Context-Awareness in Multiagentensimulationen auf Basis von Complex Event Processing

Daniel Steiman

Fakultät Technik und Informatik
Hochschule für Angewandte Wissenschaften Hamburg
Berliner Tor 7
20099 Hamburg, Deutschland
daniel.steiman@haw-hamburg.de

Abstract: Kontextbewusste Systeme bieten die Möglichkeit, dass Anwendungen ihr Verhalten ohne die aktive Einflussnahme von Anwendern selbstständig an vorherrschende Situationen anpassen. Dazu erfassen sie Kontextdaten, die verarbeitet und zur Steuerung der Anwendung verwendet werden. Aus diesem Grund wurden bereits Toolkits, Frameworks und Ansätze entwickelt, die eine einfache Entwicklung von kontextbewussten Anwendungen ermöglichen sollen. Die bisher auf dem Gebiet der *Context-Awareness* vorgestellten Arbeiten weisen jedoch Nachteile auf, die einen Einsatz in bestimmten Anwendungsbereichen stark einschränken oder sogar völlig ausschließen. Dazu zählen vor allem Anwendungsbereiche, die den Anspruch haben riesige Mengen von Ereignissen und damit riesige Volumen an Kontextdaten in Echtzeit verarbeiten zu können. Dieses Paper zeigt die Nachteile bestehender Ansätze auf und befasst sich mit der Entwicklung eines Konzepts, mit dem ein breites Spektrum von Anwendungen möglichst einfach kontextbewusst gemacht werden kann. Um die Einsatzfähigkeit und die Performance des entwickelten Konzepts zu untersuchen, wird es in einem prototypischen Fallbeispiel einer Multiagentensimulation verwendet.

1 Einführung

Der Begriff der *Context-Awareness*, also des Kontextbewusstseins im informationstechnischen Sinne, wurde maßgeblich von Gregory D. Abowd und Anind K. Dey eingeführt [ADB+99]. Abowd und Dey definieren außerdem den Begriff Kontext als Bezeichnung jeglicher Information die verwendet werden kann, um die Situation einer Person, eines Ortes oder eines Objekts zu beschreiben. Eine Anwendung die kontextbewusst ist, erfasst also diese Informationen aus ihrer Umgebung und wertet sie aus, um sich automatisch den aktuellen Gegebenheiten und Ereignissen anzupassen.

Jeder Vorgang in der realen Welt, als auch jede Zustandsänderung in einem System, kann laut David Luckham als ein Ereignis (*Event*) angesehen werden [Luc01]. Mögliche Ereignisse wären zum Beispiel: Ein Flugzeug landet am Frankfurter Flughafen, ein Wettersensor meldet eine Temperaturveränderung, es werden mit einer Kreditkarte 500 Euro von einem Konto abgebucht, ein Musikalbum wird online heruntergeladen. All diese Ereig-

nisse können einen Einfluss auf den Ablauf eines Prozesses nehmen und sich somit auf dessen weiteren Gesamtverlauf auswirken.

Mittlerweile besteht immer mehr der Bedarf, komplexe Abläufe in der realen Welt mit Hilfe von Multiagentensystemen zu simulieren. Dabei können je nach Simulation mehrere hunderttausend Agenten zum Einsatz kommen. Jede Veränderung des Simulationszustands bedeutet eine Kontextveränderung und somit eine riesige Menge von Ereignissen, die kontinuierlich und möglichst performant verarbeitet werden müssen.

Es besteht also nicht mehr das Problem, dass zu wenige Informationen vorhanden sind, sondern die Herausforderung besteht inzwischen darin, aus der Informationsflut die wichtigen Erkenntnisse herauszuziehen und darauf entsprechend zu reagieren. Bei bisherigen Ansätzen ist es so, dass alle Informationen, welche beispielsweise durch das Auftreten von Events entstehen, zunächst gespeichert werden. Danach können entweder manuelle Abfragen von Kontextänderungen gemacht oder automatische Benachrichtigungen über entsprechende Kontextänderungen veranlasst werden. Diese Vorgehensweise ist jedoch für eine Auswertung, der in einer Simulation kontinuierlich auftretenden Ereignisse, zu unperformant. Zudem sind die analysierbaren Daten durch den Zwischenschritt des Abspeicherns und Aufbereitens der Informationen im Normalfall bereits veraltet.

