Schlussbericht

gem. Nr. 8.2 NKBF 98

Für das Vorhaben

Verbundprojekt: Hanseatische Blockchain-Innovationen für Logistik und Supply Chain Management (HANSE-BLOC)

Teilprojekt: Auslotung von Optimierungspotential in der Logistik-Branche durch den Einsatz von Blockchain-Technologie (HANSEBLOC)

Projektpartner

itemis AG Am Brambusch 15-24, 44536 Lünen

Förderkennzeichen

03VNE2044H

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
KMU-NetC Mittelstand

Autor(en)

Dr. Daniel Binkele-Raible

Ort, Datum

Ort, TT.MM.JJJJ

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
KMU-NetC Mittelstand

IHR LOGO

Inhaltsverzeichnis

I. Kurzdarstellung 4

1. Aufgabenstellung 4
2. Voraussetzungen, unter denen das Vorhaben durchgeführt wurde 4
3. Planung und Ablauf des Vorhabens 4
4. Wissenschaftlicher und technischer Stand, an den angeknüpft wurde 5
5. Zusammenarbeit mit anderen Stellen 5

II. Eingehende Darstellung 6

1. Verwendung der Zuwendung und Darstellung des erzielten Ergebnisses im Einzelnen, mit Gegenüberstellung der vorgegebenen Ziele 6
2. Wichtigste Positionen des zahlenmäßigen Nachweises 12
3. Notwendigkeit und Angemessenheit der geleisteten Arbeit 13
4. Voraussichtlicher Nutzen, insbesondere Verwertbarkeit des Ergebnisses im Sinne des fortgeschriebenen Verwertungsplans 13
5. Während der Durchführung des Vorhabens dem Zuwendungsempfänger bekannt gewordener Fortschritt auf dem Gebiet des Vorhabens bei anderen Stelle 13
6. Erfolgte oder geplante Veröffentlichungen des Ergebnisses nach Nr.11. NKBF 98 14

III. Anlage: Erfolgskontrollbericht 15

I. Kurzdarstellung

1. Aufgabenstellung

Smart Contracts sind Programme, die in einer transparenten, gesicherten Umgebung ablaufen und über Zugriff auf Ressourcen verfügen. Sie werden verwendet, um Teile von Vetragsabläufen zu automatisieren. Eingehende Ereignisse werden von Ihnen verifiziert und gegebenenfalls werden Informationsflüsse veranlaßt. Die Definition solcher Smart Contracts ist momentan für Nicht-Programmierer kaum handhabbar. Ziel war es mittels der Entwicklung einer domänenspezifischen Sprache (DSL) für den Logistiksektor diese Hürde deutlich zu senken. Diese Sprache soll frei sein von technischen Aspekten und es Fachexperten aus der Logistik ermöglichen Vetragslogiken zu definieren. Zusätzlich soll es möglich sein diese auf Konsistenz zu prüfen und etwaige Vetragsabläufe zu simulieren. Um die Zugänglichkeit der DSL zu erhöhen, soll ein Vertrag nicht nur in einer für Programmierer typischen Entwicklungsumgebung möglich sein. Das Ziel ist hierfür einen im Browser lauffähigen Web-Editor zu entwickeln, der in eine Web-Kollaborationsplattform integriert ist. Dies soll heißen, daß mehrere Benutzer über einen Editor simultan einen Smart Contract definieren können.

Weitere Anforderungen an Smart Contracts sind:

- Hierarchische Verträge/Rahmenverträge,
- Instanziierung: Ausführung desselben Vertrags für viele "Lieferungen",
- Mapping logischer Signale auf echte Daten/Sensoren,
- Integration nicht formalisierbarer Aspekte,
- Gut/Default-Fall vs. Spezial/Exception-Fall.

2. Voraussetzungen, unter denen das Vorhaben durchgeführt wurde

Um Smart Contracts in das Gesamtsystem integrieren zu können, müssen zwei Vorraussetzungen erfüllt sein. Erstens muss eine Blockchain-Infrastruktur vorhanden sein. Dies ist zum einen nötig da kontinuierlich Sensor-Werte und etwaige Vertragsverletzungen auf der Blockchain im jeweilig aktiven Transportvertrag gesichert werden. Zum anderen wird beim Anlegen einer Smart-Contract-Instanz ein korrespondierender Transportvertrag auf der Blockchain simultan instanziiert.

Zweitens muß klar sein wie die Web-Applikation, also der Teil der Applikation der maßgeblich mit den Benutzern interagiert, mit einem Smart Contract zusammenwirken kann. Hierbei muß klar sein wie mit einer bestimmten Smart-Contract-Instanz der Datenaustausch stattfinden kann.

Dazu müßen geeignete Datenformat und Datenaustauschsmechanismen definiert sein.

3. Planung und Ablauf des Vorhabens

Das Vorhaben ist dem Arbeitspaket 5 (AP5) zu zuordnen, welches wie folgt aufgebaut ist:

AP5.a: DSLs für Verträge und Logistik.

AP5b: Übersetzung von DSL-Instanzen in ausführbaren Code durch Code-Generator.

AP5.c: Verifikation, Validierung, Simulation der DSL.

AP5.d: Entwicklung einer Web-Kollaborationsplattform.

4. Wissenschaftlicher und technischer Stand, an den angeknüpft wurde

Die Smart Contracts werden heute von Entwicklern programmiert. Die fachlichen Inhalte müssen nach wie vor den Entwicklern bzw. Programmierern vermittelt werden, so dass sie von diesen implementiert werden können.
Auch in der Wissenschaft stehen die Implementierungen von Smart Contracts mit sehr informatiklastigen Implementierungen in Blockchains im Mittelpunkt. Die manuelle, programmiersprachenartige Formalisierung von Verträgen ist auch hier gängig.

