Institut für Architektur von Anwendungssystemen

Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit Nr. 266

Einbindung von Informations-Ressourcen in informelle Prozesse

Benjamin Weder

Studiengang: Informatik

Prüfer: Prof. Dr. Frank Leymann

Betreuer: M.Sc. C. Timurhan Sungur

Beginn am: 16.11.2015

Beendet am: 17.05.2016

CR-Nummer: H.4.1, H.5.3

Zusammenfassung

Informelle Prozesse sind Business Prozesse, bei denen der Großteil der Aktivitäten von Menschen durchgeführt wird. Deshalb sind die Aktivitäten nur schwer vorhersehbar und übliche Modellierungsmöglichkeiten für Business Prozesse können nicht eingesetzt werden. Ein Ansatz ist es, die Aktivitäten implizit zu beschreiben. Dazu werden die im informellen Prozess verwendeten Ressourcen modelliert. Weil zu den Ressourcen auch menschliche Akteure gehören und diese die Aktivitäten durchführen, ist dies eine Möglichkeit einen informellen Prozess indirekt zu beschreiben.

Da informelle Prozesse einen großen Teil der Prozesse in Unternehmen ausmachen, bringt eine teilweise automatische Ausführung dieser Prozesse in vielen Fällen einen erheblichen Gewinn. Um die teilweise automatische Ausführung zu ermöglichen, müssen die Ressourcen von informellen Prozessen automatisch initialisierbar sein. Die vorliegende Arbeit beschäftigt sich mit der Integration von Informations-Ressourcen in informelle Prozesse. Dazu werden Eigenschaften von Informations-Ressourcen analysiert und daraus Anforderungen an eine Software zur Integration von Informations-Ressourcen abgeleitet. Anschließend wird eine Integrations-Software konzeptuell ausgearbeitet, die besonders den Aspekt der einfachen Erweiterbarkeit verfolgt. Diese Eigenschaft ist besonders wichtig, da es sehr viele verschiedene Informations-Ressourcen gibt. So soll die Möglichkeit geschaffen werden, die Software durch Erweiterungen an die Bedürfnisse eines informellen Prozesses anzupassen. Das Resultat der Arbeit ist eine Integrations-Software für informelle Prozesse, die einen Teil der Informations-Ressourcen integrieren kann und zudem einfach um weitere Ressourcen erweitert werden kann. Zur Funktionalität der Software gehört unter anderem das Auflisten und Initialisieren von Ressourcen in informellen Prozessen.

Abstract

Informal processes are business processes in which the majority of activities is performed by humans. Therefore, the activities are difficult to predict and activity centric modeling approaches for business processes can not be used. One possible approach is to describe the activities implicitly by modeling resources of informal processes. The resources modeled in informal processes also include the human actors which carry out the activities and, hence, this is a way to describe activities of an informal process indirectly.

Since informal processes constitute a major part of the processes in many companies, a partial automated execution of these processes can bring a significant benefit. In order to enable the partial automated execution, resources of informal processes must be automatically acquirable. This work deals with the integration of information resources in informal processes. Therefore, characteristics of information resources are analyzed and requirements on a software for the integration are derived. Then an integration software is elaborated conceptually, which particularly heeds the aspect of easy extendability. This property is important because of the large number of different information resources. So the software can be adapted to different informal processes easily. The result of this work is an integration software for informal processes that can integrate a part of the information resources and which can easily be extended to other resources. The functionality of the software includes listing and initializing resources of informal processes.

Inhaltsverzeichnis

1 Einleitung **7**
1.1 Motivierendes Szenario . . . 9
1.1.1 Liste der verwendeten Ressourcen und Relationen . . . 10
1.2 Ziel der Arbeit . . . 11
1.3 Aufbau der Arbeit . . . 12

2 Grundlagen und verwandte Arbeiten **13**
2.1 Topology and Orchestration Specification for Cloud Applications . . . 13
2.1.1 Service Template . . . 13
2.1.2 Definitions Document . . . 15
2.1.3 NodeType . . . 15
2.1.4 RelationshipType . . . 16
2.2 Docker . . . 17
2.3 Dependency Injection . . . 19
2.4 InProXec Methode . . . 20

3 Analyse der Eigenschaften von Informations-Ressourcen **21**
3.1 Taxonomie für Datenquellen im Kontext informeller Prozesse . . . 21
3.2 Einordnung von Datenquellen in die Taxonomie . . . 24
3.3 Relationen zwischen Ressourcen . . . 26

4 Entwurf der Software zur Integration von Informations-Ressourcen **27**
4.1 Anwendungsfälle der Software . . . 27
4.1.1 Anwendungsfälle des Administrators . . . 28
4.1.2 Anwendungsfälle des Benutzers . . . 29
4.2 Klassendiagramm des umgebenden Systems . . . 30
4.3 Klassendiagramm der Software . . . 34
4.3.1 Interfaces zur Erweiterung der Software . . . 36
4.3.2 Ablauf der wichtigsten Methoden . . . 38