Um jedoch zeitnah auf die gegebenenfalls riesigen Mengen unterschiedlichster Ereignisse einer Anwendung reagieren zu können, wurde die recht junge Technologie des *Complex Event Processing* (CEP) entwickelt [Luc01]. In diesem Paper wird deshalb anhand eines Fallbeispiels untersucht, welche Vorteile durch den Einsatz von Complex Event Processing in einer Context-Service Komponente entstehen können. Die entwickelte Lösung soll sich dabei möglichst leicht in bestehende Umgebungen integrieren lassen und in einem breiten Spektrum von Anwendungsszenarien einsetzbar sein. Es wird außerdem gezeigt, inwieweit das in diesem Paper vorgestellte Konzept bereits bestehende Ansätze auf dem Gebiet der Context-Awareness verbessert.

Der Hauptteil dieses Papers ist in die folgenden Abschnitte unterteilt: Abschnitt 2 betrachtet verwandte Arbeiten, um den Rahmen und die Problemstellung dieses Papers genauer zu eläutern. Abschnitt 3 behandelt das Testszenario der Multiagentensimulation (3.1), den Aufbau der Testumgebung anhand seiner Komponenten (3.2) und das Design der Kontextkomponente (3.3) mit dem die Probleme verwandter Arbeiten gelöst werden. In Abschnitt 4 folgt dann die Zusammenfassung der Ergebnisse und ein Fazit.

2 Verwandte Arbeiten

In den letzten Jahren sind einige Forschungen auf dem Gebiet der Context-Awareness durchgeführt worden. Grundlegende Arbeiten, wie die Definition von kontextbewusstem Computing [ADB+99], sowie die Entwicklung einer ersten Infrastruktur für die prototypische Erstellung kontextbewusster Anwendungen [DAS01], stammen von Anind K. Dey und Gregory D. Abowd. Mittlerweile gibt es neben dem *Context Toolkit* von Dey und Abowd einige weitere Arbeiten, die sich mit Konzepten zur Entwicklung kontextbewusster Anwendungen beschäftigen. Zu den bekanntesten gehören unter anderem *Hydrogen*

[HSP+03], *SOCAM* [GPZ04], *Gaia* [RC03] und *JCAF* [Bar05]. Eine Übersicht über bestehende Arbeiten auf dem Gebiet bieten [BDR07] und [KX12]. Gaia und JCAF gehören dabei bereits zu den Ansätzen, die positive Eigenschaften und Funktionalitäten vorangegangener Arbeiten aufgreifen und diese zu einem eigenen verbesserten Ansatz weiterentwickeln. Aus diesem Grund werden primär diese beiden Arbeiten als Referenz für einen eigenes Context-Service Konzept herangezogen. Die Konzepte von Gaia als auch von JCAF verfügen jedoch trotzdem noch über einige Nachteile und ungelöste Probleme die im weiteren Verlauf betrachtet werden.

Die Kontextmodelle [BBH+10, SLP04] von Gaia und JCAF verwenden jeweils eine Tripelstruktur, um Kontextinformationen zu speichern und zu verwalten. Um verschiedene Kontextinformationen ablegen zu können, werden in JCAF APIs in Form der Interfaces *Entity*, *Relationship* und *ContextItem* vorgegeben. Entwickler sind bei der Modellierung von Kontextinformationen an die jeweiligen Tripelstrukturen gebunden und im Falle von JCAF dazu gezwungen, ihre Anwendung gegen die entsprechenden Interfaces zu implementieren. Andernfalls können Gaia und JCAF die Kontextinformationen weder verwalten noch verarbeiten. Dadurch entsteht das Problem, dass sich diese Ansätze nicht dafür eignen bereits bestehende Anwendungen context-aware zu machen, da sie sich nicht nachträglich integrieren lassen.