Im Gegensatz zu spezialisierten DSLs zur Vertragsdefinition werden also Smart Contracts derzeit in generischen Blockchain-Programmiersprachen implementiert. Die Semantik derartiger Programmiersprachen (Serpent, Solidity, etc.) enthält üblicherweise neben der operationalen / funktionalen Ebene auch eine Kostenfunktion, die Instruktionen, Speicherzugriffen, etc. einen Preis zuordnet. Dies soll die verschwenderische Verwendung von Rechenzyklen/Speicherplatz der Virtuellen Maschine minimieren, wirkt sich jedoch an vielerlei Stellen auf die Semantik der Programmiersprache aus und führt daher zwangsläufig zu einer erhöhten Komplexität - und damit zu einer erhöhten Fehlerquote - der Programmerstellung bzw. Vertrags-Implementierung.

Bisherige Arbeiten und Ansätze zur Formalisierung von Domänen, wo bisher eher Verträge informeller Natur gängig sind, sind:
- Lexfi ist eine Sprache zur Beschreibung finanzieller Transaktionen (https://www.lexifi.com/faq/technology/).
- Legalese ist eine Beschreibungssprache für juristische Sachverhalte (http://ec2-18-136-1-48.ap-southeast-1.compute.amazonaws.com/aboutus).
- Hvitved (2012) zeigt in seiner Dissertation (https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.724.7779&rep=rep1&type=pdf) wie man formalisierte Verträge im Kontext eines ERP-Systems über eine Haskell-DSL beschreiben kann.

5. Zusammenarbeit mit anderen Stellen

Innerhalb des Projektes gab es vertiefte Kooperation bezüglich Komponentenarchitektur und Komponentenkommunikaiton mit den Projektpartnern HEC, HAW und Consider-it. Außerhalb des Hansebloc-Projektes gab es keine weiteren Kontakte.

Außerhalb des Arbeitspaketes 5 hat Itemis weitere Beiträge innerhalb weiterer Arbeitspakete geleistet.
- AP1: Mitarbeit an der Erarbeitung von Anwendungsfällen. Insbesondere die Anforderungsanalyse bezüglich der Allgemeinheit der Kontrakte wie sie im Logistikkontext nötig sind ist hervorzuheben. Als Resultat wurde klar, daß der ursprüngliche angedachte Ansatz zu flexible ist. Als Folgerung wurde ein vereinfachter Transportvertrag entworfen. Im Bereich Governance gab es Beiträge bezüglich Voting-Methoden.
- AP2: Im Bereich Systemarchitektur hat eine Mitarbeit im Bereich Security stattgefunden. Die Entwicklung eines Security-Models und dessen Beurteilung fand hier statt. Insbesondere eine Risikoanalyse des Gesamtsystems mittels des Itemis-eigenen Tools ‚Security Analyst' ist hervorzuheben.
- AP3: Designvorschlag zur Kommunikation mit Sensoren und darauf aufbauende Datenaustauschformate..
- AP4: Im Bereich ‚Privacy' hat sich Itemis engagiert und die Themen ‚Access Control' , ‚Blockchain ohne Transaktionskosten' und die Evaluation einer Blockchain-Implementierung. Zur Auswahl der Blockchain wurde R3 Corda evaluiert.

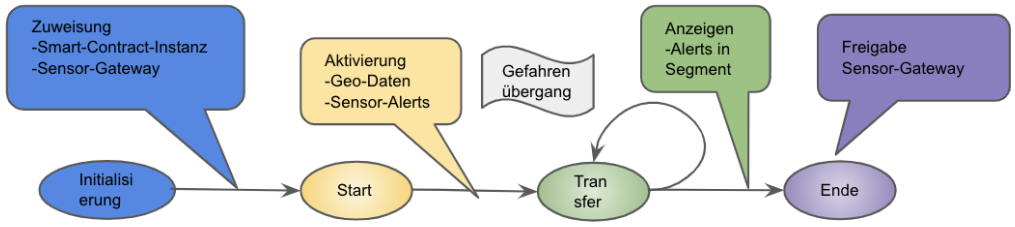
II. Eingehende Darstellung

1. Verwendung der Zuwendung und Darstellung des erzielten Ergebnisses im Einzelnen, mit Gegenüberstellung der vorgegebenen Ziele

AP5.a: DSLs für Verträge und Logistik.

Die DSL wurde ursprünglich als frei formulierbarer Smart Contract entworfen, d.h. es waren beliebige Vertragszustände und ein frei gestaltbarer Kontrollfluss vorgesehen. Ein möglicher Ansatz über Zustandsautomaten wird in dem Itemis-Blog-Eintrag

https://languageengineering.io/a-smart-contract-development-stack-54533a3a503a vermittelt. In diesen Prototyp unterstützt die Syntax ganz klar die freie Wahl der Zustände. Dieser hat sich als zu mächtig herausgestellt. Zum einen ist der Ablauf eines Transportvorgangs in der Praxis sehr einfach:



Zum anderen werden durch den freien Kontrollfluss viele Aspekte des Systems deutlich komplexer:

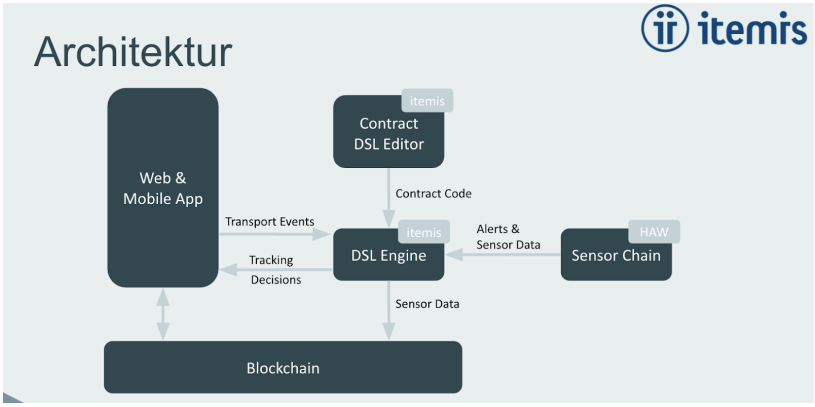
- Der DSL-Code müsste die Ablauflogik der anderen Applikationen steuern, was insbesondere die Web-Applikation deutlich komplexer gemacht hätte. Dies wäre im Rahmen des Projekts nicht umsetzbar gewesen.
- Verifikation, Test und Simulation wären deutlich aufwendiger.
- Verträge wären für Fachanwender aufgrund der komplexen Struktur schwerer nachvollziehbar.

