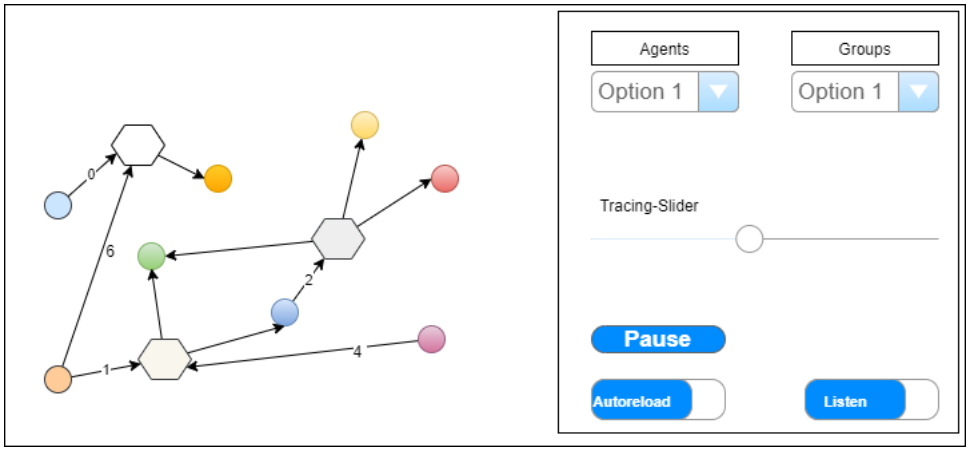

Abbildung 1: Mockup Frontend

3.4 Forschungslabore - Laborszenarien

CSTI - Creative Space for Technical Innovations

Der Creative Space for Technical Innovations (im Folgendem mit CSTI abgekürzt) ist ein Forschungs- und Testlabor für Virtual und Augmented Reality der Hochschule für Angewandte Wissenschaft Hamburg (im Folgendem mit HAW Hamburg abgekürzt). Hier wird die bereits vorgestellte Middleware eingesetzt, und Studierende können durch sie sämtliche im Labor vorhandene Hardware, Software und Sensorik über die Schnittstellen für ihr eigenes Projekt nutzen. Wobei oben genannte Herausforderungen der nicht ersichtlichen und sich stets im Wandel befindlichen Kommunikationsstrukturen, die Überprüfung von Funktionen und das Finden von Fehlern schwierig gestalten [10].

Living Place

Das zweite Forschungs- und Testlabor, Living Place (im Folgendem mit LP abgekürzt), ist ein Smart Home auf dem Campus der HAW Hamburg. Auch hier wird die Middleware eingesetzt und bietet den Studierenden die Möglichkeit, bereits vorhandene Installationen zu nutzen oder zu erweitern. Es bestehen im LP die gleichen Schwierigkeiten und Herausforderungen bezüglich der Einsicht in das System wie im CSTI [11].

3.4.1 Nutzergruppe

Nutzergruppe des Visualisierungstools sind Studierende, welche sich mit einem komplexen System auseinandersetzten müssen. Die Komplexität des Systems zeichnet sich durch seine Dynamik und die gegebene Publish-Subscribe-Architektur aus, in Folge derer die miteinander kooperierenden Agenten eine lose Bekanntschaft pflegen. Es besteht ein Bedarf eines Visualisierungstools zur Gewinnung eines Grundverständnisses des Systems, zur Überwachung der Kommunikationsstrukturen und zum Finden von Fehlern bei unerwartetem oder fehlerhaften Systemverhalten. Durch den Einsatz des Tools ist bereits während der Entwicklung des entstehenden Projektes die neu geschaffene Kommunikation im bestehenden System visualisierbar. Das klassische Debugging zur Prüfung des Codeflusses, Unit- oder Komponententests sind nicht die alleinigen Werkzeuge zur Prüfung der Funktionen.

3.4.2 Laborszenarien

Im Folgenden werden zwei Fälle unterschieden, die den Nutzen des Einsatzes eines Visualisierungstools für Kommunikationsstrukturen in einem Publish-Subscribe-System verdeutlichen sollen. Beide Szenarien geben den Einsatz im Forschungs- und Hochschulkontext wieder. Es existieren neben den vorgestellten Laboren noch weitere Forschungsbereiche der Informatik an der HAW Hamburg, das Augenmerk gilt in dieser Arbeit den Einsatzorten der Middleware dem LP und dem CSTI

Szenario Nummer 1 beschreibt Semesterprojekte. Diese Projektform bietet den Studierenden die Möglichkeit, eigenständig innerhalb des gewählten Themengebiets eine sich über das Semester erstreckende Projektarbeit zu entwickeln.

In den vorgestellten Forschungslaboren LP und CSTI finden jedes Semester eine Vielzahl an Projektarbeiten statt. Die Studierenden müssen die Infrastruktur des Labors, die Funktionsweise der Middleware und der optional zu nutzenden Aktorik und Sensorik kennenlernen. Häufig bauen die Semesterarbeiten von Studierenden aufeinander auf, die Projekte vorangegangener Semester müssen für das Entstehen aufbauender Projekte ebenfalls verstanden werden. Die Entwicklungszeit in einem Semester bietet nur eine begrenzte Zeit für eine aussagekräftige Dokumentation. Ein persönlicher Kontakt zu den Forschern älterer Projektarbeiten ist nicht immer möglich, wenn diese das Projekt bereits

verlassen haben. Durch den Einsatz eines Visualisierungstools kann ein Überblick über die in kurzen Entwicklungszeiträumen entstandenen Projekte geschaffen werden.

Die Studierenden müssen sich zu Beginn des Semesters mit der Infrastruktur des Labors vertraut machen. Hierzu muss der Studierende ein Verständnis für die Funktionen der Schnittstelle der Middleware erlangen und die Interaktionsmöglichkeiten mit dem Publish-Subscribe-Dienst kennenlernen.

Mit Hilfe eines Visualisierungstools kann die Vielfalt vorhandener Komponenten aufgezeigt und mittels grafischer Darstellung der Kommunikationsstrukturen der Projekteinstieg erleichtert werden. Der geschaffene Überblick über die Infrastruktur erleichtert es dem Studierenden realistische Projektziele zu definieren und eine Idee über den Umfang und die Richtung der eigenen Projektarbeit zu erlangen.

Bei der Einarbeitung und Entwicklung sind Verhaltensüberprüfung, erwartungskonforme Reaktionen und Ursachenermittlung von Fehlern beziehungsweise Kommunikationsausfällen von Interesse.

Zum Semesterende kann eine Projektpräsentation Teil der zu erbringenden Leistung sein. Hierbei könnte der Einsatz des Tools zur Demonstration der Kommunikationsweise des Projektes nützlich sein.

In **Szenario Nummer 2**, das eine Abschlussarbeit beschreibt, deckt sich der Einsatzbereich des Tools mit dem im Projektszenario.

Bei der Erstellung einer Abschlussarbeit besteht die Möglichkeit, dass der Studierende bereits aus Projektarbeiten in den Laboren die Kommunikationsstrukturen kennt und sich nicht erneut einarbeiten muss. Aufgrund der Komplexität von Abschlussarbeiten findet hingegen eine tiefere Einarbeitung in die Systemstrukturen der Labore statt. Die Komplexität erfordert, dass die Abschlussarbeiten die Schnittstellen berücksichtigen und die Arbeits- und Aufgabenverteilung der Komponenten grob bekannt sind, um beispielsweise Redundanz zu vermeiden.

Abschlussarbeiten finden nicht zwingend im Semesterrhythmus der Projektarbeiten statt. Die Erstellung der Abschlussarbeit kann mehr als ein Semester beanspruchen. Durch diesen Rhythmusunterschied können bei einer Überschneidung von Arbeiten Informationen zu Funktionen und dem Entwicklungsstand unter Studierenden über die Abschlussarbeit und Projekte ausgetauscht werden. Mit Hilfe des Visualisierungstools und dessen Einsatz zu Demonstrationszwecken könnte die Übergabezeit verkürzt werden.

In beiden Szenarien dient das Tool nicht nur zur Kontrolle bereits laufender Komponenten und deren funktionierendem Zusammenspiel, sondern auch zur Überprüfung des erfolgreichen Einbindens von Erweiterungen.

Ein weiterer Aspekt entsteht durch die gleichzeitige Nutzung des Systems von mehreren Nutzergruppen. Das System befindet sich stets im Wandel. Bei Projekten wie beispielsweise Abschlussarbeiten, die über einen längeren Zeitraum stattfinden, ist es nicht unwahrscheinlich, dass im System genutzt Komponenten/Agenten verändert werden.

Bei Änderungen anderer Komponenten mit Auswirkungen auf das Verhalten des eigenen Projektes kann das Visualisierungstool helfen, diese Änderungen zu identifizieren.

3.5 Schnittstelle und Formate

Die von der Middleware angebotenen Schnittstellen ermöglichen das Anmelden von Agenten an der Middleware (Registrierung) und das anschließende Nutzen der Publish-Subscribe-Kommunikation. Die Interaktion mit dem von der Middleware bereitgestellten Agenten (dem *middleware-interface-agent*), ermöglicht Anfragen zu den Kommunikationsteilnehmern und ihren Relationen.

Nach der Registrierung an der Middleware kann der Publish-Subscribe-Dienst genutzt werden. Der Publish-Subscribe-Dienst dient mit dem *middleware-interface-agent* als Informationsquelle, dieser Agent ist ebenfalls an der Middleware registriert. Durch Verwendung der API können Informationen zu registrierten Agenten oder Gruppen erfragt werden. Diese Informationen werden durch den *middleware-interface-agent* geliefert.

Übersicht der angebotenen Schnittstelle:

- Mit der Nachricht `msg GetRegisteredAgents()` wird eine Liste der aktuell registrierten Agenten erfragt.
- Mit der Nachricht `msg RegisteredAgents(agents: Seq[Agent])` wird die Liste der aktuell registrierten Agenten geliefert.
- Der Objekttyp Agent `msg Agent(id: String, name: String)` beinhaltet zwei Strings, die ID des Agenten und seinen Namen. Dieser Objekttyp wird als Sequenz von Agenten in der Liste *RegisteredAgents* übergeben.

- Mit der Nachricht `msg GetGroupList()` wird eine Liste der aktuell existierenden Gruppen erfragt.
- Mit der Nachricht `msg GroupList(groups: Seq[String])` wird die Liste der aktuell existierenden Gruppen geliefert.
- Mit der Nachricht `msg GetSubscriptions(filter: SubscriptionFilter)` wird eine Liste der aktuell existierenden Abonnements erfragt.

  Der Filter `msg SubscriptionFilter(agentId: Option[Int], agentName: Option[String], group: Option[String])` ist nicht implementiert, daher wird der Default genutzt und nicht gefiltert.
- Mit der Nachricht `msg SubscriptionList(subs: Seq[Subscription])` wird die Liste der aktuellen Abonnements geliefert.
- Der Objekttyp `msg Subscription(agent: Agent, group: String, apis: Seq [ApiId])` beinhaltet die Metadaten: Abonnierender Agent, abonnierte Gruppe und die verwendeten API-IDs für jedes Abonnement in der Liste.
- Der Objekttyp `msg ApiId(groupId: String, artifactId: String, version: String)` beinhaltet die Metadaten: ID der Gruppe, ID des Artefakts und die Version. Dieser Objekttyp ist Teil eines *Subscription*-Objektes.
- Mit der Nachricht `msg GetMessagesPublished()` wird eine Liste der aktuell veröffentlichten Nachrichten erfragt.
- `msg MessagesPublished(count: Seq[AgentMessagesPublished])` Mit dieser Nachricht wird eine Liste mit *AgentMessagePublished* Elementen geliefert, wobei sich der gelieferte Wert eines Elements auf die von einem bestimmten Agenten in einer bestimmten Gruppe veröffentlichten Nachrichten innerhalb der letzten 5 Sekunden bezieht.
- Der Objekttyp `msg AgentMessagesPublished(agent: Agent, group: String, count: Long)` in der von *MessagePublished* gelieferten Liste enthält einen Agenten, eine Gruppe und einen Zähler. Innerhalb eines 5 Sekunden Zeitintervalls werden die Nachrichten gezählt, die von einem Agenten in einer Gruppe publiziert werden.

Events:

Folgende Events werden vom *middleware-interface-agent* als Nachricht in der Gruppe *middleware-reply* publiziert. Um über diese Events informiert zu werden, muss die Gruppe *middleware-reply* abonniert werden.

- `msg AgentRegistered(agent: Agent)`: Wenn sich ein Agent an der Middleware registriert, wird ein Event ausgelöst.
- `msg AgentTerminated(agent: Agent)`: Wenn sich ein Agent an der Middleware abmeldet, wird ein Event ausgelöst.
- `msg NewSubscription(sub: Subscription)`: Wenn ein an der Middleware registrierter Agent eine Gruppe abonniert, wird ein Event ausgelöst.

3.6 Debugging-Techniken und Monitoring

Das Visualisierungstool ermöglicht Monitoring. Die Darstellung der Kommunikationsstrukturen zur Laufzeit kann zur Überwachung und zum grafischen Debuggen verwendet werden. Zu visualisieren, wer mit wem, wenn auch indirekt, kommuniziert, ist eine Möglichkeit zur Analyse.

3.6.1 Analyseoptionen - Multiagentensystem

In Multiagentensystemen kann nicht vorhergesagt werden, ob oder wann ein Agent am aktuellen Geschehen teilnimmt, die Struktur ist dynamisch. Das bedeutet, dass die Remodellierung einer Fehlersituation in einem verteilten System schwierig ist. Das gesamte System zu jedem Zeitpunkt an allen Stellen gleichzeitig zu betrachten stellt eine Herausforderung dar. Es ist möglich, einen Snapshot zu erstellen. Dieser zeigt jedoch kein Verhalten über die Zeit, sondern eine Momentaufnahme.