Die Anwendung muss also von Anfang an in den Frameworks entwickelt werden. Sowohl bei Gaia, als auch bei JCAF ist die Anwendung tief mit den Frameworks und ihrer Infrastruktur verwoben. Dies verursacht eine starke Kopplung und eine niedrige Modularität der Anwendung. Außerdem kann die Anwendung dadurch nicht mehr in einer beliebigen Sprache implementiert werden, sondern muss die Sprache des jeweiligen Frameworks verwenden.

JCAF verwendet Java RMI zur Verteilung und Kommunikation zwischen der Anwendung (Client) und dem Kontext-Service (Server). Dies hat zur Folge, dass jede Instanz die zur Laufzeit in der Anwendung verwendet wird als Java RMI Stub auf der Serverseite generiert werden muss. Dadurch wird ein entsprechend hoher Ressourcenverbrauch beim Arbeitsspeicher verursacht, welcher bei der Verwaltung von gegebenenfalls mehreren hunderttausend Agenteninstanzen nicht mehr vertretbar ist.

Context Reasoning beschreibt den Vorgang der Deduktion, also der Schlussfolgerung, von für die Anwendung relevanten Informationen aus verschiedenen Kontextinformationen. Dabei werden sogenannte *low-level* Informationen, z. B. durch Aggregation, auf eine höhere Hierarchiebene (*high-level*) abgebildet. Beispielsweise können GPS-Signale als Längen- und Breitengrad vorliegen und dann durch Reasoning auf Orte wie ZUHAUSE oder ARBEITSPLATZ abgebildet werden (siehe Abbildung 1). Ein weiteres Problem besteht darin, dass JCAF über keinerlei Möglichkeiten des Reasonings von Kontextinformationen verfügt. Gaia unterstützt Context Reasoning, welches jedoch auf aussagenlogische Ausdrücke mit den Operatoren UND, ODER und NICHT begrenzt ist. Dadurch ist das Spektrum der möglichen Operationen sehr eingeschränkt. Die Unterstützung temporallogischer Ausdrücke wird deshalb bei Gaia explizit als ein offener Punkt für zukünftige Arbeiten genannt.

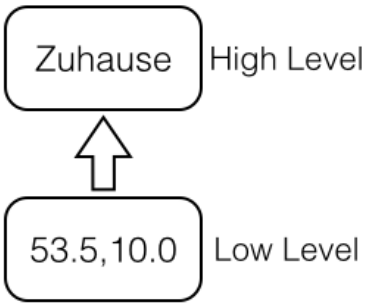

Abbildung 1: Abbildung von Koordinaten (low-level) auf Orte (high-level) durch Reasoning

Neben der Möglichkeit des Context Reasoning fehlt JCAF auch eine umfangreiche *Context Query Language* (CQL) [RWK+08] zur spezifischen Abfrage von Kontext. JCAF bietet APIs an, mit denen sich der gesamte Kontext von Objekten abfragen lässt. Jedoch lassen sich damit keine Auswahlkriterien zur Eingrenzung der Ergebnismenge, wie zum Beispiel bei einer SQL-Abfrage von Datensätzen, formulieren. Gaia dagegen unterstützt etwas spezifischere Abfragen von Kontextinformationen, welche die in [RWK+08] genannten Anforderungen an eine CQL jedoch nur zu einem Bruchteil erfüllen.

Die Performance von JCAF ist, hinsichtlich der Anforderung riesige Mengen kontinuierlich auftretender Kontextereignisse in Echtzeit verarbeiten zu können, ebenfalls nicht ausreichend. Bei einem Testlauf wurde nur ein geringer Durchsatz bei der Verarbeitung von Kontextänderungen erreicht. JCAF scheint also zumindest für möglichst echtzeitfähige Simulationen nicht in Betracht gezogen werden zu können, sobald die zu verarbeitenden Datenmengen groß werden. Da Gaia nicht öffentlich für Testzwecke zur Verfügung steht, können an dieser Stelle keine Aussagen zum Ressourcenverbrauch sowie zur Performance gemacht werden.