Aus diesem Grund wurde der ursprüngliche, auf endlichen Automaten basierende Ansatz verworfen.

Vereinfachter DSL-Entwurf

In einem neuen, vereinfachten Entwurf ist der Kontrollfluss von vorneherein festgelegt. Nachdem der Transport gestartet wurde, kann es 0-n Gefahrenübergänge geben, bevor der Transport endet. Der Kontrollfluss wird von der Web-Applikation gesteuert; der DSL-Code wird über Events angestoßen.

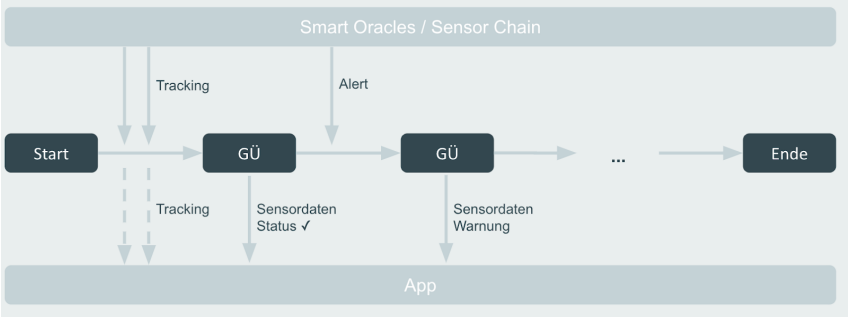
Dies hat eine weitreichende Konsequenz für das Design des Gesamtsystems. Die Web-Applikation steuert jetzt den Transportprozess, und ist nicht mehr auf die DSL angewiesen. Damit gibt die Web-Applikation ein Default-Verhalten vor, das vom DSL-Code beeinflusst werden kann. Die DSL-Komponente wird damit zu einer optionalen Komponente, was der modularen Struktur von Hanse-bloc zugute kommt.



GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung
KMU-NetC
Mittelstand

IHR LOGO

Exemplarisch wird eine Interaktion zwischen der Web-Applikation und der DSL-Engine aufgeführt:



Wurde der SmartContract initialisiert (Fig. 1) dann definiert er für jeden Sensor, der im SmartContract erwähnt wird, einen Wertebereich. Dieser Wertebereich wird von der Sensorchain abgerufen. Sollten die gemessenen Werte außerhalb diese Bereichs liegen, so wird dieser Messwert an die DSL-Engine gesendet. Dieses Vorgehen wurde so gewählt, um die Lebensdauer der Stromquellen der Sensoren zu erhöhen. Wird dieser Wertebereich nicht verlassen, so muss nicht gesendet werden. Geokoordinaten werden mit jedem der Meßwerte mitgeschickt. Damit aber eine kontinuierliche Nachverfolgung unabhängig der Sensormesswerte gewährleistet ist, wird in konstanter Frequenz auch die Geokoordinaten an die DSL-Engine geschickt.

Die Web-Applikation steuert grundsätzlich den Zustand des DSL-Engine. Sie gibt vor, in welchen nächsten Zustand die DSL-Engine wechselt. Beim Gefahrenübergang werden Geo-Daten und Alerts an die Web-Applikation übermittelt. Die Geo-Daten werden unverändert zwischengespeichert und dann weitergereicht. Die Alerts sind das direkte Resultat der Ausführung des SmartContracts, der dem aktiven Transportauftrag zugeordnet ist. Der SmartContract wertet die Messwerte der Sensorchain aus und kann gegebenenfalls Alerts generieren. Beim Gefahrenübergang werden diese Alerts dann der Web-Applikation angezeigt. Die Web-Applikation kann ihrerseits auf Alerts reagieren (z.B. einen Vorbehalt kenntlich machen). Damit die Web-Applikation kontinuierlich und zeitnah über Geo-Daten und Alerts in Kenntnis ist, werden diese Daten nicht nur beim Gefahrenübergang übermittelt. Sie werden über eine Message-Queue mittels Push-Mechanismus übertragen.

Interaktion DSL-Engine und Contract-DSL-Editor

Die DSL-Engine ist, neben der Interaktion mit der Web-Applikation, zuständig für die Ausführung von SmartContracts. Solch ein Kontrakt liegt der DSL-Engine als ausführbarer Java-Code vor. Ein SmartContract wird aber im Contract-DSL-Editor in einer DSL definiert. Die Transformation der SmartContract-DSL-Instanz in Java Code sollte typischerweise durch einen externen Build-Job übernommen werden, der mittels Codegenerierung diese Übersetzung vornimmt und der DSL-Engine über ein Netzlaufwerk bereitstellt.

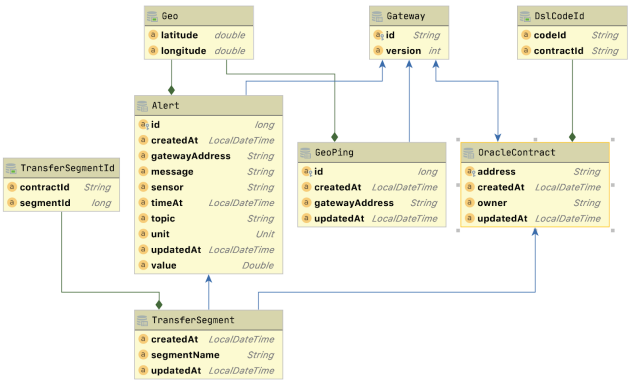
Interaktion Blockchain

Neben der Web-Applikation kommuniziert auch die DSL-Engine mit der Blockchain. Es gibt zwei Modalitäten wann auf die Blockchain geschrieben wird. Wird im Init-Zustand ein Transportauftrag mit einem Sensor-Gateway und einen neu instanzierten SmartContract verbunden, so wird auch ein entsprechender Vertragseintrag auf der Blockchain angelegt.