Aufgrund der Publish-Subscribe-Architektur sind Mechanismen synchroner Kommunikation zur Kontrolle erfolgreicher Nachrichtenvermittlung nicht nutzbar.

Im Gegensatz zur synchronen Kommunikation ist das Nachvollziehen der Kommunikation in verteilten Systemen schwieriger. Aufgrund der asynchronen Struktur ist nicht klar, ob ein Empfänger für den Empfang der Nachricht bereit ist oder der gewünschte Rezipient überhaupt die Gruppe abonniert hat, in der die Nachricht veröffentlicht wird.

Dem hier betrachteten, lose gekoppelten System, dessen Teilnehmer über einen Publish-Subscribe-Dienst kommunizieren, fehlt die Möglichkeit, wie bei der synchronen Kommunikation, zu erkennen, ob eine Nachricht erfolgreich zugestellt wurde.

Die durch lose Kopplung gewonnene Flexibilität führt zu Abstrichen im Überblick der Kommunikationsstruktur.

Teilnehmer können unabhängig voneinander publizieren und abonnieren. Es kann nur vermutet werden, dass ein Teilnehmer nicht richtig läuft oder nicht mehr da ist, wenn Nachrichten von ihm erwartet werden, aber keine Kommunikation zustande kommt. Die Middleware ist hierbei die einzige wissende Komponente. Die grafische Auswertung und Verarbeitung der vorhandenen Daten der „wissenden Middleware“ kann zur Analyse der dynamischen Abläufe und der Dynamik ihrer Teilnehmer im System genutzt werden. Diese Form des grafischen Monitorings ermöglicht ein Debugging der laufenden Kommunikationsprozesse.

3.6.2 Testen und Debuggen in Publish-Subscribe-Systemen

Während klassische Testverfahren das Testen isolierter Komponenten ausreichend abdecken, werden Kommunikationsstrukturen, Asynchronität und Nichtdeterminismus auf diese Weise nicht geprüft. Die Unterschiede beim Testen und Debuggen verteilter Anwendungen und dem klassischen Testen und Debuggen monolithischer Programme und Systeme werden durch das Fehlen einer einheitlichen globalen Zeit und dem Fehlen von globaler Ordnung von Ereignissen deutlich. In verteilten Systemen teilen sich die einzelnen Komponenten keinen gemeinsamen lokalen Speicher.

Eine Remodellierung von Systemzuständen zu einem bestimmten Zeitpunkt ist in verteilten Systemen durch das Fehlen einer gemeinsamen Zeit schwierig. Das Einführen einer logischen Zeit durch beispielsweise den Einsatz der Lamport Zeit (nach Leslie Lamport) ermöglicht unter anderem die Ermittlung einer Happened-Before-Relation. Diese Relation zur Ordnung von Ereignissen ermöglicht Momentaufnahmen des Systemzustands (Snapshots). Eine Beobachtung über die Zeit ist trotz der so gewonnenen gemeinsamen Zeitreferenz nicht gegeben [19]. Diese Art das System zu betrachten ist nicht von Vorteil, um die dynamischen Kommunikationsstrukturen des Publish-Subscribe-Systems aufzuzeigen. Nicht die Systemzustände zu einem bestimmten Zeitpunkt sind von Interesse, sondern wie in dem System agiert wird.

Die Middleware stellt den zentralen Kommunikationspunkt in einem dezentral organisierten Multiagentensystem dar und lässt Monitoring von Agenten durch die Nutzung der Publish-Subscribe-Struktur zu. Die Kommunikation aus Logdateien der einzelnen Agenten zu lesen ist nicht dienlich, da die Einträge in der Logdatei keine globale Zeit besitzen und es schwierig ist, die relevanten Daten zur Kommunikation zwischen einem Teil der Agenten zu filtern. Übersichtlicher als beispielsweise die Auswertung von Logdateien ist hier eine grafische Darstellung der Kommunikationsstrukturen des Publish-Subscribe-Systems. Der Fokus liegt darauf, einen Überblick der Abläufe zwischen den Agenten zu schaffen und nicht darauf, Abläufe in einzelnen Agenten zu überwachen oder zu testen. Dies sollte im Vorhinein durch modulare Testverfahren wie Unit-Tests erfolgen. Es hängt vom Bedarf ab, ob ein entsprechendes Monitoring live oder recorded (zum Beispiel Tracing) eingesetzt wird. Hier liegt der Fokus auf Live-Monitoring, zur Hilfe beim Verständnis der Arbeitsweisen, dem Kennenlernen der beteiligten Akteure, der Interaktion, dem Finden von Fehlern und dem Testen des Entwicklungsfortschritts des entstehenden Projekts.

Das Betrachten vergangener Abläufe via Tracing erfordert eine Datenbank zur Speicherung der Daten. Diese Form des Tracing kann sinnvoll sein, um aktuelles Verhalten mit dem bisherigen zu vergleichen oder nach einem Ausfall von Komponenten nachzuvollziehen, wer bis zum Fehlerfall auf welche Weise mitwirkend war.

3.7 Fazit der Analyse

Es besteht ein Bedarf an einer Form des Monitorings, das die Kommunikationsstrukturen der beteiligten Agenten darstellt. Die Dynamik der Existenz von teilnehmenden Agenten und Fehlerzuständen kann durch grafisches Monitoring abgebildet werden. Dieses Monitoring ermöglicht trotz loser Kopplung und einer entsprechenden Blackbox-Betrachtung der jeweiligen internen Abläufe eine Darstellung der Kommunikationsstrukturen.

In diesem Kapitel wurde analysiert, welche Anforderungen im Schwerpunkt stehen, um eine Unterstützung der Systementwicklung durch Visualisierung der Kommunikationsstrukturen beim Software-Engineering und beim Reengineering zu ermöglichen.

Die in diesem Kapitel geforderten Rahmenbedingungen werden im Folgenden noch einmal zusammengefasst.

Das **plattformunabhängige** Visualisierungstool soll intuitiv und leicht verständlich in der Handhabung sein. Es soll übersichtliche und lesbare Ergebnisse liefern. Durch optisch klar unterscheidbare Entitäten (siehe Kapitel 3.3.1) und **Positionserhalt** soll die **Lesbarkeit** verstärkt werden. Es darf die **Übersichtlichkeit** aufgrund von Überschneidungen der Kanten im Graphen nicht vernachlässigt werden (siehe Kapitel 3.3.3), da aus einem Verlust der Übersichtlichkeit ein Verlust der Lesbarkeit resultiert. Die Anforderung **Erweiterbarkeit** soll durch die Möglichkeit, den bestehenden Graphen zu modifizieren, umgesetzt werden. Explorative Analysen der Strukturen sollen die Übersichtlichkeit des Graphen nicht beeinträchtigen.

Die gewählten Formen, Farben und Formate bieten klare optische Unterscheidungen der Entitäten und geben Aufschluss zum Kommunikationsverhalten.

Gerichtete Kanten sollen zum Beispiel die Richtung der Kommunikation mitteilen und mit ihrer Färbung darüber aufklären, ob eine Nachricht in eine Gruppe publiziert beziehungsweise die Gruppe abonniert wird (siehe Kapitel 3.3.1).

Die Nutzergruppen der Webapplikation zur Visualisierung der Kommunikationsstrukturen sind Studierende, die Projekt- oder Abschlussarbeiten in den bereits vorgestellten Laboren LP (siehe Kapitel 3.4 - Living Place) und CSTI (siehe Kapitel 3.4 - Creative Space for Technical Innovations) der HAW Hamburg entwickeln. Das grafische live Monitoring soll das Nachvollziehen der Systemstrukturen erleichtern. Die Visualisierung der Kommunikationsstrukturen soll die Verifizierung des Systemverhaltens, das Überprüfen aktueller Kommunikationszustände, das Verhalten der an dem Zustand beteiligten Aktoren und Sensoren sowie das Finden von Fehlern bei unerwartetem Systemverhalten erleichtern.

4 Design und Realisierung

Dieses Kapitel befasst sich mit dem Design und der Realisierung des Visualisierungstools unter Berücksichtigung der aus der Analyse hervorgegangenen Anforderung.

Das Visualisierungstool ist eine plattformunabhängige Webapplikation, die ihren Nutzern einen Überblick der dynamischen Kommunikationsstrukturen des zu untersuchenden Publish-Subscribe-Systems verschafft. Publish-Subscribe-Systeme stellen aufgrund ihrer Dynamik wegen den ständig wechselnden Teilnehmern, die unablässig ihre Kommunikationsstrukturen verändern, eine besondere Herausforderung bezüglich kontinuierlicher und gleichzeitig übersichtlicher Abbildung dar. In „Information Visualization: Perception for Design“ [22, Seite 4, eigene Übersetzung] wird der Prozess der Visualisierung in 4 Phasen zusammengefasst wie folgt beschrieben:

- Sammeln und Speichern der Daten
- Transformieren der Daten in eine für den Menschen verständliche Form
- Darstellen der Daten und Modifizierung der Darstellung
- Perzeption und kognitive Verarbeitung der Daten durch den Menschen

4.1 Visualisieren der Kommunikation (Graphendarstellung)

Die Dynamik des Systems hat einen ständigen Wechsel der Kommunikationsteilnehmer und ihrer Kommunikationsstrukturen zur Folge. Daher soll beobachtet werden können, wie sich Kommunikationsstrukturen bilden und verändern. Die Kommunikationsstrukturen sollen zudem inspiziert werden können, um eine Situation in eben diesem Moment im Detail zu betrachten. Dies geschieht, indem der Updatezyklus unterbrochen wird, um den aktuellen Ist-Zustand darzustellen.

Das Abbilden des aktuellen Zustands (Snapshot) erfordert, die verschiedenen Elemente voneinander differenzierbar darzustellen (siehe Kapitel 4.1.1). Inspektion und Analyse können auf aussagekräftige Darstellungen aufbauen, wenn Gruppen, Agenten, ein- und ausgehende Nachrichten und explizite sowie implizite Entitäten optisch unterscheidbar sind.

In Bezug auf die dynamische Veränderung der Kommunikation ist es von Interesse, dass der Graph nicht ständig massiven Veränderungen unterliegt, um ein Bewahren des Überblicks über die Zeit, in der die einzelnen Knoten relativ zueinanderstehen, zu erleichtern.

Das Ziel ist, die Graphentopologie zu erhalten und ohne permanente drastische Änderungen am Layout, die Dynamik des Systems zu visualisieren.

Visuelle Charakteristiken der Graphenelemente dienen einem verbesserten Überblick. Zur Förderung der Übersichtlichkeit sollen die Positionen der Elemente im Layout weitestgehend erhalten bleiben.

4.1.1 Darstellung von Agenten und Gruppen

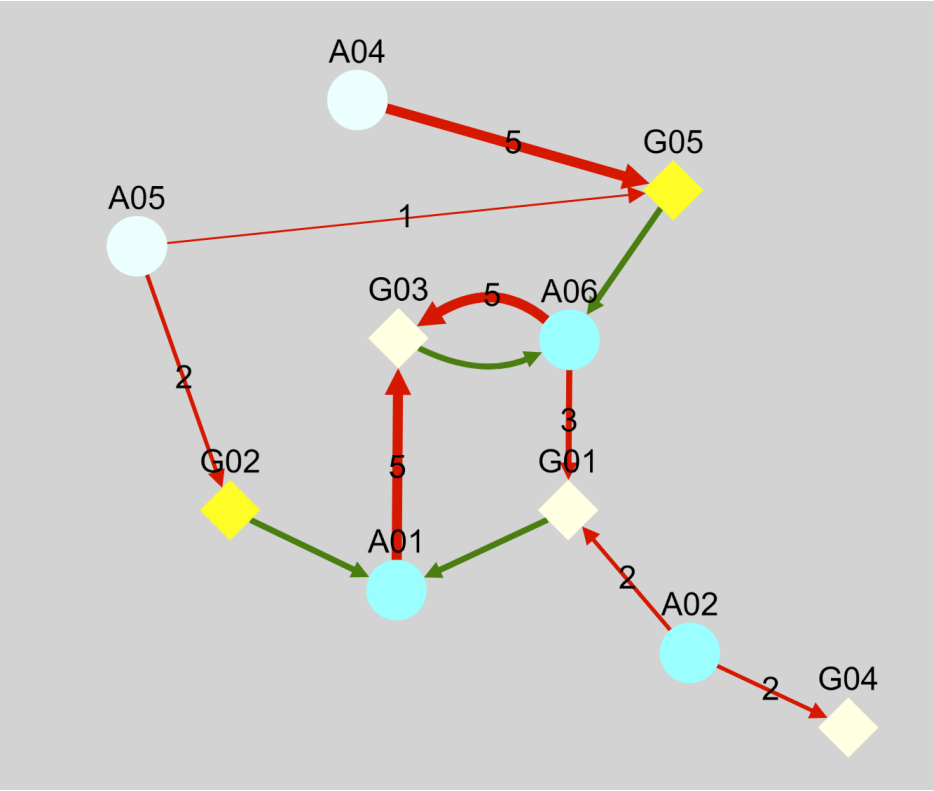

Abbildung 2: Kommunikationsstrukturen von Agenten und Gruppen

Nach den aufgeführten Anforderungen (siehe Kapitel 3.3) müssen die Gruppen von den Agenten optisch unterscheidbar sein. Dazu werden verschiedene Farben und Formen ein-

gesetzt. Die geometrische Form von den jeweiligen Entitäten ist einheitlich und klar von anderen Entitätstypen unterscheidbar. Das bedeutet, Gruppen werden in einer anderen geometrischen Form dargestellt als Agenten. Bezüglich der Farben gibt es ein Farbspektrum für Gruppen, das sich nicht mit dem Farbspektrum von Agenten überschneidet. Explizit ausgewählte Gruppen unterscheiden sich innerhalb des für Gruppen vorgesehen Farbspektrums von durch implizite Auswahl dargestellten Gruppen. Das gleiche gilt für Agenten und ihre Färbung. Während explizit ausgewählte Entitäten durch einen kräftigen Farbton gekennzeichnet werden, charakterisiert eine blasse Farbgebung die implizite Auswahl. Die Abbildung 2 zeigt wie Gruppen und Agenten dargestellt werden. Die Gruppen werden in Rautenform dargestellt und die Agenten als Kreise. Explizit ausgewählte Entitäten sind in kräftigen Tönen gefärbt, Gruppen in Blau und Agenten in Gelb.