In den Ansätzen von Gaia und JCFA wurden also Nachteile und Probleme in den folgenden Teilbereichen identifiziert:

- Systemarchitektur
- Kontextmodellierung
- Context Reasoning
- Abfrage von Kontext
- Performance

In diesem Paper wird deshalb ein Konzept zur einfachen Realisierung von kontextbewussten Anwendungen vorgestellt, welches die zuvor identifizierten Nachteile und Probleme der bisherigen Arbeiten lösen soll.

3 Testumgebung und Beispielszenario

Das in diesem Paper vorgestellte Konzept eines Context-Service, soll bezüglich seiner Verbesserungen in den zuvor genannten Kriterien evaluiert werden. Dazu wird ein Beispielszenario für eine Multiagentensimulation [MN11] entwickelt und eine Testumgebung aufgebaut, in der die Multiagentensimulation gemeinsam mit der Context-Service Komponente als Fallbeispiel zum Einsatz kommt.

3.1 Beispielszenario

Das Beispielszenario basiert auf einem ökologischen Modell, in dem das Verhalten von Geparden und Gazellen in freier Natur simuliert wird [Eck13]. Die Geparden werden in ihrem Jagdverhalten simuliert und verfolgen dabei Gazellen. Die Gazellen fliehen wiederum vor den Geparden, solange sie genügend Energie haben. Eine Gazelle kann dem Geparden dann entweder entkommen oder sie wird von dem Geparden erlegt. Während der gesamten Simulationszeit verändern sich nun in jeder Simulationsiteration die Zustände der Gazellen und Geparden, wie zum Beispiel ihre Position, ihre Energie usw. und damit ihr derzeitiger Kontext.

Die Agenten sind grundsätzlich kontextsensitiv. Diese Kontextsensitivität ist jedoch auf das Bewusstsein über ihren eigenen Zustand und ihre nähere Umgebung zu einem bestimmten singulären Zeitpunkt beschränkt. Mit Hilfe des Context-Service kann diese Beschränkung überwunden werden. Er ermöglicht eine globale Erfassung von Kontexten in der Simulation über definierbare Zeitabschnitte. Dadurch können auch Ereignisse bzw. Teilkontexte, die räumlich (spatial) und ggf. auch zeitlich (temporal) weit auseinander liegen, miteinander in Korrelation gebracht werden. Die Simulation kann also nun zusätzlich auch globale Kontexte wahrnehmen und falls nötig den Simulationsablauf entsprechend anpassen. Die Kontextmuster, die während der Simulation automatisch erkannt werden sollen, können mit Hilfe von deklarativen Regeln im Context-Service definiert werden.

Um nun für den Simulationsverlauf relevante Situationen erkennen und entsprechend darauf reagieren zu können, werden die dafür benötigten Kontextdaten kontinuierlich als Ereignisse an den Context-Service gesendet. Dort kann dann überprüft werden, ob entsprechend definierte Kontextmuster vorliegen und die Anwendung darüber informieren. Die Anwendung kann nun eine entsprechende Reaktion in der Simulation einleiten und sich den aktuellen Gegebenheiten anpassen. Der Context-Service kann also so gesehen den Ablauf der Simulation beeinflussen. Die Anwendung wäre somit context-aware.

3.2 Teilkomponenten der Testumgebung

Die Testumgebung besteht im wesentlichen aus einer Multiagentensimulation, einer *Message Oriented Middleware* zur Übertragung von Nachrichten und dem Context-Service. Die drei Teilkomponenten, werden im Folgenden kurz erläutert.

3.2.1 Multiagentensimulation

Laut einer Evaluation von [RLJ06], eignet sich *Repast* für eine einfache Realisierung und vor allem für eine performante Simulation von komplexen Modellen am besten. Deshalb wird zur Implementierung und Visualisierung des Beispielszenarios, das Simulationstool *Repast Simphony*[1] verwendet. Dabei handelt es sich um ein Open Source Tool, welches neben hilfreichen APIs zur Umsetzung einer Multiagentensimulation auch eine integrierte Visualisierungsmöglichkeit bietet. Die damit realisierte Simulation des Besipielszenarios wird mit dem Context-Service verbunden.