Ebenso wird im Falle eines Alerts dieser im Vertragseintrag auf der Blockchain dokumentiert.

Architektur DSL-Engine

Das ursprüngliche Ziel einen SmartContract, der über eine DSL definiert wurde, in Solidity-Code (Blockchain-Native-Code) zu überführen, wurde aus oben aufgeführten Gründen verworfen. Damit ist auch die anfängliche Idee die Blockchain als Ausführungsumgebung zu nutzen obsolet. Da gegenwärtig das Codegenerierungsziel Java ist, muss eine neue Ausführungsumgebung bereitgestellt werden. Diese Umgebung ist die DSL-Engine. Ihre Funktion geht über die reine Ausführung des SmartContracts hinaus. Grundsätzlich muß dieser Microservice das Triple Transportvertrag, Gateway und SmartContract verwalten. Insbesondere muß eine Mehrfachverwendung eines Gateways (Abstraktion einer Menge von Sensoren) verhindert werden. Diese Buchhaltung wird durch folgendes Datenbankschema implementiert:



Ein Transportvertrag ist mit einem OracleContract über die ‘contractId’ verbunden.

Prinzipiell werden so folgende dynamischen Daten für einen Transportauftrag festgehalten:

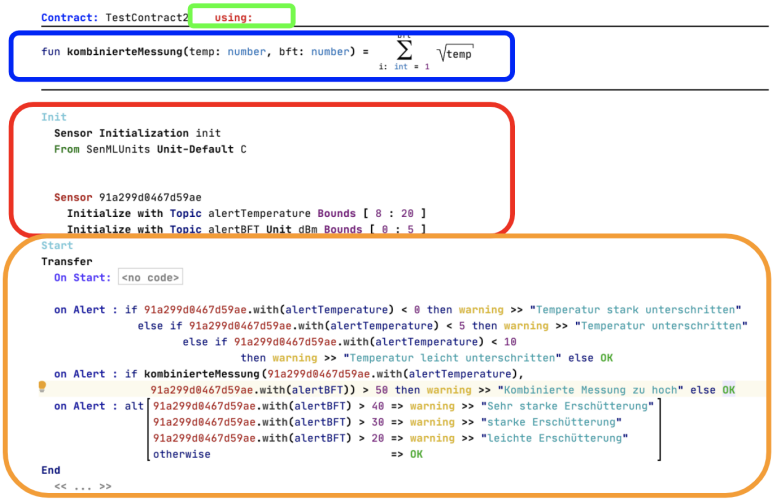
1. Es werden in vorgegebenen Intervallen die Geokoordinaten festgeschrieben.
2. Es wird die Dauer eines Segments des Transportauftrages festgehalten.
3. Die Alerts, die vom SmartContract herrühren, werden gespeichert.
4. Es wird erfasst, von welchem Gateway der Transportauftrag die Sensordaten bezieht.

Um eine Deploybarkeit zu erleichtern wurden mehrere Skripte erstellt:

- Ein reines Docker-Deployment wird in https://github.com/dbinkele/hansebloc/tree/master/engine/docker erklärt. Hierbei muss nur ein Shell-Skript ausgeführt werden.
- Eine Deployment mittels Kubernetes wird in https://github.com/dbinkele/hansebloc/tree/master/engine/kubernetes dargestellt. DSL-Engine und die zugehörige Datenbank muss separat, über jeweilige Shell-Skripten, ausgerollt werden.

Sprachkomposition

Die verwendete DSL ist modular aufgebaut. Konkret bedeutet dies, dass sie aus mehreren DSLs zusammengesetzt ist. Dies ermöglicht im allgemeinen, dass eine DSL in mehreren Kontexten wiederverwertet werden kann. Dies ist ein Alleinstellungsmerkmal von MPS, welches sonst von keiner anderen Language-Workbench unterstützt wird.



In obiger Darstellung sind vier Teile zu erkennen, jeweils mit farbiger Umrandung kenntlich gemacht. Die roten und orangenen Teile sind zwei Teil-DSLs, die eigene Konzepte zur Spezifikation von Verträgen im Logistikkontext mitbringen. Der blaue Teil ermöglicht die Definition eigener Auswertungsfunktionen. Die Basis hierfür bildet die itemis-eigene, funktionale Programmiersprache KernelF. Der grüne Teil ermöglicht das Einbinden dieses Teils auch als Bibliothek. Aber auch der rote und orangene Teil bindet KernelF ein, um z.B. Vergleichsoperatoren, Bedingungen etc. zur Verfügung zu haben.

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
KMU-NetC Mittelstand

IHR LOGO

Zur Zeit ermöglicht es die DSL Kriterien bezüglich der Sensordaten festzulegen, so dass spezifische Alerts das Verhalten der Web-Applikation beeinflussen können. Doch durch die Möglichkeit der DSL-Komposition ist es möglich weitere domänenspezifische Vertragskomponenten einzubinden. Ein Beispiel wäre die Spezifizierung unter welchen Bedingungen die Fracht einem Versicherungsschutz unterliegt. Aufbauend auf kontinuierlich gesendeten Werten der Sensor-Chain könnte dieser Wertebereich spezifiziert werden. Mit zusätzlichen Sensoren, die die Versiegelung von Frachtobjekten überwachen, wäre ein SmartContract für die Zollabfertigung im Transitverkehr denkbar.

Vertragsimplementationen auf der Blockchain

Wie bereits erläutert wurde die Idee verworfen mittels einer Vertrags-DSL Blockchain-Code zu generieren. Die Alternative ist fixe Verträge zu nutzen. In Zusammenarbeit mit Consider-It entstanden zwei Vetragsdefinitionen, die für den konkreten Anwendungsfall relevant sind.