Diese optische Unterteilung dient dem schnelleren Wiederfinden der im Fokus der Betrachtung stehenden Entitäten. Der Erhalt eines Layouts von Plot zu Plot, ist nur bedingt dauerhaft mit der Dynamik des Systems zu vereinbaren.

Ein einzelner, nicht mehr vorhandener Agent kann den Erhalt des Layouts nachteilig beeinflussen. Verschwindet ein Agent und hinterlässt eine oder mehr Gruppen auf die keine Referenz mehr existiert, wird auch die Gruppe entfernt, wenn diese nicht explizit ausgewählt wurde.

Bei der expliziten Auswahl eines Agenten werden die Gruppen des Agenten in den Graphen eingefügt, wenn diese noch nicht im Graphen vorhanden waren.

Die äquivalente Situation tritt beim Auftauchen und Verschwinden von Gruppen auf.

4.1.2 Kreuzungsfreiheit, Lesbarkeit und Übersichtlichkeit

Der Positionserhalt von Entitäten stellt eine Herausforderung in Bezug auf die Kreuzungsfreiheit für den Algorithmus dar. Es wird versucht, eine angemessene Balance zwischen dogmatischem Positionserhalt und Kreuzungsfreiheit zu finden.

Es ist möglich, Elemente nach persönlichen Vorlieben per Hand neu zu positionieren.

Die Darstellungsfläche nimmt Einfluss auf die mögliche Dauer des Positionserhalts der Elemente. Bei vielen Entitäten, unausgewogener Verteilung der Entitäten oder nach dem der Nutzer mittels *Redraw*-Button eine Neuzeichnung des Graphen anfordert, wird ein neues, dem gewählten Algorithmus zugrunde liegendes Layout erstellt.

4.1.3 Darstellung des Nachrichtenflusses

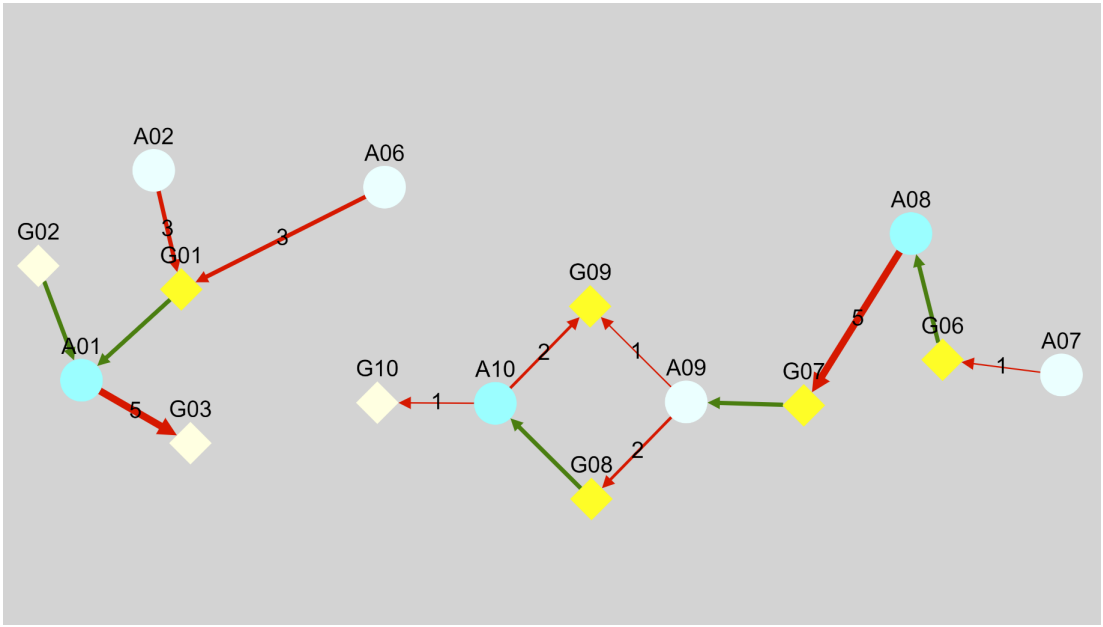

Abbildung 3: Darstellung des Nachrichtenflusses

Der Nachrichtenfluss wird (wie auf Abbildung 3 zu sehen) durch unidirektionale Pfeile symbolisiert. Ausgehende Nachrichten werden vom Agenten zur Gruppe durch einen roten Pfeil dargestellt.

Jeder Agent, der eine Gruppe abonniert hat, erhält eine Nachricht, sobald in dieser Gruppe eine Nachricht publiziert wurde. Der Erhalt der Nachricht wird durch einen grünen Pfeil von der Gruppe zum jeweiligen Agenten symbolisiert.

Von der Darstellung mindestens eines Pfeils (richtungsunabhängig) kann ausgegangen werden, wenn eine Gruppe existiert. Eine Gruppe ist erst im Konfigurationsmenü präsent, wenn diese existent ist. Als existent gilt eine Gruppe, wenn mindestens ein Agent diese Gruppe abonniert hat (siehe Kapitel 3.1). Die Existenz einer Gruppe und ihre explizite Auswahl oder mindestens eine Referenz auf diese Gruppe durch einen explizit ausgewählten Agenten sind Voraussetzung für ihre Darstellung mit mindestens einem Pfeil im Graphen.

Eine Ausnahme bilden nicht existente Gruppen. Die Middleware liefert über ihre Schnittstelle eine Liste *GroupList*, in der alle existenten Gruppen enthalten sind. Es werden nur Gruppen dargestellt, wenn diese für den Kontext relevant sind. Teil der Anforderungen

ist es, Gruppen, in die publiziert wird, darstellen zu können. Publiziert ein explizit ausgewählter Agent in eine nicht existente Gruppe wird diese Gruppe mit dargestellt (siehe Kapitel 3.3.1). In Abbildung 3 publiziert der Agent *A10* in die Gruppen *G09* und *G10*. Beide Gruppen sind nicht existent. Durch die explizite Auswahl der Gruppe *G09* ist zu erkennen, dass kein Agent diese Gruppe abonniert hat.

Eine weitere Anforderung ist es, explorative Visualisierung durch Modifizierung des geplotteten Graphen zu ermöglichen (siehe Kapitel 3.3.2). Es ist möglich, die nicht existente Gruppe, wenn sie dargestellt wird, in eine explizit ausgewählte Gruppe zu ändern, um zu prüfen, welche Agenten noch in diese Gruppe publizieren und diesen Nachrichtenfluss darzustellen (siehe Abbildung 3, Gruppe *G09*). Eine nicht existente aber explizite Gruppe kann auch ohne Referenz im Layout erhalten bleiben.

Die Zunahme der Breite eines roten (publizierenden) Pfeils verdeutlicht ein erhöhter Nachrichtenfluss (siehe Abbildung 3). Es kommen zusätzlich Indizes an den publizierenden Pfeilen zum Einsatz, um den Interpretationsspielraum der Breite gering zu halten. Die Indizes geben die Anzahl der von einem Agenten in einer Gruppe publizierten Nachrichten in den letzten 5 Sekunden an. Ein qualitativer Aufschluss über die quantitative Veränderung des Durchsatzes lässt sich über den Index ermitteln.

Der Nutzer erhält durch eine im Konfigurationsmenü abgebildete Legende zusätzliche Orientierung zum Verständnis der Entitäten (siehe Abbildung 4).

4.2 Graphenkonstruktion

Nach initialem Laden des Graphen erhält der Graph zyklisch Updates. Im Folgenden werden die beiden Fälle des initialen Ladens und der Updates eines bestehenden Graphen betrachtet.

4.2.1 Fall - Aufbau eines neuen Graphen

Ein Nutzer muss Vorbedingungen definieren, um einen Graphen zu konstruieren. Diese Vorbedingungen definiert der Nutzer im Konfigurationsmenü der Webapplikation (siehe Abbildung 4). Es können Agenten und Gruppen explizit ausgewählt werden. Die Nutzung von Dropdown-Menüs für die Präsentation aktuell aktiver Agenten beziehungsweise existenter Gruppen soll das Konfigurationsmenü übersichtlich gestalten. Die explizite

Auswahl von Entitäten hat nicht zur Folge, dass ausschließlich diese dargestellt werden. Die Auswahl vermittelt, welche Entitäten für den Nutzer von Interesse sind.

Dem Nutzer sind die Kommunikationsstrukturen unbekannt. Abhängig von der Größe des Systems kann es für den Nutzer schwierig sein, alle relevanten Entitäten für die zu untersuchende Situation explizit auszuwählen. Der Sinn der Webapplikation liegt darin, einen Eindruck der Strukturen zu erhalten und ihr dynamisches Verhalten beobachten zu können, um beteiligte Entitäten zu ermitteln, Verhalten zu verifizieren, Fehler zu finden oder Abläufe zu demonstrieren. Die im Konfigurationsmenü getroffene Auswahl ermöglicht die explorative Erstellung von Subsystemen und die Inspektion der zugrunde liegenden Strukturen.

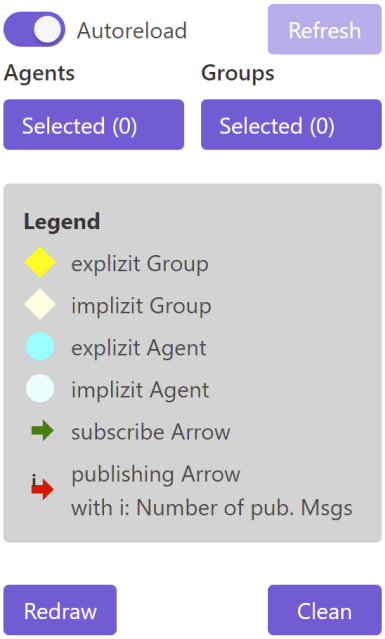

Abbildung 4: Benutzerschnittstelle - Konfigurationsmenü

Die Dropdown-Menüs werden kontinuierlich aktuell gehalten, so dass jeder an der Middleware angemeldete Agent und jede dort existierende Gruppe präsentiert und ausgewählt werden kann. Registriert sich ein Agent oder stirbt, wird auch das Auswahlspektrum des Dropdown-Menüs angepasst. Auf die gleiche Weise wird mit Gruppen verfahren.

Durch das Betätigen des *Clean*-Buttons (unten rechts in Abbildung 4) wird die bestehende Auswahl verworfen und der Graph gelöscht. Die Daten werden in der Benutzerkonfiguration neu ausgewählt und ein neuer Graph wird konstruiert.

4.2.2 Ermittlung der Abhängigkeiten

Im Frontend werden die Abhängigkeiten der ausgewählten Entitäten ermittelt (siehe Kapitel 4.4.3).

Die Abhängigkeiten der ausgewählten Entitäten sind die für die Betrachtung relevanten impliziten Entitäten des zu untersuchenden Subsystems.

Der aus den Vorbedingungen erfolgende Datensatz wird durch das Auswerten aller vorhandenen Daten ermittelt.

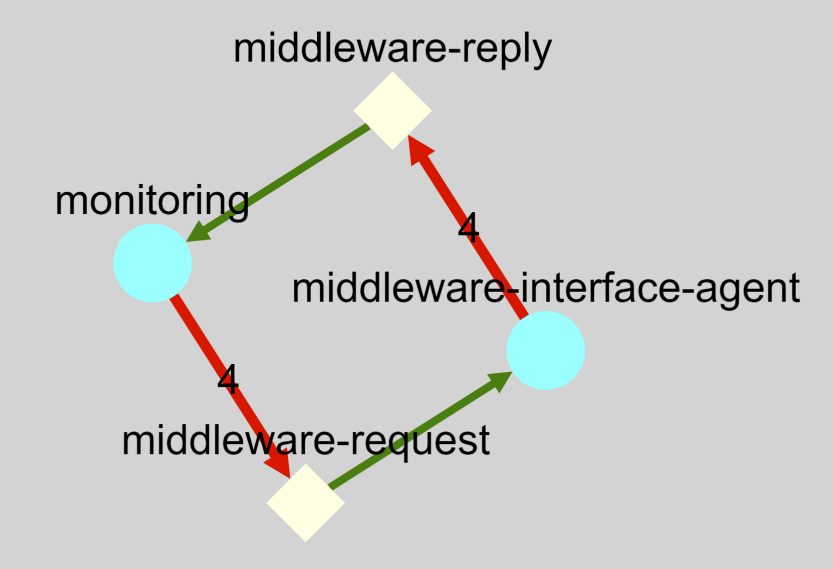

Abbildung 5: Kommunikation - Middleware und Backend

Die Daten werden vom Backend an der Middleware erfragt (siehe Abbildung 5). Zur Kommunikation mit der Middleware registriert das Backend einen Agenten, genannt *monitoring*-Agent, an der Middleware, wo der Agent die für ihn relevanten Gruppen abonniert. Zwei Gruppen sind für den *monitoring*-Agent von Interesse, die Gruppen *middleware-request* und *middleware-reply*. In der Gruppe *middleware-request* veröffentlicht der *monitoring*-Agent zyklisch, alle 5 Sekunden, eine Reihe von Anfragen für Listen. Die Gruppe *middleware-reply* wird vom *monitoring*-Agenten abonniert. In dieser Gruppe

werden von einem Agenten der Middleware, dem *middleware-interface-agent*, die Listen publiziert. Die Listen enthalten die Antwort auf die Anfrage des *monitoring*-Agenten in der Gruppe *middleware-request*. Die Listen geben Auskunft über an der Middleware registriert Agenten, existierende Gruppe, Publikationen und Abonnements (siehe Kapitel 4.4.2).