3.2.2 Message Oriented Middleware

Zur Kommunikation zwischen dem Context-Service und der Multiagentensimulation, wird die Open Source Message Oriented Middleware *ActiveMQ*[2] mit einer *Java Message Service* (JMS) Schnittstelle verwendet. Dies dient zur Entkopplung der Kontext-Komponente vom Rest der Anwendung. Über die Middleware werden Kontextdaten als Nachrichten an den Context-Service übermittelt und dessen Antwortnachrichten von der Multiagentensimulation empfangen.

3.2.3 Context-Service

Der Context-Service empfängt kontinuierlich die Snapshots des aktuellen Simulationszustands und wertet diese in Echtzeit aus. Der Context-Service kann grob in vier wesentliche Bestandteile unterteilt werden, die im Folgenden genannt werden:

1. Context Server: Stellt die Verbindung zur Anwendung her und enthält Event Consumer. Ein Event Consumer ist in diesem Fall ein Parser, der die über die Message Queue ankommenden Ereignisse konsumiert und diese an die CEP-Engine zur Auswertung übergibt.
2. CEP-Engine: Übernimmt die kontinuierliche Auswertung der eingehenden Events und überprüft diese auf definierte Ereignismuster. Als CEP-Engine wird hier die Open Source Version von *Esper*[3] verwendet. Die Unterstützung einer SQL-basierten Context Query Language, temporaler Kontextregeln, hoher Performance sowie die freie Verfügbarkeit, sind Argumente für die Verwendung von Esper.
3. Regelbasis: Hier sind die Regeln hinterlegt, welche die Ereignismuster definieren die erkannt werden sollen und bei deren Erkennung eine entsprechende Reaktion ausgeführt werden soll.
4. Rule Listener: Lauscht kontinuierlich, ob ein in der Regelbasis hinterlegtes Ereignismuster erkannt wurde. Hier kann dann eine entsprechende Antwortnachricht generiert werden, die festlegt welche Prozedur in der Endanwendung aufgerufen werden

[1] http://repast.sourceforge.net/repast_simphony.php
[2] http://activemq.apache.org/activemq-590-release.html
[3] http://esper.codehaus.org/esper/download/download.html

soll. Optional können hier neue komplexe Ereignisse erzeugt und direkt wieder zur Auswertung an die CEP-Engine geschickt werden.

Der Aufbau des Context-Service ist in Abbildung 2 dargestellt.

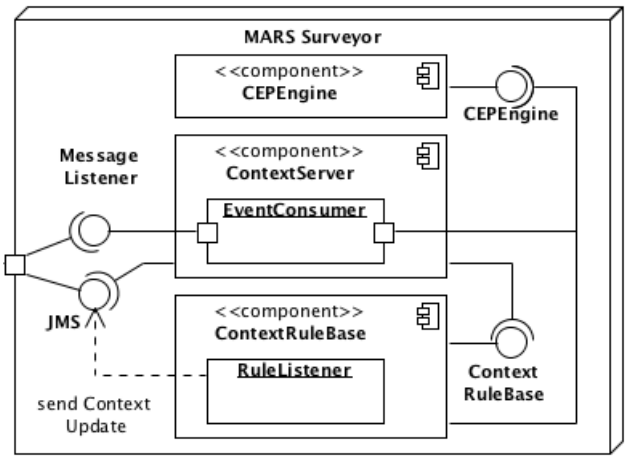

Abbildung 2: Runtime Architektur des Context-Service

3.3 Design

Das in diesem Paper vorgestellte Konzept zur Entwicklung eines Context-Service, löst mehrere der in den bereits bestehenden Arbeiten identifizierten Nachteile. Das Vorgehen zur Lösung der in Abschnitt 2 genannten Probleme, wird nun näher erläutert.