1. Registry-Contract: dient der Identifikation der Teilnehmer in Hanseblock-Netzwerk
2. Transport-Contract: dient der Abbildung der Transportprozesses insbesondere der Gefahrenübergänge. Etwaige Überschreitungen von Grenzwerten bezüglich gemeßener Sensorwerte werden ebenso dokumentiert.

Weitere Details finden sich im Bericht von Consider-it.

Gegenüberstellung zu vorgegebenen Zielen

Folgende Anforderungen für die DSL wurden bei Projektstart festgehalten:

Konkret soll die DSL

1. von der konkreten Ausführungs-Technologie (Blockchain) abstrahieren,
2. die Beschreibung von in der Logistik-Branche üblichen (hierarchischen) Vertragswerken ermöglichen,
3. auf der einen Seite allgemein genug sein, um die automatisierbaren Anteile möglichst vieler der in den Anwendungsfällen beschriebenen Verträge beschreiben zu können, darf jedoch andererseits nicht Turing-vollständig sein, um eine automatische Analyse /Verifikation der Verträge zu ermöglichen, sowie
4. die Vertragsdefinition durch Fachexperten ermöglichen, und
5. das Zusammenstellen der modellierten Vertragsarten einfach und schnell zu ermöglichen indem sie einen direkten Zugriff auf alle relevanten Parameter bietet.

Wie folgt wurden diese Ziele erreicht:

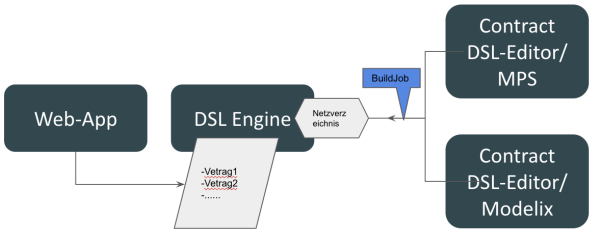
1. Die implementierte DSL wird momentan in Java-Code überführt. Dies ist allerdings für den Anwender der DSL transparent, so daß durch die Implementierung eines weiteren Generators beliebige Platformen als Ausführungs-Technologie nutzbar sind. Damit ist das Ziel erreicht.
2. Dieses Ziel ist teilweise verworfen worden, da dies bedeutet hätte, daß vertrauliche Informationen aus Verträgen öffentlich auf der Blockchain gespeichert würden. Gleichzeitig hat sich in der Diskussion mit unseren Partner aus der Logistik herausgestellt, daß die dort üblichen Vertragswerke weit aus weniger komplex sind. Deshalb sind wir zur Konklusion geklommen, daß eine DSL hierfür nicht gerechtfertigt ist. Stattdessen wurden statische Verträge auf der Blockchain deployt die für den Anwendungsfall eines Transportauftrags ausreichend sind. Der dynamische Teil der Vetragsformulierung umfaßt nun die Auswertung von Sensor-Werten, die von der Sensorchain übermittelt werden.
3. Es ist möglich Teile der DSL zu verifizieren wie im Abschnitt für AP5.c noch erläutert wird.
4. Nach Rücksprache mit den Logistikpartner ist die DSL hierfür ausreichend ausdrucksstark, um gängige Auswertungsbedingungen zu formulieren.
5. Im konkreten Fall hat die DSL zugriff auf das momentane Transportsegment, die Dauer des Transportsegments, Geokoordinaten und auf alle Werte der registrierten Sensoren.

AP5b: Übersetzung von DSL-Instanzen in ausführbaren Code durch Code-Generator.

Im Zuge des Projektes hat sich herauskristallisiert, dass es aufgrund von Geheimhaltungsaspekten nicht möglich ist, beliebige Arten von Transportaufträgen auf der Blockchain auszuführen. Es werden nämlich keine Klardaten abgelegt, sondern nur verschlüsselte Hashwerte. Erstere wären aber nötig, wenn Kontrakte auf der Blockchain spezifisches Verhalten an den Tag legen sollen. “Damit verbietet sich allerdings auch die Verwendung von Smart Contracts, die auf Basis der zu schützenden Daten bestimmte Aktionen ausführen können” ist die Schlussfolgerung aus folgender Publikation, die unter der Beteiligung von Itemis im Rahmen von Hansebloc erarbeitet wurde (https://www.hamburg-logistik.net/fileadmin/user_upload/aktivitaeten/projekte/Hansebloc/twenhoeven-Blockchain-und-Privacy-IM2020-1.pdf).
Eine wichtige Erkenntnis hieraus ist, dass eine Übersetzung von einer DSL in Blockchain-lauffähigen Code nicht nötig ist. Damit erübrigt sich Solidity als Generierungsziel. Als Alternative bot es sich an einen SmartContract, der über die DSL definiert wird, in Java-Code zu übersetzen. Diese Entscheidung bringt einige Vorteile mit sich:

- Im Vergleich mit Blockchain-Code ist Java um ein Vielfaches schneller in der Ausführung.
- Es fallen keine Kosten für die Ausführung an.
- Grundsätzlich kann der Code überall ausgeführt werden, wo ein Java-Umgebung vorhanden ist, also auch auf mobilen Endgeräten.