Der Grad der impliziten Darstellung ist eins. Explizit ausgewählte Entitäten (Agent oder Gruppe) werden mit ihren direkten Interaktionspartnern (ersten Grades) angezeigt.

Wurde beispielsweise Agent X explizit ausgewählt, werden alle Gruppen in denen Agent X publiziert und alle Gruppen, die Agent X abonniert hat, mit angezeigt. Wird eine der implizit dargestellten Gruppen explizit gesetzt, werden in Folge dessen alle in dieser Gruppe publizierenden Agenten sowie alle diese Gruppe abonnierenden Agenten mit dargestellt.

Der Wechsel des Attributs explizit/ implizit, ist mit einem Klick auf die jeweilige Entität möglich (siehe Kapitel 4.4.3).

4.2.3 Fall - Update und Modifizierung eines bestehenden Graphen

Der bestehende Graph erhält ein Update, alle 5 Sekunden. Der Updatezyklus kann durch die Deaktivierung der *Autoreload*-Funktion unterbrochen werden (siehe Abbildung 4).

Modifikationen am Graphen, zum Beispiel das Hinzufügen/Entfernen von Agenten, führen zu einem unmittelbaren Update unter Berücksichtigung des Algorithmus.

Ein bereits gezeichneter Agent bleibt solange er lebendig ist oder bis er abgewählt wird (direkt oder indirekt) Element des Graphen. Dabei wird die Position der Entität so lange wie möglich gehalten (siehe Kapitel 4.1.2). Der Positionserhalt beeinträchtigt den Algorithmus, der zwecks geringer Anzahl an Kreuzungen genutzt wird (siehe Kapitel 4.3).

Dauerhaftes Halten der Position ist keine Option, da das Kreuzen der Kanten gering gehalten werden soll, um den Überblick bewahren zu können.

Beim Darstellen bestimmter Kombination von Agenten und Gruppen kann das Hinzukommen neuer Agenten oder Gruppen dafür sorgen, dass es zu Überschneidungen kommt, da die bestehenden Agenten und Gruppen durch den Positionserhalt vermeintlich freie

Plätze blockieren. Dieser Effekt entsteht durch den Umstand, dass der gewählte Algorithmus zukünftige Veränderungen nicht voraussehen kann.

Ein Layout, dass die aktuellen Graphenelemente enthält und dem Algorithmus zugrunde liegt, ist erst wieder möglich, wenn die Positionen der bestehenden Gruppen und Agenten angepasst werden. Eine Erhöhung der Anzahl der Elemente verlangt ebenso eine andere Verteilung der Entitäten. Auf diese Weise ist gewährleistet, dass der Graph vollständig innerhalb des Displays abgebildet werden kann.

Es ist möglich, Kreuzungen händisch zu reduzieren, in dem ein Knoten vom Nutzer per Maus neu positioniert wird. Durch den Nutzer sind weitere manuelle Änderungen am Layout möglich.

Der Nutzer hat zu dem die Option, manuell ein neues Layout anzufordern, in dem der *Redraw*-Button (siehe Abbildung 4) betätigt wird. Mit dem Betätigen des Buttons werden die alten Positionen verworfen, und der Graph wird neu erstellt.

Mit aktivierter *Autoreload*-Funktion ist das Entdecken und Verfolgen der dynamischen Struktur möglich. Der Graph bildet alle im Betrachtungszeitraum und -kontext relevanten und aktiven Entitäten und Kommunikationen ab.

Zwecks Inspektion des Graphen (Snapshot-Analyse) ist ein Deaktivieren der *Autoreload*-Funktion möglich. Die Updatezyklen werden ausgesetzt, es wird das Plotten der Änderungen am Graphen verhindert, so dass dem Nutzer ein Snapshot des zu untersuchenden Moments präsentiert wird. Bei deaktivierter *Autoreload*-Funktion ist ein manuelles Anfordern der Änderungen am Graphen durch die Nutzung des *Refresh*-Buttons möglich (siehe Abbildung 4).

Änderungen am bestehenden Graphen sind via Nachkonfiguration durch den Nutzer möglich. Der Nutzer kann über die Dropdown-Menüs der Benutzerschnittstelle Agenten und Gruppen auch während der Betrachtung ab- und anwählen. Durch Klicken auf explizite Knoten (Gruppen und Agenten) kann der Nutzer ebenfalls den Graphen modifizieren, da auf diese Weise die explizite Auswahl aufgehoben wird. Der entsprechende explizite Knoten wird also durch das Anklicken implizit. Aufgrund der Darstellung von Interaktionen ersten Grades werden in jedem Fall alle nicht expliziten und nicht anderweitig referenzierten Interaktionspartner des angeklickten Knotens mit entfernt.

Wird ein Klick auf einen impliziten Knoten ausgeführt, wird dieser zu einem expliziten Knoten und alle noch nicht abgebildeten Interaktionspartner ersten Grades werden mit dargestellt.

4.3 Algorithmen zur Überschneidungsfreiheit

Cytoscape.js ist eine in JavaScript geschriebene Graphenbibliothek mit vielen Funktionen und einer großen Auswahl von Algorithmen zur Graphenerstellung. Sie ist open-source, vollständig serialisierbar und deserialisierbar via JSON und verwendet Layouts für die automatische und für die manuelle Positionierung von Knoten. Alle untersuchten Layout-Algorithmen mit gewünschten Eigenschaften unterscheiden sich in ihrer Tauglichkeit vor allem durch die Effekte die auftreten, wenn das Fixieren der Knotenpositionen von Layout zu Layout die Ausführung des Algorithmus beeinträchtigt beziehungsweise nicht möglich macht. Hauptkriterium an die Auswahl des Algorithmus ist, dass ein Positionserhalt durch ein Fixieren der Position wenigstens temporär umsetzbar ist, ohne den sofortigen Verlust der Übersichtlichkeit des zu betrachtenden Graphen.

4.3.1 Definition eines Layouts

Ein Layout ist eine Zuordnungsfunktion: Es bildet einen Knoten auf eine Position ab. Als Eingabe nutzt ein Layout eine Menge von Knoten und Kanten zusammen mit einer Reihe von Optionen [6].

Klassen von Layouts werden in die Hauptklassen diskrete und kontinuierliche Layouts unterteilt.

Das diskrete Layout berücksichtigt nicht den Wunsch nach Positionserhalt bestehender Entitäten. Bei Veränderungen der Datenbasis des Graphen wird der gesamte Graph neu gezeichnet und die Positionen der Entitäten neu festgelegt. Durch den fehlenden Positionserhalt ist das Neuberechnen des Graphen kostengünstiger zu realisieren, da keine dem Algorithmus abweichenden Informationen verarbeitet werden müssen. Die Knoten sind bei diskreten Layouts im Allgemeinen in einem geometrischen Muster angeordnet [6].

Diskrete Layouts sind entsprechend ungeeignet, da in der zu entwickelnden Anwendung dynamisch Knoten hinzugefügt werden. Aus der Dynamik der Kommunikationsstruktur

soll aber keine Neupositionierung aller vorhandenen Elemente resultieren. Die Kanten sollen trotz Positionserhalt möglichst kreuzungsfrei dargestellt werden, was mit Anordnung nach geometrischen Formen mit fixen Positionen nur zufällig gelingen kann.

Bei einem kontinuierlichen Layout werden die Knotenpositionen über mehrere Iterationen hinweg bestimmt, in der Regel durch einen force-directed-Algorithmus (kräftegesteuert). Bei jeder Iteration wird eine neue Knotenposition festgelegt. Während ein kontinuierliches Layout im Allgemeinen teurer ist als ein diskretes Layout, kann ein kontinuierliches Layout anspruchsvollere Ergebnisse liefern [6]. Die höheren Kosten aufgrund der Komplexität haben in dieser Anwendung keine merkbaren Unterschiede gezeigt. Zumindest bei heutiger durchschnittlicher Hardware sind keine Latenzen bei der Ausführung des Algorithmus im betrachteten Anwendungsfall spürbar. Das Layout wird dabei durch Anziehungs- und Abstoßungskräfte erzeugt, die auf Basis physikalischer Gesetze den Kanten und Knoten zugewiesen werden, wobei versucht wird, die Anzahl an Kreuzungen gering zu halten.

Als Unterklassen der diskreten und kontinuierlichen Layouts werden geometrische, hierarchische und force-directed-Layouts unterschieden, wobei die geometrischen Layouts der Klasse der diskreten Layouts zuzuordnen sind. Hierarchische Layouts, die sich für Bäume und DAGs (directed acyclic graph) eignen, zählen im Allgemeinen zu den kontinuierlichen Layouts, und auch force-directed-Layouts werden der Klasse der kontinuierlichen Layouts zugeordnet.

4.3.2 Vergleich von Algorithmen

Zum Vergleich wurde für geometrische, hierarchische und für eine Auswahl von potentiell geeigneten force-directed-Layouts eine Simulation genutzt, so dass jeder im Folgenden dargestellte Graph mit denselben Daten generiert wurde und sich ausschließlich durch den jeweilig genutzten Algorithmus unterscheidet.

circle

Das *circle*-Layout ordnet die Knoten im Kreis an - in der Reihenfolge, in der die Knoten zum Graphen hinzugefügt wurden. Wie auf Abbildung 6 zu erkennen ist, kann dabei nicht auf Kreuzungsfreiheit geachtet werden. Vielmehr ist das Layout dazu gedacht, zu visualisieren, von welchen Knoten besonders viele Kanten ausgehen. Ähnlich verhält es sich bei den anderen geometrischen Layouts bezüglich der Kreuzungsfreiheit. Damit sind diese Layouts für den in dieser Arbeit vorgestellten Anwendungszweck ungeeignet.

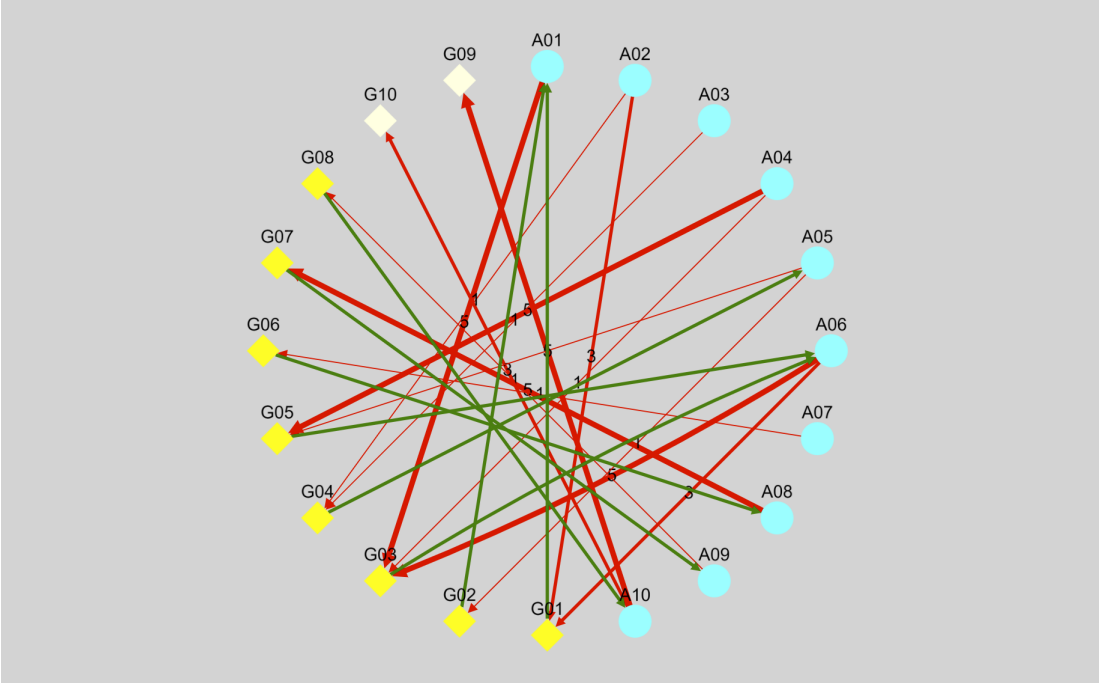

Abbildung 6: Layout circle Cytoscape.js

breadthfirst

Bei diesem Algorithmus wird das Layout über die Breitensuche bestimmt und die Knoten in Ebenen angeordnet [6]. Das Layout bietet durch die genutzte Breitensuche einen hohen Grad an Kreuzungsfreiheit, solange sich der Graph als Baum darstellen lässt. Dies gilt jedoch nur wenn nicht forciert wird, dass die Positionen der Knoten gehalten werden, wenn sich der Graph verändert. Die abzubildenden, dynamischen Kommunikationsstrukturen könnten in einer Baumstruktur auftreten, doch selbst in diesem Fall wäre nicht gewährleistet, dass trotz der Dynamik die Baumstruktur erhalten bliebe. Es ist zu erwarten, dass Kommunikationsstrukturen Zyklen aufweisen, da jeder Agent die Möglichkeit hat, über jede Gruppe zu kommunizieren und keine hierarchische Beziehung zwischen Knoten besteht. Dieser Algorithmus ist ebenfalls nicht für den gewünschten Einsatzzweck geeignet [4]. Wie im linken Teilgraph der Abbildung 7 zu erkennen ist, werden bei Graphen, die keiner Baumstruktur zugrunde liegen und Zyklen aufweisen, die Knoten unvorteilhaft angeordnet. Es gibt Kanten zwischen Knoten in einer Ebene, die auch zu weiter entfernten Knoten dieser Ebene reichen und nicht nur zu den direkt benachbarten, wodurch es zu Kantenüberlagerung kommt (siehe Abbildung 7, Gruppe *G01* zu Agent *A01* und die dazwischen liegende Überlagerung). Die Kantenüberlagerung erhöht den Effekt von visu-

ellem Rauschen eventuell weniger als Kantenkreuzungen, allerdings können überlagerte Kanten im schlimmsten Fall nicht wahrgenommen werden. Dieser Folgeschluss bezieht sich auf sämtliche getestete Layouts mit hierarchischem Aufbau.