3.3.1 Systemarchitektur

Durch eine vollständige Entkopplung des Context-Service von der Anwendung mit Hilfe einer Message Oriented Middleware (MOM), werden mehrere Probleme gelöst. Da die Kommunikation der Anwendung mit dem Context-Service nun ausschliesslich über Nachrichten stattfindet, bleibt die Endanwendung modular und die technische Umsetzung des Context-Service volkommen transparent. Zusätzlich entfällt die Kommunikation per Java RMI, da auf der Seite des Context-Service primär die Verarbeitung der Kontextinformationen stattfindet und keine unnötige redundante Persistierung bzw. Verwaltung der entsprechenden Daten. Der Ressourcenverbrauch wird dadurch stark minimiert und es wird eine sehr lose Kopplung erreicht. Ein weiterer Vorteil der dadurch entsteht ist, dass die Anwendung nun in einer beliebigen Programmiersprache entwickelt werden kann. Um sich mit der Message Oriented Middleware und letztendlich mit dem Context-Service verbinden zu können, muss lediglich ein *Java Message Service* (JMS) Client in der Anwendung vorhanden sein.

3.3.2 Kontextmodellierung

Die Kontextmodellierung in diesem Ansatz basiert auf einem objektorientierten Kontextmodell [BBH+10, SLP04]. Die Kontextdaten der Anwendung werden dabei als Ereignistypen verwaltet und verarbeitet. Ein Ereignistyp ist vergleichbar mit einer Klasse. Er spezifiziert also die Eigenschaften und die Struktur aller gleichartigen Ereignisse. Ein Beispiel für einen Ereignistyp, der die Kontextinformationen von Geparden abbildet, ist in Abbildung 3 dargestellt. Die Ereignisse können in der Anwendung in einem beliebigen Format vorliegen und als Nachrichten an den Context-Service übertragen werden. Die Übersetzung der Nachrichten, in ein für die CEP-Engine verständliches Ereignisformat, kann dann im Context-Service mit Hilfe eines Adapters erfolgen. Dem Entwickler steht also völlig frei, in was für einer Struktur und in welchem Format die relevanten Kontextinformationen in der Anwendung gespeichert und verwaltet werden. Zusätzlich entfällt die Notwendigkeit vorgegebene Interfaces und die damit implizit vorgegebene Programmiersprache zu verwenden. Insgesamt entsteht der Vorteil, dass auch eine bereits bestehende Anwendung nachträglich ohne Probleme und ohne aufwändige Anpassungen contextaware gemacht werden kann.

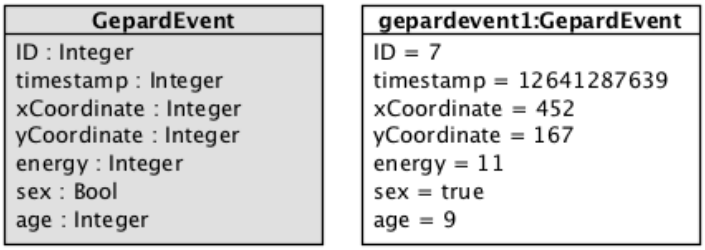

Abbildung 3: Ereignistyp und -instanz

3.3.3 Context Reasoning

In diesem Paper wird die bei JCAF fehlende und bei Gaia begrenzte Möglichkeit des Context Reasoning um wichtige Operationen und die Fähigkeit der Temporallogik erweitert. Die verwendete CEP-Engine unterstützt die Formulierung temporaler und spatiotemporaler Ausdrücke, als auch Operationen wie die Filterung, die Aggregation, das Matching, das Sortieren und das Merging von Kontextinformationen. Dadurch wird nun unter anderem die Erkennung und Auswertung von zeitlichen Zusammenhängen zwischen Ereignissen möglich. Zur Überprüfung wurden dazu in der Regelbasis des Context-Service mehrere komplexe Regeln hinzugefügt. Jeder Regel wurde ein *RuleListener* zugewiesen. Im Test wurden die Regeln erkannt und die entsprechenden Listener aktiv. Dies zeigt, dass das in diesem Paper vorgestellte Konzept die bisherigen Ansätze verbessert und den Nachteil des nicht oder nur begrenzt möglichen Context Reasonings beseitigt.