Um sicherzustellen, dass alle Parteien sich auf den gleichen Vertrag beziehen, muss sein Hashwert auf der Blockchain gespeichert werden. Damit sind jegliche Modifikationen, die außerhalb der Blockchain, am ausführbaren Code gemacht werden identifizierbar. Dies ist im momentanen Prototyp so noch nicht implementiert, stellt aber auch keine technische Hürde dar.
Ein SmartContract wird in einem eigenem Service (DSL-Engine) ausgeführt. Da dieser Sensordaten interpretiert kommuniziert er direkt mit der Sensorchain. Der SmartContract reichert die Rohdaten der Sensoren mit spezifischen Alerts an. Die DSL-Engine ist allerdings nur für die Ausführung des SmartContracts, der hier in Java-Code vorliegt, zuständig.
Die Definition des SmartContracts kann entweder in MPS oder der Web-Platform (Modelix) durchgeführt werden. In beiden Fällen muss nach Abschluss dieser Definition ein Build-Job angestoßen werden, der aus der DSL-spezifischen Definition des SmartContract Java-Code generiert und der DSL-Engine über ein gemeinsames (Netz-)Verzeichnis zur Verfügung stellt. Ist dies abgeschlossen, wird der SmartContract für die Web-Applikation sichtbar und kann benutzt werden.



Gegenüberstellung zu vorgegebenen Zielen

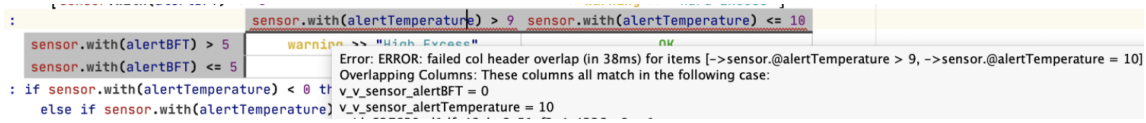
Der entwickelte Generator bildet dein DSL-Instanz auf Java ab, a die Generierung von Solidity (Blockchaincode) verworfen wurde. Im Falle von Solidity hätte die Blockchain selbst als Ausführungsumgebung fungiert. Im Falle von Java ist dies nicht möglich. Deswegen war unter anderem notwendig einen Java-Microservice zu implementieren, der die aus einer DSL-Instanz generierten Java-Artefakte ausführen kann. Dies ist mit dem oben genannten DSL-Engine aufbauend auf dem Spring Framework erfolgt.

AP5.c: Verifikation, Validierung, Simulation der DSL.

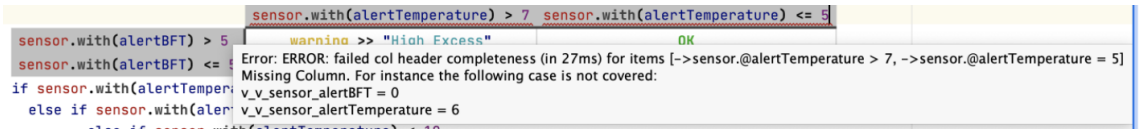
Die neue Vertrags-DSL ist stark an den grundsätzlichen Ablauf in Fig. 1 angepasst. Damit ist die Ablauflogik im wesentlichen festgelegt. Die ursprüngliche Vertragsmethodik hätte ja beliebige Zustandsübergänge erlaubt, weswegen eine Verifikation deutlich komplexer geworden wäre. Unter anderem hätten man verhindern müssen, dass ein Vertrag nicht mehr terminieren kann, da eine Teilmenge an Zuständen aufgrund fehlerhafter Modellierung nicht mehr verlassen werden kann.
Da nun der Zustandsgraph fest vorgegeben ist, entfällt dies. Dadurch dass die DSL die Sensorwerte interpretiert, ist ein wichtiges Ziel den gesamten Wertebereich eines Sensors abzudecken. Ein Beispiel hierfür liefert eine Entscheidungstabelle:

	`sensor.with(alertTemperature) > 9`	`sensor.with(alertTemperature) <= 10`
`sensor.with(alertBFT) > 5`	`warning >> "High Excess"`	`OK`
`sensor.with(alertBFT) <= 5`	`OK`	`OK`

Hier wird über eine Tabelle definiert, wann eine Warnung gegeben wird. Die speziellen Fälle pro Tabelleneintrag sollen überschneidungsfrei sein, d.h. für gegebene Sensorwerte muss sich genau ein Eintrag ergeben, der die Aktion vorgibt. Das ist hier aber nicht der Fall:



Die Verifikation, die durch einen shortcut ausgelöst wird, identifiziert den Wert 0 für ‘alertBFT’ und 10 für ‘alertTemperature’ als Sensordaten, für die sich mehrere Einträge ergeben (nämlich beide Einträge der oberen Zeile). Dieses Beispiel ist noch sehr einfach, allerdings ist diese Validierung auch auf eine beliebige Anzahl von Sensoren und gängige Operatoren (+,-,*,/, &&,||, =>,!) ausdehnbar. Genauso läßt sich ermitteln, ob es einen nicht definierten Bereich gibt:



Die Wertekombination 0 für ‘alertBFT’ und 6 für ‘alertTemperature’ führt zu keinem Eintrag.

Ein weiterer Fall ist das Alt-Tab, welches Bedingungen und deren Folgen als vertikale Aufzählung darstellt. Wie im vorherigen Fall würden eventuelle Überschneidungen angezeigt. Im vorliegenden Fall sind aber Sensorwerte unterhalb der 0 nicht abgedeckt.

Um diesen fehlenden Bereich abzudecken, kann man einen ‘otherwise’ Zweig hinzufügen.

```
alt
    sensor.with(alertBFT) > 20                                    => warning >> "Earthquake"
    sensor.with(alertBFT) > 10 && sensor.with(alertBFT) <= 20 => warning >> "Heavy Shock"
    sensor.with(alertBFT) >= 0 && sensor.with(alertBFT) <= 10 => warning >> "Shock"
    otherwise                                                     => OK
```