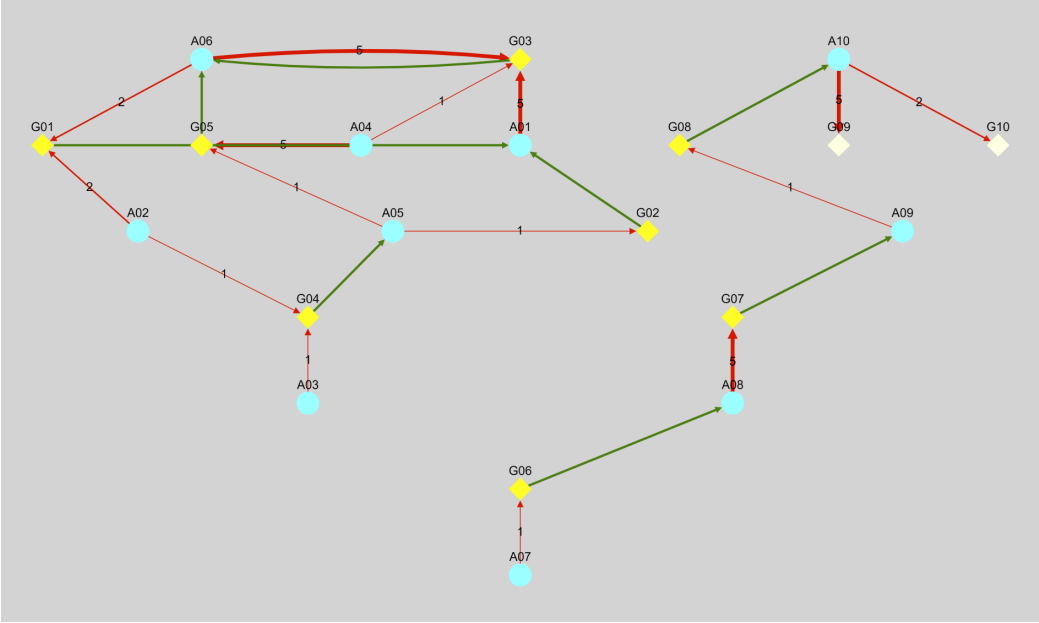

Abbildung 7: Layout breadthfirst Cytoscape.js

dagre

Wie auch der *breadthfirst*-Algorithmus ist der *dagre*-Algorithmus hierarchisch. Die Knoten werden auch hier in Ebenen angeordnet [6].

Bei der Verwendung des *breadthfirst*-Algorithmus kommt es zu Überlagerungen innerhalb einer Ebene. Bei dem hier verwendeten *dagre*-Algorithmus kommt es dazu, dass Kanten Ebenen überspringen. Dieses Überspringen reduziert das Auftreten von Überlagerungen, jedoch führt es zu einem vermehrten Auftreten von sich überschneidenden Kanten (siehe Abbildung 8, linker Teilgraph).

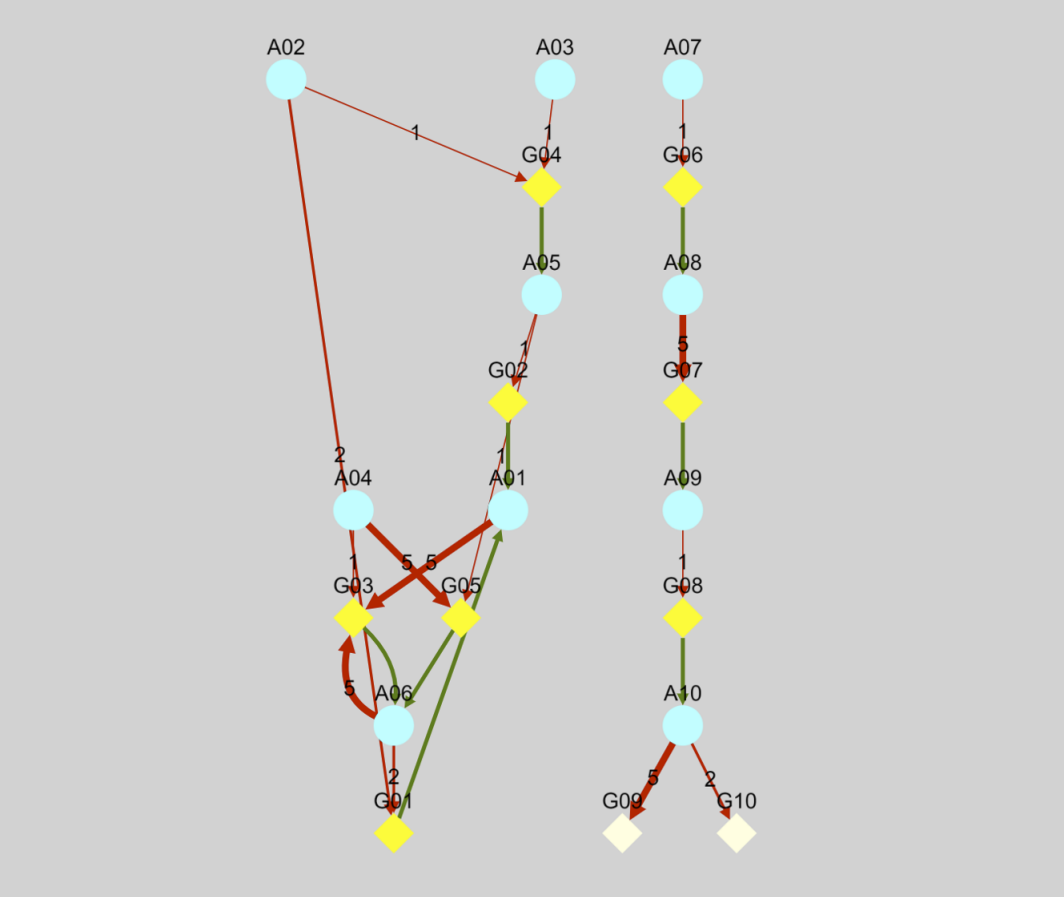

Abbildung 8: Layout dagre Cytoscape.js

cola

Das *cola*-Layout ist ein force-directed-Layout, das eine Vielzahl an Konfigurationsmöglichkeiten durch Nebenbedingung bietet, um die Kräftesteuerung nach Bedarf einzuschränken. Wie auf Abbildung 9 zu erkennen ist, liefert der Algorithmus sowohl beim unteren gerichteten azyklischen, als auch beim oberen gerichteten zyklischen Teilgraph gute Ergebnisse. Es gibt wenig Überschneidungen von Kanten, die Kommunikationsstrukturen zwischen den Agenten sind leicht ablesbar und Zusammenhänge erkennbar. Das hier sichtbare Ergebnis ist nur möglich, wenn auf das Halten der Positionen verzichtet oder via Initialisierung der Datensatz des Graphen übergeben wird. Dynamische Interaktion zu beobachten würde das Ergebnis beeinträchtigen, wenn nicht auf das Halten der Positionen bereits vorhandener Elemente verzichtet werden würde. Das zu sehende Ergebnis entspricht einem ohne dynamisches Zufügen von Knoten. Der Algorithmus wurde

auf einen bereits vorhandenen Graphen angewendet [5].

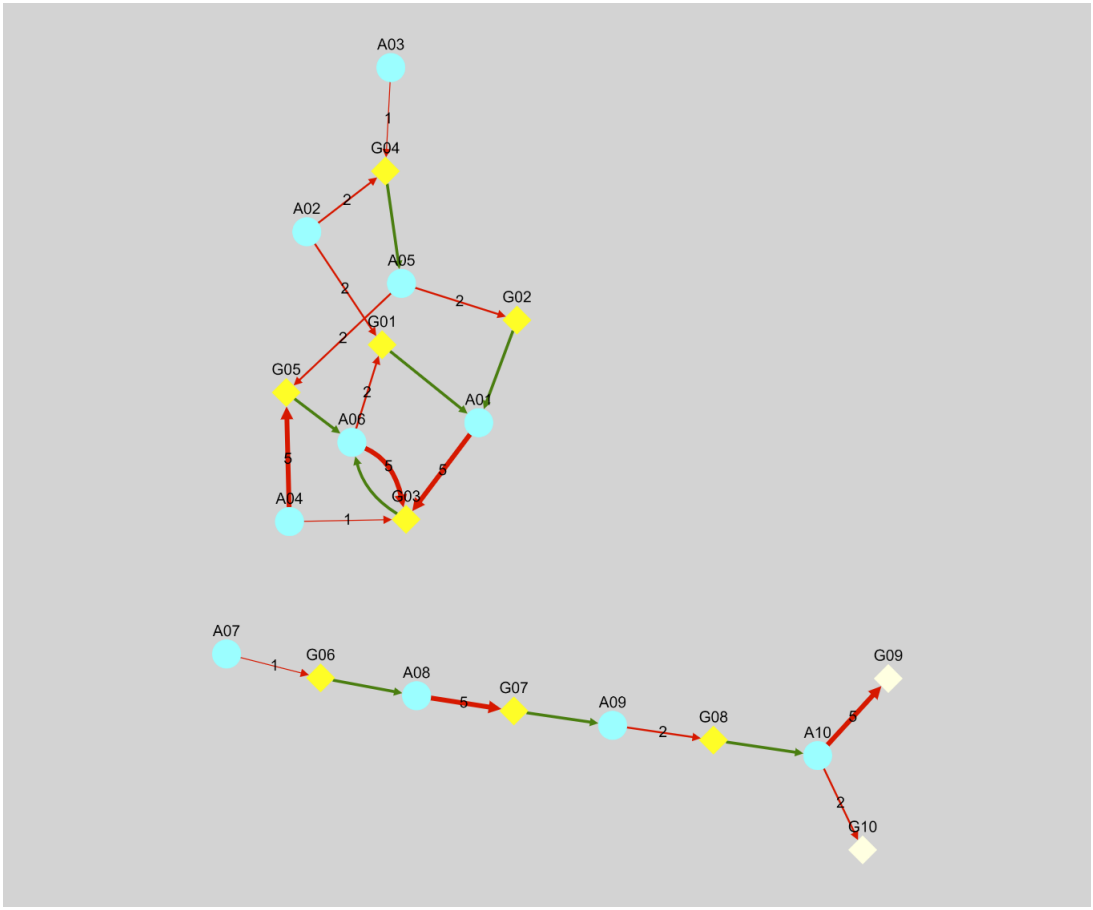

Abbildung 9: Layout cola Cytoscape.js

FCoSE

FCoSE (Fast Compound Spring Embedder) ist eine Erweiterung des klassischen *CoSE*-Layouts und wie das *cola*-Layout ebenfalls ein force-directed-Layout. Der erzeugte Graph ist dem vom *cola* sehr ähnlich (siehe Abbildung 10, linker Teilgraph). Hier gibt es ebenfalls eine Überschneidung der Kanten, es sind auch hier die Kommunikationsstrukturen zwischen den Agenten leicht ablesbar und Zusammenhänge sind erkennbar. Im Unterschied zum *cola*-Layout bietet *FCoSE* mit die Option `fixedNodeConstraint` die Möglichkeit, die Position der verschiedenen Knoten zu fixieren [12][1]. Mit dieser Option bleibt der Graph auch während eines kontinuierlichen Aufbaus und Veränderungen lesbar, und Überschneidungen werden minimal gehalten, während die Position vorhandener Knoten erhalten bleibt.