3.3.4 Abfrage von Kontext

Für die Abfrage von Kontext, wurden bereits mehrere verschiedene Arten von Context Query Languages (CQLs), wie zum Beispiel XML-basierte, RDF-basierte, Graph-basierte oder SQL-basierte CQLs entworfen. Dabei sollte die verwendete Query Language einige wichtige Kriterien und Anforderungen erfüllen [RWK+08]. Eine Übersicht und Bewertung der verschiedenen CQL Typen in [HZK06] ergibt, dass sich neben den RDF-basierten Ansätzen die SQL-basierten Ansätze besonders gut eignen. In diesem Paper wird deshalb eine SQL-basierte CQL verwendet. Dabei orientieren sich die Abfragen an der Basisstruktur gewöhnlicher SQL Querys, weshalb diese Art von CQLs als SQL-basiert bezeichnet werden. Das folgende Beispiel zeigt wie eine Abfrage aussehen könnte, mit der die IDs aller männlichen Geparden die sich in einem Gebiet zwischen 50 und 250 auf der x-Achse aufhalten, abgefragt werden können:

SELECT id **as** gepardId **FROM** GepardEvent **as** gepard **WHERE** gepard.sex = true **and** gepard.xCoordinate **between** 50 **and** 250

Mit der in diesem Paper verwendeten SQL-basierten Context Query Language, lassen sich viele der in [HZK06] genannten Aspekte von Kontext und der in [RWK+08] genannten Anforderungen an eine gute CQL bedienen. Die verwendete CQL bietet damit ein deutlich umfangreicheres Spektrum an möglichen Abfragen von Kontext und Operationen auf Kontextdaten, als die bisherigen Ansätze.

3.3.5 Performance

Die Performance, um die Ereignismengen mehrerer hunderttausend Agenten einer Multiagentensimulation in Echtzeit verarbeiten zu können, soll durch den Einsatz von Complex Event Processing erreicht werden. Der Zwischenschritt, die ggf. riesigen Volumen an zu verarbeitenden Ereignissen zunächst zu speichern, entfällt bei Complex Event Processing. Die Ereignisdaten werden unmittelbar direkt gegen die in der Regelbasis des Context-Service persistierten Ereignisregeln geschickt. Dadurch kann die Verarbeitungsgeschwindigkeit möglicherweise stark gesteigert werden, sodass mehrere hunderttausend Ereignisse pro Sekunde[4] verarbeitet werden können. Dies muss jedoch noch überprüft werden.

4 Schlussfolgerungen und Fazit

Das in diesem Paper vorgestellte Konzept zur Realisierung eines Context-Service zeigt, wie Nachteile bisheriger Ansätze behoben und fehlende Funktionalitäten umgesetzt werden können. Durch grundlegende Designentscheidungen, wie die vollständige Entkopplung des Context-Service von der Endanwendung mit Hilfe einer Message Oriented Middleware, konnten einige Verbesserungen erzielt werden. So wird vor allem die Entwicklung der Endanwendung durch die Verwendung des Context-Service nicht beeinträchtigt. Um

[4] http://.codehaus.org/tutorials/faq_esper/faq.html

den Context-Service verwenden zu können, wird lediglich ein JMS Client benötigt. Durch die lose Kopplung bleibt die Anwendung modular und kann in einer beliebigen Sprache implementiert werden. Die Architektur des Context-Service sowie die technische Umsetzung, bleibt durch die Kommunikation über Nachrichten vollkommen transparent. Dies wird durch die Verwendung eines objektorientierten Kontextmodells in Form von Ereignistypen unterstützt. Es erlaubt eine einfache Modellierung von Kontextstrukturen, die auch in bereits bestehende Anwendungen integriert werden können, ohne diese aufwändig anpassen zu müssen. Durch die Verwendung von Complex Event Processing und einer SQL-basierten Context Query Language, werden temporales Reasoning, umfangreichere Abfragemöglichkeiten und eine performantere Verarbeitung von Kontextdaten ermöglicht.

Um die Leistungsfähigkeit der Context Service-Komponente unter hoher Last zu untersuchen, müssen im weiteren Verlauf der Forschung Performancetests durchgeführt werden. Dabei wird die Anzahl der simulierten Agenten, sowie die Anzahl der in der Regelbasis des Context-Service hinterlegten Ereignisregeln variiert. Zu jeder Konfiguration, wird dann die Anzahl der verarbeiteten Nachrichten und die dafür jeweils benötigte Verarbeitungszeit dokumentiert. Dadurch soll festgestellt werden, wie leistungsstark sich der Context-Service unter den jeweiligen Konfigurationen verhält und wie das Skalierungsverhalten ist.