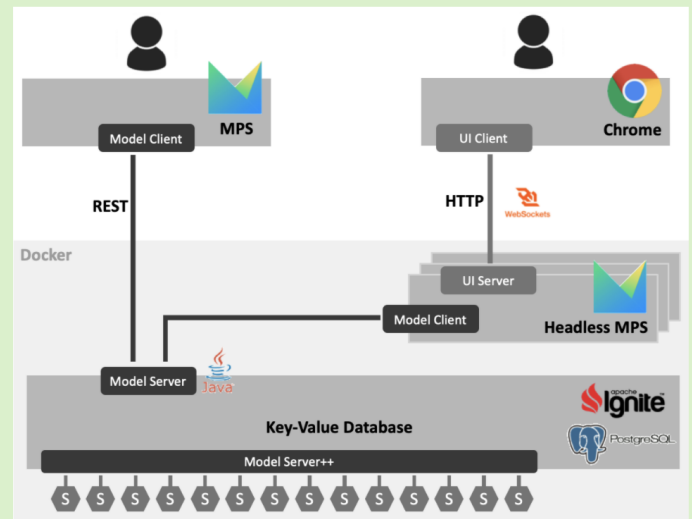
Gegenüberstellung zu vorgegebenen Zielen

In der konkreten DSL geht es darum auf ein Ereignis mit genau einer Handlungsanweisung zu reagieren. Ein Ereignis wird durch einen boolschen Ausdruck über Sensorwerte, Segmentnummer, Segmendauer und Geokoordinaten definiert. Im konkreten Fall ist es möglich Überschneidungen von komplexen Fallszenarien, die über boolsche Ausdrücke spezifiziert werden, zu erkennen. Dies vermeidet, daß es für eine Ereignis mehrere Handlungsanweisung innerhalb des Vertrags gibt. Ebenso können Definitionslücken erkannt werden, welches garantiert, daß für jedes Ereignis auch eine Handlungsanweisung existiert. Dies hilft dabei eventuelle Unstimmigkeiten im Vertrag vorab zu erkennen. Das angestrebte Ziel der Simulation des Vertrages wurde einem Vorgängerprototypen vorgestellt, ist aber aufgrund von Zeitmangel für die finale DSL abschließend nicht erreicht worden. Grundsätzlich stehen hierbei aber keine technischen Hürden im Wege.

AP5.d: Entwicklung einer Web-Kollaborationsplattform.

Die traditionelle Form der Verwendung einer DSL geschieht mittels einer Entwicklungsumgebung (IDE), in unserem Fall Jetbrains MPS. Das ist dasselbe Werkzeug, mit dem auch die DSL definiert wird und etwaige Testfälle definiert sind. Hat man diese feature-reiche IDE das erste Mal vor

sich, dann fühlt man sich insbesondere als Fachanwender ohne expliziten IT-Hintergrund sehr leicht überfordert. Demzufolge ist es nötig, dem Fachanwender einen reduzierten Editor zur Verfügung zu stellen. Dieser sollte nur die Merkmale mitbringen, die für die Handhabbarkeit der Fach-DSL benötigt werden. Gleichfalls ist es wünschenswert, dass die Installation eines schwergewichtigen Editors obsolet ist. Vorwiegend für Gelegenheitsnutzer ist dies von Vorteil.
Unser Lösungsansatz ist eine Web-Kollaborations-Plattform mit Namen ‚Modelix'. Es folgt eine Architekturdarstellung:



Modelix besitzt eine Key-Value Datenbank, in welcher die Modelle (Instanzen einer Sprache/DSL) gespeichert werden. Diese Modelle können parallel von mehreren Nutzern im Browser geöffnet und bearbeitet werden. Jeder Nutzer ist mit einer Headless-Instanz von MPS verbunden, welche Modelländerungen atomar weiter an die Datenbank propagiert. Da Editierungen in Echtzeit propagiert werden ergeben sich keine größeren Versionskonflikte und entstehende könne aufgelöst werden. Hierbei können eventuell in kleinerem Maße Editierungen verloren gehen. Dies ist die Grundlage für Echtzeit-Kollaboration im Stile von Google-Docs. Gleichzeitig ist aber auch möglich sich über MPS über den Model-Server mit der Datenbank zu verbinden. Auf diesem Wege ist es ebenfalls möglich kollaborativ Änderungen an Modellen vorzunehmen.
Im Browser ist es nur möglich das Modell zu editieren; alle anderen Aspekte einer IDE sind nicht vorhanden. Dies ist auch nicht erforderlich, da das Augenmerk hierbei auf der fachlichen Modellierung durch die DSL liegt.
Modelix kann lokal mit Docker betrieben werden, aber auch in der Cloud über Kubernetes.
Weitere Details sind auf folgenden beiden Links zu finden (https://github.com/modelix/modelix, https://blogs.itemis.com/en/modelix-and-the-future-of-language-engineering).
Damit die Projektpartner gegenbenenfalls selbst ein Modelix-Cluster anlegen können, wurden eine Anleitung und Shell-Skripte eingerichtet (https://github.com/dbinkele/hansebloc/tree/master/deployment). Es wird auch exemplarisch dargelegt, wie eine DSL unter Modelix an den Nutzerkreis ausgerollt wird.

Gegenüberstellung zu vorgegebenen Zielen

Wie angestrebt ist es möglich über einen Web-Browser Verträge über die DSL zu definieren, verifizieren und zu analysieren. Als nicht praktikabel erwies sich das Anstoßen der Generierung des Java-Artefakte aus der Web-Umgebung. Dies kann stattdessen mittels eines build-jobs (z.B. in Jenkins oder Team-City) über ein gradle-skript angestoßen werden. In diesem Zuge wurde ein gradle-plugin (https://github.com/modelix/modelix/tree/master/gradle-plugin) entwickelt, welches es ermöglicht vom Modelix-Server das Model der DSL abzurufen. Am Ende sollte der build-job des Java-Artefakt in den Netzordner des DSL-Engine schreiben.