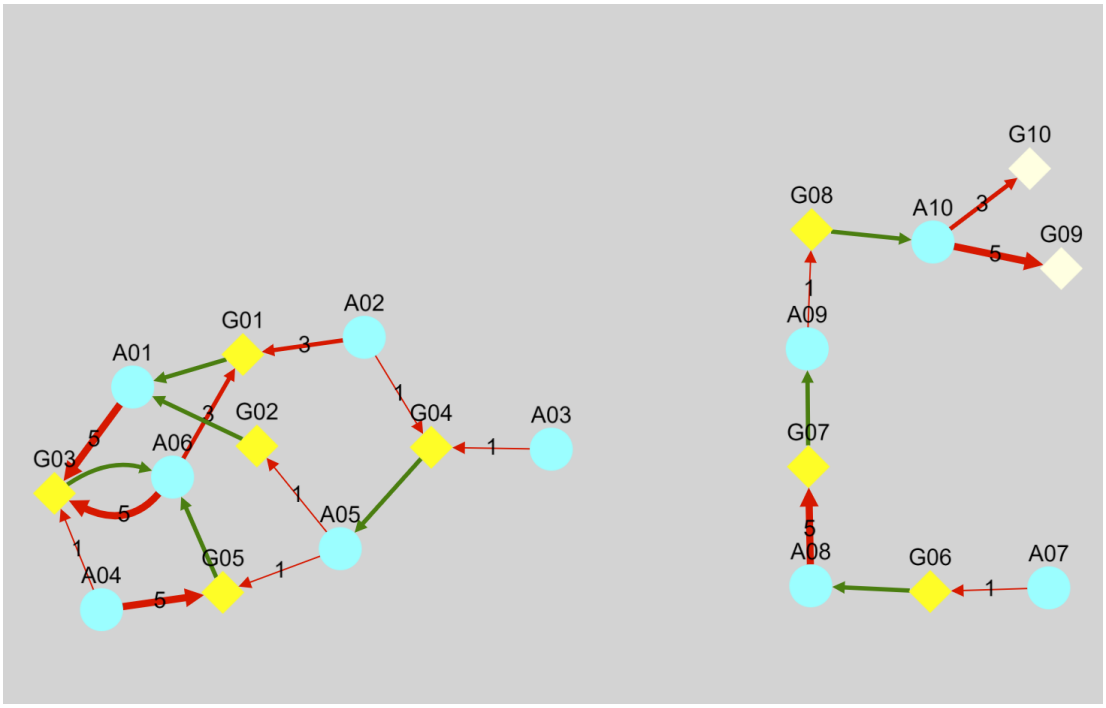

Abbildung 10: Layout FCoSE Cytoscape.js

Von allen untersuchten Algorithmen bringen die der force-directed-Layouts für diesen Anwendungsfall die besten Ergebnisse in Bezug auf die Überschneidungsfreiheit und die Darstellung von Kommunikationsstrukturen. Da die Implementation des *FCoSE*-Layouts durch die Option `fixedNodeConstraint` die Möglichkeit bietet, Positionen zu fixieren und bei kontinuierlicher Veränderung des Graphen trotzdem ein den Anforderungen entsprechendes Layout zu erhalten, erfüllt dieser Algorithmus die Anforderungen am besten und wird im Prototyp eingesetzt.

4.4 Entwurf

Im Folgenden werden die Komponenten des Prototyps beschrieben und ihr Zusammenspiel mit der Middleware erläutert.

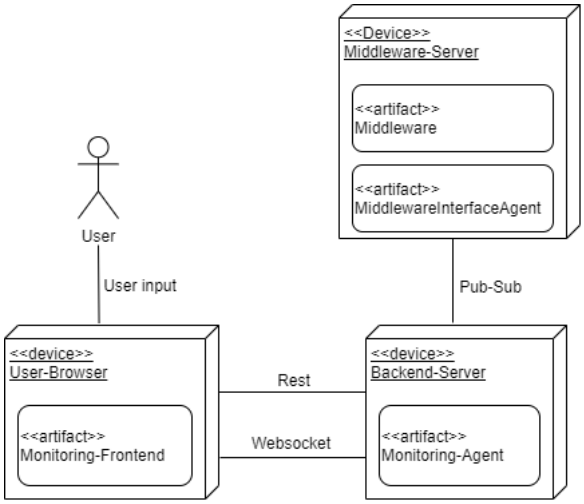

Abbildung 11: Verteilungsdiagramm

Wie in Abbildung 11 zu erkennen ist, besteht das System aus drei Komponenten. Die Middleware mit einem *middleware-interface-agent*, der als Quelle der zu visualisierenden Daten agiert; Der *monitoring*-Agent, der die Daten von der Middleware erfragt, aktuell hält und dem Frontend bereitstellt; Das Frontend, das die Daten vom Backend und dem Nutzer entgegennimmt, sie verarbeitet und visualisiert.

4.4.1 Middleware

Die Middleware hält die Daten zu den Kommunikationsstrukturen. Um von der Middleware Daten zu erhalten wird wie folgt vorgegangen. Für das Erlangen der Datensätze, auf Basis derer die Webapplikation arbeitet, ist die Registrierung an der Middleware Voraussetzung. Durch die Registrierung ist die Nutzung der Middleware-API möglich. Mit Hilfe der Publish-Subscribe-Kommunikation zwischen dem *monitoring*-Agenten des Visualisierungstools und dem *middleware-interface-agent* der Middleware werden Datenanfragen und Datensätze ausgetauscht. In der Gruppe *middleware-request* werden die Anfragen vom *monitoring*-Agenten publiziert, woraufhin in der Gruppe *middleware-reply* vom *middleware-interface-agent* eine Antwort publiziert wird. Beide Agenten haben jeweils die Gruppe, in der sie nicht publizieren, abonniert (siehe Abbildung 5).

Der von der Middleware zur Verfügung gestellte Publish-Subscribe-Dienst ermöglicht die Kommunikation zwischen den verschiedenen Agenten im System. Das Fehlen von Events erfordert ein zyklisches Abfragen der Datensätze durch den *monitoring*-Agenten

der Webapplikation. Würden Events vom *middleware-interface-agent* der Middleware in der Gruppe *middleware-reply* publiziert werden, könnten auf diese Weise die vom *monitoring*-Agenten initial abgefragten Datensätze kontinuierlich aktuell gehalten werden.

Nach dem Registrieren des *monitoring*-Agenten an der Middleware, dem Abonnement der Gruppe *middleware-reply* und dem Publizieren der Anfragen zu den Datensätzen in der Gruppe *middleware-request*, ist die Nutzung des Publish-Subscribe-Dienstes ohne Events also nur in eingeschränkter Form möglich.

4.4.2 Backend

Die vom Backend genutzte Schnittstelle zur Middleware wird als Bibliothek in Skala angeboten. Das Backend selbst ist in Java geschrieben. Auf diese Weise ist Kompatibilität der Sprachen gewährleistet, da beide in der JVM (Java Virtual Machine) laufen. Für die Schnittstelle von Backend und Frontend wird das Spring Framework genutzt.

Der *monitoring*-Agent registriert sich an der Middleware und meldet sich in den Gruppen an, in denen die für das Monitoring relevanten Informationen erfragt und publiziert werden. Für den initialen Erhalt dieser Daten werden alle Anfragen des Backends in der dafür vorgesehenen Gruppe veröffentlicht. Der *middleware-interface-agent*, der diese Gruppe abonniert hat, publiziert die Antworten in der Gruppe, welche vom *monitoring*-Agent abonniert wurde.

Der Publish-Subscribe-Dienst wird genutzt, um zu gewährleisten, dass die Daten aktuell bleiben. Hierfür werden zyklische Abfragen verwendet (siehe Kapitel 4.4.1). Es werden zyklisch Anfragen publiziert und die aufeinander folgenden Datensätze mittels Hashcode miteinander verglichen. Liegen Änderungen vor, wird der alte Datensatz verworfen und der neue Datensatz wird genutzt.

Es gibt Daten, deren Informationen in Relation zu einem vordefinierten Zeitfenster liegen. Dieses Zeitfenster liefert indirekt auch die Orientierung für sinnvolle Intervalle zur Abfrage.

Theoretisch ermöglicht die vorliegende Publish-Subscribe-Architektur eine Aktualisierung der Datensätze durch Auslösen von Events. Wenn ein Agent sich an der Middleware registriert, stirbt, eine Gruppe abonniert oder in einer Gruppe publiziert und diese Ereignisse via Events publik gemacht würden, wäre ein kontinuierliches Abfragen von

eventuellen Änderungen nicht notwendig. Die Datensätze würden mittels der empfangenen Events vom Backend angepasst werden.

Eine zyklische Abfrage bezüglich der Publikationen der einzelnen Agenten in eine Gruppe würde dennoch ausgeführt werden. Alle nicht-funktionierenden, in der API angedachten Events decken einmalige Ereignisse ab. Eine Information, die Auskunft bezüglich des Verhaltens eines Agenten über ein vordefiniertes Zeitfenster liefert, ist kein mit einem Event lieferbarer Wert. Es bestünde die Möglichkeit, bei jeder Publikation eines jeden Agenten in jeder Gruppe ein Event auszulösen. Das würde eine Klassifikation nach Sender (Agent) und Empfänger (Gruppe) unter Berücksichtigung der Zeit bedeuten und gegebenenfalls ein Überspringen von Events notwendig machen, wenn ein zu hohes Aufkommen von Events in einer Eventflut münden würden. Die Ergänzung um die zyklische Abfrage bezüglich der Publikationen eines bestimmten Agenten in einer bestimmten Gruppe innerhalb eines bestimmten Zeitintervalls wäre auch bei einer eventbasierten Nutzung der Publish-Subscribe-Architektur inklusive Events eine dem System entsprechende sinnvolle Ergänzung.

Da die API der Middleware zum Zeitpunkt des Entstehens dieser Bachelorarbeit keine Events unterstützt, werden alle möglichen Änderungen zyklisch geprüft. Änderungen werden mittels zyklischer Abfragen von der Middleware erfragt. Die auf Anfrage nach Publikationen gelieferte Liste beinhaltet alle lebendigen, publizierenden Agenten, die Gruppen in denen publiziert wurde, sowie die Anzahl der publizierten Nachrichten des entsprechenden Agenten in der entsprechenden Gruppe in den letzten 5 Sekunden.

Für die zyklische Abfrage wird ein Intervall von 5 Sekunden festgelegt. Dieses Intervall wird genutzt, um eine Aussage zu den von einem bestimmten Agenten in der bestimmten Gruppe innerhalb der letzten 5 Sekunden veröffentlichen Nachrichten treffen zu können.

Die von der Middleware erhaltenen Daten müssen im nächsten Schritt an das Frontend gereicht werden. Das Frontend erhält initial alle angefragten Daten. Im laufenden Prozess werden ausschließlich Änderungen am Datenbestand an das Frontend weitergereicht. Wie das Frontend mit den Datensätzen verfährt, folgt in Kapitel 4.4.3.

Damit das Frontend den kompletten Datensatz initial laden kann, werden dem Frontend vom Backend REST Endpoints für eine initiale Abfrage angeboten. Das initiale Laden ist notwendig, um das Konfigurationsmenü zu füllen. Bei den REST Endpoints schickt das Frontend eine Anfrage mittels HTTP ans Backend. Das Backend beantwortet diese Anfrage mit dem Datensatz im Body.

Bei den REST Endpoints müssen die Anfragen vom Frontend abgeschickt werden. Das Frontend hat keine Informationen darüber, ob sich der Datensatz geändert hat und aktualisiert werden muss. Der REST Endpoint bietet sich nur zum Initialisieren an, um nicht alle künftigen Informationen mittels Polling zu erfragen [14].

Es werden Websockets verwendet, um die erhaltenen Daten an das Frontend zu liefern wenn erkannt wird, dass sich die Daten geändert haben. Websockets sind mittels TCP realisiert und ermöglichen einen bidirektionalen Austausch von Daten. Aufgrund der Nutzung von Websockets ist ein Pollen der Daten vom Frontend nicht nötig.

Das Websocket-Protokoll besteht aus zwei Teilen: Dem Handshake und dem darauf folgenden Datentransfer. Beim Handshake verschickt der Client zuerst einen HTTP GET Request an den Server, woraufhin der Server mit dem 101 Switching Protocols antwortet, die Verbindung ein Upgrade erhält, und so zu einer dauerhaften Websocket-Verbindung (HTTP) wird. Dauerhaft bedeutet in diesem Fall, dass die Verbindung bis zur Beendigung aufrechterhalten wird. Regulär wird die Verbindung via closing Handshake beendet. Eine Unterbrechung der Verbindung aus anderen Gründen beendet diese ebenfalls. Der Request des Clients bezüglich Änderungen des aktuellen Datensatzes bleibt mit der Verbindung bestehen und hat ein Update der Daten seitens des Servers bei jeder Änderung zur Folge.

Ein initiales Laden der Daten ist ebenfalls mittels Websockets möglich, der zusätzliche Einsatz von REST Endpoints kapselt die Initialisierung vom laufenden Geschehen [16].

4.4.3 Frontend

Die vom Backend bereitgestellten Daten sollen im nächsten Schritt dem Nutzer zugänglich gemacht werden. Dazu wird eine Nutzeroberfläche (Frontend) benötigt.

Dem Nutzer soll ermöglicht werden, die Applikation ohne weiteren Installationsaufwand nutzen zu können, womit sich eine Webapplikation als Lösung anbietet. Für diese Art der Anwendungen existiert eine große Auswahl an Frameworks. Je nach Einsatzzweck können die Unterschiede gewichtig sein. Für die Auswahl des Frameworks war ausschlaggebend, dass die Komplexität für die Entwicklung der Anwendung ausreichend war [9].

Neben weiteren großen Frameworks wie Angular oder React, die ebenfalls hätten zum Einsatz kommen können, fiel die Wahl des Frameworks für den Prototyp auf Vue.js.

Das Frontend erfragt den initialen Datensatz via REST Request vom Backend und stellt den Datensatz dem Nutzer in den Dropdown-Menüs der Konfigurationskonsole bereit. Die dort präsentierten Daten werden, solange die *Autoreload*-Funktion nicht deaktiviert wurde, kontinuierlich aktualisiert.

Zur Erstellung und Pflege des Graphen wird Cytoscape.js genutzt (siehe Kapitel 4.3). Die folgenden Schritte werden benötigt, um den Datensatz vorzubereiten, damit der Datensatz den Anforderungen von Cytoscape entspricht. Die von Cytoscape.js benötigten Datensätze werden aus bis zu drei Eingabequellen erfasst und verarbeitet, um den aktuellen Graphen zu konstruieren. Analog funktioniert auch das Aktualisieren des Graphen. Der Erhalt und der Wandel der Graphentopologie ist wiederum durch die Auswahl des Layout-Algorithmus, unter Berücksichtigung der ermittelten Anforderungen, festgelegt (siehe Kapitel 4.3.2).

Die Abbildung 12 veranschaulicht den Erhalt, die Verarbeitung und die Aktualisierung der Daten im Frontend.