Bei einem guten Skalierungsverhalten könnte das Konzept letztendlich einen neuen Ansatz für andere Arbeiten bieten, in denen ein entsprechend leistungsfähiger Context-Service benötigt wird, um neue Erkenntnisse erzielen zu können.

5 Danksagung

Ein besonderer Dank gilt Prof. Dr. Thomas Thiel-Clemen, welcher mir bei der Erschließung des Themengebietes geholfen und mich bei der Umsetzung des Projekts mit Ideen und Anregungen tatkräftig unterstützt hat.

Literatur

[ADB+99] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith und Pete Steggles. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, HUC '99, Seiten 304–307, London, UK, UK, 1999. Springer-Verlag.

[Bar05] Jakob E. Bardram. The Java Context Awareness Framework (JCAF); a Service Infrastructure and Programming Framework for Context-aware Applications. In *Proceedings of the Third International Conference on Pervasive Computing*, PERVASIVE'05, Seiten 98–115, Berlin, Heidelberg, 2005. Springer-Verlag.

[BBH+10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan und Daniele Riboni. A Survey of Context Modelling and Reasoning Techniques. *Pervasive Mob. Comput.*, 6(2):161–180, April 2010.

[BDR07] Matthias Baldauf, Schahram Dustdar und Florian Rosenberg. A Survey on Context-Aware Systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, Juni 2007.

[DAS01] Anind K. Dey, Gregory D. Abowd und Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications. *Hum.-Comput. Interact.*, 16(2):97–166, Dezember 2001.

[Eck13] Malte Eckhoff. Ein Agenten-basiertes Gepardenmodell - Konzept und räumliche Wahrnehmung. Hamburg, HH, Germany, 2013.

[GPZ04] Tao Gu, H.K. Pung und Da Qing Zhang. A middleware for building context-aware mobile services. In *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*, Jgg. 5, Seiten 2656–2660 Vol.5, 2004.

[HSP+03] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann und Werner Retschitzegger. Context-Awareness on Mobile Devices - the Hydrogen Approach. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9 - Volume 9*, HICSS '03, Seiten 292.1–, Washington, DC, USA, 2003. IEEE Computer Society.

[HZK06] P.D. Haghighi, A. Zaslavsky und S. Krishnaswamy. An Evaluation of Query Languages for Context-Aware Computing. In *Database and Expert Systems Applications, 2006. DEXA '06. 17th International Workshop on*, Seiten 455–462, 2006.

[KX12] Anup Kumar und Bin Xie. *Handbook of Mobile Systems Applications and Services*. Auerbach Publications, Boston, MA, USA, 1st. Auflage, 2012.

[Luc01] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 2001.

[MN11] C.M. Macal und M.J. North. Introductory tutorial: Agent-based modeling and simulation. In *Simulation Conference (WSC), Proceedings of the 2011 Winter*, Seiten 1451–1464, Dec 2011.

[RC03] Anand Ranganathan und Roy H. Campbell. An Infrastructure for Context-awareness Based on First Order Logic. *Personal Ubiquitous Comput.*, 7(6):353–364, Dezember 2003.

[RLJ06] Steven F. Railsback, Steven L. Lytinen und Stephen K. Jackson. Agent-based Simulation Platforms: Review and Development Recommendations. *Simulation*, 82(9):609–623, September 2006.

[RWK+08] R. Reichle, M. Wagner, M.U. Khan, K. Geihs, M. Valla, C. Fra, N. Paspallis und G.A. Papadopoulos. A Context Query Language for Pervasive Computing Environments. In *PerCom 2008. Sixth Annual IEEE International Conference on Pervasive Computing and Communications, 2008*, Seiten 434–440, 2008.

[SLP04] Thomas Strang und Claudia Linnhoff-Popien. A Context Modeling Survey. In *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.