2. Wichtigste Positionen des zahlenmäßigen Nachweises

3. Notwendigkeit und Angemessenheit der geleisteten Arbeit

4. Voraussichtlicher Nutzen, insbesondere Verwertbarkeit des Ergebnisses im Sinne des fortgeschriebenen Verwertungsplans

Folgende Anwendungsfelder haben oder werden von den Ergebnissen des Hanseblockprojekts innerhalb von itemis profitieren:

A. Die Entwicklung einer Web-Platform für den kollaborativen Gebrauch von MPS stellt eine wesentliche Innovation dar. Das Editieren von Modellen über eine DSL (insbesondere Smart Contracts) kann hierbei von Nutzern im Browser vollzogen werden. Dies hat den enormen Vorteil, dass die Hürden zur Nutzung einer DSL für Nicht-Programmier und gelegentliche User wesentlich geringer sind. Dies zeigt sich bereits in der Praxis, wie wir aus zwei Kundenprojekten heraus schon beurteilen können. Insbesondere die Echtzeit-Kollaboration bei der Editierung von Modellen ist im Alltag wichtig. Wir rechnen damit, daß diese Technologie zur dominanten Variante wird, wie MPS genutzt wird. Insbesondere die backend-seitige Komponente zur Synchronisation von Modelländerungen läßt sich in vielerlei Editor-Kontexten einsetzen. Insbesondere können spezifische browsertypische Javascript-Editoren für eine DSL entwickelt werden, welche dieser Komponente die Modelländerungen mitteilt. Ebenso bietet die Backend-Komponente REST-Schnittstellen an, die es uns erleichtern MPS leichter in eine Enterprise-Architektur beim Kunden zu integrieren.

B. Itemis richtet sein Augenmerk in verstärktem Maße auf den Bereich IoT. Das Fotohub-Projekt (https://fotahub.com/) ist ein Beispiel hierfür. Im Hanseblock-Kontext wurden Daten von Sensoren mittels Smart Contracts bewertet. Dies ist eine Blaupause für Anwendungsfälle im weiteren IoT-Kontext: Daten die von IoT-Decvices stammen werden von Code, der durch eine DSL definiert wurde, weiterverarbeitet und bewertet. Um Sensor-Daten auszuwerten hat itemis einen Web-Service entwickelt, der zum einen mit dem Sensor-Netzwerk den Datenaustausch organisiert, als auch zum anderen in der Lage ist den generierten Code eines Smart Contract zu laden und auszuführen. Dadurch entstand eine Infrastruktur mit der Daten mittels des Sen-ML Formats ausgetauscht werden können und anschließend mittels generiertem DSL-Code bewertet werden können. Itemis sieht erhöhtes Potential für weitere Einsätze dieser Architektur in Kundenprojekten im Bereich IoT. Besonders, daß der Endnutzer durch nutzen einer DSL Systemverhalten in einer IoT-Architektur beeinflußen kann ist unserer Ansicht sehr innovativ.

C. Um die Verifizierbarkeit von Teilen des Smart Contract zu bewerkstelligen hat itemis eine Integration zwischen MPS und dem SMT-Solver Z3 weiterentwickelt. Diesbezüglich wurden DSL-Konstrukte in First-Order-Logic Formeln übersetzt, die Z3 versteht. Anhand dieser Formeln, je nachdem, ob sie erfüllbar sind oder nicht, läßt sich eine Aussage darüber treffen, ob eine Verifikation erfolgreich war oder nicht. Die Aufwände die in die Weiterentwicklung geflossen sind, schlagen sich direkt in anderen DSL-Anwendungsbereichen, die eine Verifikation benötigen, nieder. Speziell in unserem Produkt ‚Variability', welches sich mit der Modellierung von Produktvarianten beschäftigt, ist dieser Verifikationsansatz verankert. Die geleistete Arbeit im Bereich Verifikation unterstützt uns dabei weitere Checks in weiteren DSLs zu implementieren. Daneben könnten wir durch verbesserte Integration einen Performance-Gewinn auf Seiten des SMT-Solvers realisieren.

5. Während der Durchführung des Vorhabens dem Zuwendungsempfänger bekannt gewordener Fortschritt auf dem Gebiet des Vorhabens bei anderen Stelle

Mittlerweile hat Jetbrains Projector veröffentlicht. Es ist ein Tool, um IDEA-basierte IDEs wie IntelliJ oder MPS auf dem Server laufen zu

lassen und die Benutzeroberfläche über den Browser ferngesteuert zugänglich zu machen. Es ist im Grunde ein für Jetbrains-IDEs optimiertes Screenshare-Tool. Es projiziert die gesamte MPS-Benutzeroberfläche in den Browser, einschließlich des Modelleditors. Es gibt derzeit keine Unterstützung für den Zugriff mehrerer Benutzer auf dieselbe MPS-Instanz. Eine sinnvolle Kollaboration ist derzeit nicht möglich. In Szenarien, in denen das Ziel darin besteht, über das Netzwerk auf eine IDE zuzugreifen, kann Projector eine bessere Lösung bieten als das, was Modelix im Moment kann. Wenn wir uns die Ziele von Modelix ansehen, wird der Unterschied im Umfang deutlich. Modelix als Plattform, die darauf abzielt, Modelle zu adressierbaren Entitäten in der Cloud zu machen, Modelle zentral mit Blick auf Echtzeit-Kollaboration zu speichern, Infrastrukturprimitive bereitzustellen, um Dienste auf Modellen aufzubauen und Modelle im Browser editierter zu machen.

6. Erfolgte oder geplante Veröffentlichungen des Ergebnisses nach Nr.11. NKBF 98

Eine wissenschaftliche Fachveröffentlichung zum erarbeiteten Privacy-Konzept, welches in Hansebloc Anwendung fand, erschien im Februar 2020 (https://www.hamburg-logistik.net/fileadmin/user_upload/aktivitaeten/projekte/Hansebloc/twenhoeven-Blockchain-und-Privacy-IM2020-1.pdf).

Itemis versucht insbesondere durch Blogpost den Bekanntheitsgrad von Modelix zu erhöhen (https://blogs.itemis.com/en/modelix-and-the-future-of-language-engineering, https://www.itemis.com/en/it-services/methods-and-tools/dsls-mps-deployment-options).

III. Anlage: Erfolgskontrollbericht

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
KMU-NetC Mittelstand

IHR LOGO

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

KMU-NetC
Mittelstand

IHR LOGO