Beim „Warten auf Änderungen“ (oberer Teil der Abbildung 12) werden Änderungen am Datensatz, welche die aktuelle Graphentopologie betreffen, berücksichtigt. Dabei entstehende Aktualisierungen beziehen sich auf hinzukommende oder sterbende Agenten oder Änderungen an ihrem Kommunikationsverhalten, dem Abonnieren von Gruppen oder dem Publizieren in Gruppen. Die Graphentopologie kann ebenfalls verändert werden. Durch An- und Abwählen von Agenten und Gruppen kann der Nutzer Anpassungen im Konfigurationsmenü vornehmen. Diese Art der Konfiguration der Graphentopologie ist dem Nutzer auch innerhalb des bestehenden Graphen möglich. Durch das Anklicken eines Knotens kann dessen Zustand zwischen implizit und explizit wechseln. Auf diese Weise können selektive und explorative Subsysteme erstellt werden.

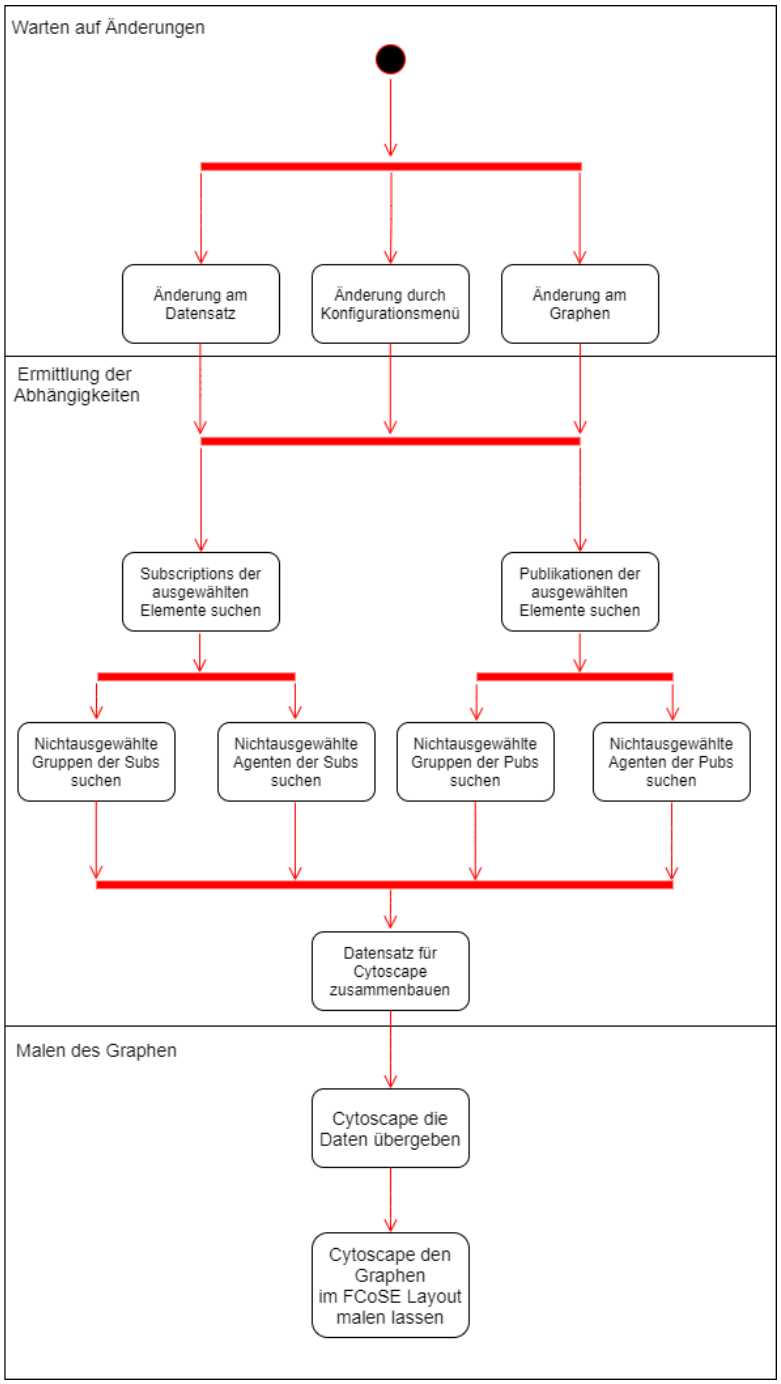

Abbildung 12: Aktivitätsdiagramm Aufbau und Update des Graphen

Die auf diese Weise zusammengesetzten Daten dienen als Grundlage zur „Ermittlung der Abhängigkeiten“ (mittlerer Teil der Abbildung 12). Die ausgewählten Elemente (Gruppen oder Agenten) werden hinsichtlich ihrer Abonnements und ihrer Publikationen untersucht. Zu jeder Relation zwischen einem Agenten und einer oder mehreren Gruppen oder aber einer Gruppe und einem oder mehreren Agenten wird, ausgehend vom ausgewählten Element, das Gegenstück ermittelt. Die Gegenstücke sind die impliziten Agenten und Gruppen. Anschießend werden implizite und explizite Elemente in einem für Cytoscape passendem Datensatz zusammengesetzt. Dieser Datensatz enthält auch die Kommunikation darstellenden Kanten (Nachrichten), wobei zwischen eingehenden und ausgehenden Nachrichten (*pubs* und *subs*) unterschieden wird. Die Definition der Kanten wird dabei durch Ziel und Quelle beschrieben. Die entsprechende Farbe und gegebenenfalls zugehöriger Index sind je nach Klasse der Kante zugehöriges Attribut. Der Datensatz ist eine Liste. Die Definitionen der Entitäten sind in dieser Liste wie folgt enthalten.

```
//Agent
{
  group: "nodes",
  data: {
    id: <id>,
    label: <name>,
    colors: <implitzit> ? "LightCyan" : "Cyan"
  }
}

//Gruppe
{
  group: "nodes",
  data: {
    id: <element>,
    label: <element>,
    colors: <implitzit> ? "LightYellow" : "Yellow",
    shape: "diamond"
  }
}
```

```
//Kante (Subscription)
{
  group: "edges",
  data: {
    id: <id>,
    source: <source>,
    target: <target>,
    msg_type: "sub"
  }
}

//Kante (Publication)
{
  group: "edges",
  data: {
    id: <id>,
    source: <source>,
    target: <target>,
    msg_type: "pub",
    traffic: <count>
  }
}
```

Der vorbereitete Datensatz muss an Cytoscape übergeben werden, um im letzten Schritt „Malen des Graphen“ den Graphen zeichnen zu können. Der aus dem Datensatz resultierende Graph wird von Cytoscape im gewählten Layout *FCoSE* (siehe Kapitel 4.3.2 - *FCoSE*) gezeichnet.

Die vom Backend vorgegeben Updatezyklen haben im Fall von Änderungen ein Update des Graphen und der Daten im Konfigurationsmenü zur Folge. Ein Unterbrechen des Zyklus ist durch das Deaktivieren der *Autoreload*-Funktion (siehe Abbildung 4) möglich. Ein manuelles Anfordern von Aktualisierungen der Daten ist bei deaktivierter *Autoreload*-Funktion durch die Nutzung des *Refresh*-Buttons gegeben.

Einen weiteren die Topologie beeinflussenden Faktor stellt der *Redraw*-Button dar. Dieser Button ermöglicht dem Nutzer eine Anpassung des geplotteten Graphen bezüglich

der Positionen. Der Layout-Algorithmus wird erneut auf den vorliegenden Datensatz angewendet, wobei die bisher gehaltenen Positionen der Knoten verworfen werden.

Dem Nutzer steht ein *Clean*-Button zur Verfügung, um zum Initialzustand zurückzukehren. Es werden alle Daten verworfen und das Konfigurationsmenü aktualisiert.

4.5 Fazit

Die entstandene Webapplikation visualisiert die dynamischen Kommunikationsstrukturen im vorgestellten Publish-Subscribe-System.

Die Webapplikation bietet eine Möglichkeit des Monitorings zur Überwachung und Überprüfung von Systemabläufen und kann eine Einarbeitung in ein Projekt ermöglichen oder beschleunigen.

Bei laufender Entwicklung ist durch die Überprüfung von Programm- und Systemverhalten ein weiterer Einsatzpunkt der Applikation gegeben. Der demonstrative Einsatz durch abstrahierte Bilder von Kommunikationsstrukturen, im Vergleich zum Auflisten von Logdateien ist zum Verstehen von Abläufen hilfreich. Mit Hilfe von Definitionen bezüglich Lesbarkeit und Symbolik kann der Interpretation des Menschen ausreichend Orientierung gegeben werden, um zu versuchen, einem jeden Betrachter die gleichen Informationen mitzuteilen, wenn komplexe Sachverhalte mit Kreisen und Linien nachvollziehbar gemacht werden.

Die Anwesenheit von Agenten, Gruppen und deren Kommunikation im System wird im Konfigurationsmenü und im bestehenden Graphen präsentiert und kontinuierlich aktualisiert.

Ohne den Aufbau oder die Handlungsweise der Agenten zu kennen, deren interne Abläufe verborgen bleiben, erschließt sich dem Anwender das Interaktionsverhalten der Agenten durch Darstellung der Kommunikation und ihrem dynamischen Wechsel.

Zur Erstellung des Graphen und auch zu seiner Modifizierung, über die systemgegebene Dynamik hinaus, sind Nutzereingaben erforderlich.

Die Systemstruktur beziehungsweise die beteiligten Akteure der zu untersuchenden Situation sind dem Nutzer unbekannt. Die Webapplikation gibt dem Benutzer die Möglichkeit mittels Exploration der Umgebung die Kommunikationsstrukturen durch An- und Abwählen von Agenten und Gruppen kennenzulernen.

In dem Konfigurationsmenü (siehe Kapitel 4.2) der Webapplikation ist eine Auswahl mehrerer Entitäten möglich. Ist ein Objekt von Interesse und nicht in der Liste enthalten, ist dieses Objekt nicht an der Middleware registriert. In diesem Fall ist das Verifizieren von fehlerhaftem oder nicht erwartungskonformen Verhalten schon vor dem Erstellen des Graphen möglich.

Eine Ausnahme bilden nicht existente Gruppen (siehe Kapitel 4.1.3). Nicht existente Gruppe sind Gruppen, in denen kein Abonnement besteht, in die aber publiziert wird. Diese Art von Gruppen werden nicht im Dropdown-Menü angezeigt. Nicht existente Gruppen werden allerdings gezeichnet, wenn ein in dieser Gruppe publizierender Agent betrachtet wird. Durch die Darstellung nicht existenter Gruppen kann festgestellt werden, welche Publikationen nicht empfangen werden.

Zum Erhalt von Übersichtlichkeit und einem anforderungsspezifischen Layout wurde der Fall eines wiederauftauchenden Knotens, mit selbem Kontext wie vor seinem Verschwinden, an einer dem Algorithmus entsprechenden Position implementiert (siehe Kapitel 3.3.3 - Positionserhalt).

Es wurden optisch klar unterscheidbare, dennoch klassifizierbare Formen und Farben gewählt, um die Effizienz von Visualisierung als gewähltes Mittel zur Vermittlung hoher Informationsdichte nutzen zu können (siehe Kapitel 4.1.1).

Das Konfigurationsmenü der Webapplikation besitzt eine Legende, um Abweichungen bezüglich der Interpretation vorzubeugen.

Der Effekt der Informationsvermittlung durch Visualisierung der Kommunikationsstrukturen würde beeinträchtigt werden, wenn die theoretisch les- und interpretierbaren Graphenelemente in ihrer Anordnung visuelles Rauschen erzeugen. Diesem Problem wurde mit dem Einsatz eines den Anforderungen entsprechenden Layout-Algorithmus entgegengewirkt (siehe Kapitel 4.3).

Der Graph erhält zyklische Updates. Unterliegt der zu untersuchende Teilgraph hoher Fluktuation, beispielsweise bei einer Snapshot-Betrachtung, ist eine manuelle Anpassung der Updatezyklen möglich (siehe Kapitel 4.4). Durch ein Deaktivieren der *Autoreload*-Funktion werden die Updates ausgesetzt. Manuelle Updates können in diesem Zustand durch Nutzung des *Refresh*-Buttons angefordert werden.

Ein persönliches Empfinden mangelnder Übersichtlichkeit kann für den Anwender vor den Grenzen des Algorithmus erreicht werden (siehe Kapitel 4.3). Eine manuelle Ausfüh-

rung des Algorithmus, ohne den erneuten Erhalt der bisherigen Positionen, ist durch die Betätigung des *Redraw*-Buttons möglich.

Der dargestellte Graph zeigt eine Übersicht der Interaktionspartner, der von dem Nutzer explizit gewählten Entitäten. Zur bedarfsorientierten Anpassung des Fokus der Betrachtung, ist durch das Klicken auf vorhandene Entitäten eine Erweiterung oder Einschränkung der Darstellung möglich.

Mit diesem Fazit zum Design und der Realisierung konnte die Erfüllung der Anforderungen aus dem Kapitel „Analyse“ erfolgreich verifiziert werden.

5 Schluss

Dieses Kapitel fasst die Arbeit bis zu diesem Punkt noch einmal kurz zusammen und beschreibt optionale Weiterentwicklungsmöglichkeiten.

5.1 Zusammenfassung

In dieser Arbeit wurde ein Prototyp zur explorativen Visualisierung dynamischer Kommunikationsstrukturen in einem Publish-Subscribe-System entwickelt.

Zu Beginn wurde die Motivation zu dieser Arbeit und zu der Entwicklung des Prototyps erörtert. Anschließend wurden die durch die Entwicklung angestrebten Ziele geklärt. Die Einleitung schließt mit der Gliederung der Ausarbeitung (siehe Kapitel 1).

Diese Arbeit wird mit einem Grundlagenkapitel fortgeführt, um grundlegende und für diese Arbeit relevante Begriffe und Sachverhalte zu klären und zu definieren (siehe Kapitel 2). Nach einer kurzen Erläuterung verteilter Systeme wurden verschiedene Kommunikationsstrukturen erklärt. Im Folgenden wurden die Begriffe „Agent“ und „Multiagentensystem“ spezifiziert und die Systemarchitektur einer Middleware vorgestellt.

Auf dieser Grundlage wurde im Kapitel „Analyse“ (siehe Kapitel 3) das Publish-Subscribe-System, dessen dynamischen Kommunikationsstrukturen visualisiert werden und die beteiligten Entitäten vorgestellt. Es wurden durch das System gegebene Herausforderung und Anforderungen formuliert. Eine zu diesen Formulierungen passende Anforderungsanalyse wurde bezüglich Entitäten, Funktionen und Eigenschaften erörtert. Im Analysekapitel wurden des Weiteren zwei Laborszenarien dargelegt, um ein genaueres Bild des Einsatzes der Webapplikation und der Nutzergruppen zu erhalten. Die Schwerpunkte der Anforderung und die damit verbundenen Herausforderungen des Monitorings in Publish-Subscribe-Systemen wurden vor dem Fazit der Analyse beschrieben.

Design und Realisierung wurden im gleichnamigen Kapitel (siehe Kapitel 4) behandelt. Entwurfsentscheidungen bezüglich der Auswahl der Algorithmen, Graphendarstellung und -konstruktion wurden unter Berücksichtigung der aus der Analyse hervorgegangenen Anforderungen diskutiert. Anschließend wurde die Art der Umsetzung beschrieben. Das Kapitel schließt mit einem Fazit, welches die Erfüllung der Anforderungen aus dem Kapitel „Analyse“ evaluiert.

5.2 Ausblick

An dieser Stelle der Arbeit werden einige Ideen für optionale Erweiterungen des entwickelten Prototyps erläutert.

Eine Erprobung des Prototyps mit realen Nutzern unter realen Bedingungen war aufgrund der Corona-Pandemie bisher nicht möglich. Eine Nutzerevaluation des Prototyps wäre von Interesse gewesen.

Ein Einsatz in den Laboren hätte aufzeigen können, wie die visuellen Eindrücke der Graphendarstellung auf den Nutzer wirken. Die Förderung des Verstehens der Infrastruktur durch den Einsatz des Tools hätte erprobt werden können. Des Weiteren hätte ermittelt werden können, ob die Informationsdichte ausreichend zum Verstehen der Strukturen gewesen wäre. Mit Nutzererfahrungen dieser Art hätte das Tool weiter an die Anforderungen in den Laboren angepasst oder der Bedarf an optionalen Erweiterungen bestätigt werden können. Ein Austausch mit der Nutzergruppe hätte zu neuen Denkanstößen führen können. Problembetrachtung aus anderen Perspektiven und Ermittlung individueller Bedürfnisse der Nutzer klären häufig den Bedarf für mögliche Erweiterungen oder Optimierungen.

Eine Erweiterungsmöglichkeit wäre es, durch Klicken mit der rechten Maustaste auf einen Knoten oder eine Gruppe, diese zu verstecken. Mit Hilfe einer Nutzeranalyse könnte geklärt werden, ob diese Möglichkeit den Graphen optisch zu bereinigen beim Verstehen der Systemstrukturen hilft. Eine weitere Idee wäre es, die Kanten abhängig ihres Nachrichtenaufkommens stärker farblich zu unterscheiden, um die unterschiedlich stark gezeichneten Kanten besser abgrenzen zu können.

Die hier entwickelte Webapplikation könnte ebenfalls in anderen Publish-Subscribe-Systemen eingesetzt werden, solange die entsprechenden Schnittstellen vorhanden sind (beziehungsweise angepasst werden können). Eine Übertragbarkeit ist durch die Architektur gegeben. Die Grenzen der Übertragbarkeit zeichnen sich durch den Einsatzzweck des jeweiligen Systems beziehungsweise der Größe des Systems oder der im System vorherrschenden Informationsdichte der Kommunikationsstrukturen ab. Nimmt die Anzahl von Knoten und Kanten massiv zu, stößt die explorative Visualisierung an ihre Grenzen, da visuelles Rauschen auftritt. Die Übertragbarkeit bezieht sich auf die Visualisierung überschaubarer, auf einen Blick darstellbarer Netze, deren Inspektion von Interesse ist.

Größer gearteten Netzen benötigen hierarchische Strukturen und Gruppierung, um lesbare Darstellungen zu visualisieren. Der Prototyp hingegen hat eine flache Struktur und ist nicht hierarchisch.

Das Zoomen in Teilgraphen wurde im Kapitel „Analyse“ als eine optionale Erweiterung der Funktionen aufgeführt und in dieser Arbeit nicht realisiert (siehe Kapitel 3.3.2). Da das Hineinzoomen in einen Teilgraphen nach einer hierarchischen Struktur verlangt, entspricht die Struktur des Prototyps nicht der notwendigen Tiefe. Diese Funktionalität könnte eine Möglichkeit sein, oben beschriebene Systeme, die die Grenzen des Prototyps sprengen, optisch zu gruppieren.

Eine andere Art von Zoom, die technisch möglich wäre, wäre das Zoomen in Nachrichten. Als optionales Feature der Funktionen wurde das Lauschen bereits im Vorhinein als mögliche Erweiterung im Kapitel „Analyse“ beschrieben und nicht implementiert (siehe Kapitel 3.3.2). Tatsächlich wäre der Ressourcenaufwand bei starker Kommunikation oder einer hohen Anzahl von Gruppen nicht zu unterschätzen, da der *monitoring*-Agent Abonnent jeder Gruppe sein müsste und das Halten jeder einzelnen Nachricht notwendig wäre.

Die Konfiguration des Grades der impliziten Darstellung wäre ebenfalls eine mögliche Erweiterung. Der Grad der Darstellung der implizit angezeigten Knoten ist eins. Das bedeutet, dass jeder mit einem explizit ausgewählten Knoten durch eine Kante verbundene Konten mit angezeigt wird. Eine Erweiterung durch das Wechseln des Wertes eines Knotens von implizit auf explizit erweitert zwar indirekt den Grad der Darstellung, dies allerdings auch nur punktuell. Ein höherer Grad implizit angezeigter Agenten und Gruppen könnte über einen Slider eingestellt werden (0 bis x). Der Nutzer könnte selbst bestimmen, wie viel von der Umgebung des erstellten Teilgraphen er sehen möchte. Cytoscape.js stellt zwar Metriken zur Auswahl von Teilgraphen zur Verfügung und eine theoretisch gut passende Metrik wäre die Topologie in Kombination mit einer Punktzahl. Es wäre dabei möglich, basierend auf der Topologie des Graphen relativ zu einem ausgewählten Ort (zum Beispiel N Schritte von einem ausgewählten Knoten entfernt) einen Teilgraphen auszuwählen. Dabei soll allerdings die topologische Menge des Graphen reduziert werden. Der bestehende Graph bildet also kein selektives Subsystem ab, das nach Modifizierung von N erweitert wird, sondern die Größe des Systems wird reduziert [6]. Eine Erweiterung durch Rekursion ist umsetzbar und könnte bei Bedarf realisiert werden.

Die Webapplikation um eine Tracing-Funktion zu erweitern könnte eine hilfreiche Ergänzung sein, Situationen nicht nur zu erkunden, zu verfolgen und zu analysieren, sondern

auch zu rekonstruieren könnte den Einsatzbereich des Tools erweitern. Abgesehen von beispielsweise einem Tracing-Slider im Konfigurationsmenü, zur Übergabe der gewünschten Startzeit der Aufzeichnung, bedarf eine solche Implementierung einer Datenbank, da die Historie der Graphendaten zwecks Rekonstruktionen gehalten werden muss. Die Dauer der Datenspeicherung steht dabei in Relation zu dem Zeitraum über den die Aufnahme abrufbar sein soll.

Literatur

[1] Balci, Hasan ; Dogrusoz, Ugur: fCoSE: a fast compound graph layout algorithm with constraint support. In: *IEEE Transactions on Visualization and Computer Graphics* (2021)

[2] Bullinger, Hans-Jörg ; Hompel, Michael: Multiagentensysteme im Internet der Dinge – Konzepte und Realisierung. In: *Internet der Dinge - www.internet-der-dinge.de*. Berlin Heidelberg New York : Springer-Verlag, 2007, S. 281 – 281. – ISBN 978-3-540-36733-8

[3] CONFERENCE, ACM/IFIP INTERNATIONAL M.: *ACM/IFIP Middleware 2020*. – URL https://2020.middleware-conference.org/. – Zugriffsdatum: 2021-11-23

[4] Cytoscape: *Cytoscape - Breadthfirst*. – URL https://js.cytoscape.org/#layouts/breadthfirst. – Zugriffsdatum: 2021-07-29

[5] Cytoscape: *Cytoscape - Cola*. – URL https://github.com/cytoscape/cytoscape.js-cola. – Zugriffsdatum: 2021-07-29

[6] Cytoscape: *Cytoscape - Using Layouts*. – URL https://blog.js.cytoscape.org/2020/05/11/layouts/. – Zugriffsdatum: 2021-07-29

[7] Eichler, Tobias: *Agentenbasierte Middleware zur Entwicklerunterstützung in einem Smart-Home-Labor*, HAW Hamburg - Hochschule für Angewandte Wissenschaften, Diplomarbeit, 10 2014

[8] Eichler, Tobias ; Draheim, Susanne ; Grecos, Christos ; Wang, Qi ; Luck, Kai von: Scalable context-aware development infrastructure for interactive systems in smart environments. In: *13th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2017, Rome, Italy, October 9-11, 2017*, IEEE Computer Society, 2017, S. 147–150. – URL https://doi.org/10.1109/WiMOB.2017.8115848

[9] Fabrik GmbH, SnipClip die digitale: *SPA Frameworks - ein Vergleich zwischen Angular, React und Vue*. Januar 2021. – URL https://www.snipclip.com/blog/spa-frameworks-ein-vergleich-zwischen-angular-react-und-vue/

[10] Hamburg, HAW: *Creative Space for Technical Innovations*. – URL https://csti.haw-hamburg.de/. – Zugriffsdatum: 2021-10-08

[11] Hamburg, HAW: *Living Place.* – URL https://livingplace.haw-hamburg.de/. – Zugriffsdatum: 2021-10-08

[12] Hasan Balci, Ugur D.: *Cytoscape - fcose.* – URL https://github.com/iVis-at-Bilkent/cytoscape.js-fcose. – Zugriffsdatum: 2021-07-29

[13] Horváth, I. (Hrsg.) ; Keenaghan, G. (Hrsg.): *INTEGRATION OF COMPLEX EVENT PROCESSING INTO MULTI-AGENT SYSTEMS: TWO USE CASES FOR DISTRIBUTED SOFTWARE DEVELOPMENT SUPPORT.* Organizing Committee of TMCE, 2020. – URL https://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/papers/TMCE2020.pdf. – Zugriffsdatum: 2020-10-31. – ISBN 978-94-6384-131-3

[14] Keranen, Ari ; Kovatsch, Matthias ; Hartke, Klaus: *RESTful Design for Internet of Things Systems.* September 2017. – URL https://tools.ietf.org/id/draft-keranen-t2trg-rest-iot-05.html

[15] Klügl, Franziska: *Aktivitätsbasierte Verhaltensmodellierung und ihre Unterstützung bei Multiagentensimulationen*, Universität Würzburg, doctoralthesis, 2000

[16] Melnikov, Alexey ; Fette, Ian: *The WebSocket Protocol.* RFC 6455. Dezember 2011. – URL https://rfc-editor.org/rfc/rfc6455.txt

[17] Oracle: *GlassFish Message Queue 4.4.2 Technical Overview.* Jul 2011. – URL https://docs.oracle.com/cd/E18930_01/html/821-2443/idx-11.html

[18] Richter-Peill, Jarig: *Visualisierung der Interaktion in Multiagentensystemen.* Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, Universität Hamburg, Fachbereich Informatik, Baccelaureatsarbeit, Oct. 2005. – URL https://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/416/richter-peill_bacc.pdf

[19] Tanenbaum, Andrew S.: *Distributed Systems.* Third edition 3. Version: 01 (February 2017). Maarten van Steen, Februar 2017. – ISBN 978-90-815406-2-9

[20] Tarkoma, Sasu: *Publish / Subscribe Systems: Design and Principles.* 1st. Wiley Publishing, 2012. – ISBN 1119951542

[21] Tonn, Jakob: *ASGARD - Ein grafisches Monitoring- und Management-Tool für JIAC*, Technische Universität Berlin, Diplom Thesis, 2011

[22] Ware, Colin: *Information Visualization: Perception for Design*. 3. Amsterdam : Morgan Kaufmann, 2012 (Morgan Kaufmann Series in Interactive Technologies). – URL http://www.sciencedirect.com/science/book/9780123814647. – ISBN 978-0-12-381464-7

[23] Wismüller, Roland: *Verteilte Systeme*. 2016. – URL https://www.bs.informatik.uni-siegen.de/web/wismueller/vl/ss16/rn2/v_2.pdf. – Zugriffsdatum: 2019-11-10

[24] Wooldridge, Michael ; Jennings, Nicholas R.: Intelligent agents: theory and practice. In: *The Knowledge Engineering Review* 10 (1995), Nr. 2, S. 115–152

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Explorative Visualisierung dynamischer Kommunikationsstrukturen in einem Publish-Subscribe-System

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original