5 Implementierung der Software zur Integration von Informations-Ressourcen **43**
5.1 Verwendete Technologien und Libraries . . . 43
5.2 Konfigurationsparameter der Software . . . 46
5.3 Fallstudie . . . 49

6 Zusammenfassung und Fazit **55**

7 Ausblick **57**

8 Abbildungsverzeichnis **58**

9 Listingverzeichnis **58**

10 Quellenverzeichnis **59**

Abkürzungsverzeichnis

BPEL Business Process Execution Language

BPM Business Process Management

BPMN Business Process Model and Notation

DI Dependency Injection

DIPEA Deployable Informal Process Essentials Archive

DM Domain Manager

EEI Execution Environment Integrator

IPE Informal Process Essentials

JDBC Java Database Connectivity

QName Qualified Name

SSL Secure Sockets Layer

URI Uniform Resource Identifier

1 Einleitung

Unter einem *Business Prozess* versteht man einen Prozess, der in einer Firma oder einem Unternehmen durchgeführt wird, um ein bestimmtes Ziel zu erreichen. Ein Business Prozess besteht aus einer Menge von Aktivitäten, die nacheinander oder zum Teil parallel ausgeführt werden. Zum Beispiel kann in einer Bank ein Business Prozess zur Vergabe von Krediten definiert sein. Die Aktivitäten des Prozesses, sind das Annehmen eines Kreditantrags, das Überprüfen der Kreditwürdigkeit des Kunden und abschließend das Informieren des Kunden über die Bewilligung oder Ablehnung des Antrags. Das Designen, Managen, Ausführen, Analysieren und Verbessern von Business Prozessen wird als *Business Process Management* (BPM) bezeichnet [1][2]. Ein wichtiger Bestandteil des Business Process Managements ist das Modellieren von Business Prozessen [3]. Zur Modellierung werden üblicherweise sogenannte Workflow Sprachen wie *Business Process Model and Notation* (BPMN) [4] oder *Business Process Execution Language* (BPEL) [5] verwendet. Diese Sprachen werden genutzt, um die einzelnen Aktivitäten eines Business Prozesses und deren Abfolge zu modellieren und damit den gesamten Prozess zu definieren. Durch die Modellierung von Business Prozessen ist es möglich, das Wissen über die Business Prozesse wiederverwendbar zu machen und zu speichern. Durch das häufige Wiederverwenden der Prozesse, können diese mit neuen Erkenntnissen ständig verbessert werden [1]. Außerdem wird eine komplette oder teilweise automatische Ausführung der Business Prozesse ermöglicht.

Informelle Prozesse unterscheiden sich von anderen Business Prozessen darin, dass die Aktivitäten innerhalb eines informellen Prozesses üblicherweise von Menschen ausgeführt werden und damit schwer vorhersagbar sind [6]. Die Aktivitäten sind deshalb nicht vordefiniert, sondern beruhen auf Entscheidungen der involvierten Menschen [7]. Ein Beispiel für einen informellen Prozess ist die Bearbeitung von Kundenpost in einer Bank. Die Post muss zunächst gelesen werden und anschließend kann anhand des Inhalts entschieden werden, welche Aktivitäten zur Bearbeitung durchgeführt werden müssen. Im Gegensatz zum Beispiel der Kreditvergabe ist es hier schwierig, alle möglichen Aktivitäten im Voraus zu definieren, da der Kunde jede beliebige Anfrage stellen kann und sich die Bearbeitung damit immer unterscheiden kann. Anstelle eines vordefinierten Workflows, hängt die Ausführung bei diesem Beispiel deshalb von den Entscheidungen des Bearbeiters ab, die sich nach seiner Erfahrung und seinem Wissen richten.

Aufgrund der Unvorhersehbarkeit und der häufigen Änderungen der Aktivitäten, sind Aktivitätsorientierte Prozessmodellierungsverfahren wie BPMN oder BPEL zur Modellierung von informellen Prozessen meistens nicht geeignet. Flexiblere Ansätze, wie datengetriebene [8] oder adaptive [9] Workflows, können zur Modellierung ebenfalls nicht verwendet werden, da diese eine vordefinierte Business Logik benötigen, welche für informelle Prozesse meistens nicht existiert [6].

Ohne die Möglichkeit informelle Prozesse modellieren zu können, kann erworbenes Wissen nicht weitergegeben werden. Eine Wiederverwendung des Wissens ist damit nur von der Person, die den Prozess durchgeführt hat, möglich und dies auch nur insoweit sie sich das erhaltene Wissen merken kann. Da in vielen Organisationen informelle Prozesse einen Großteil der Arbeit ausmachen, ist es jedoch wichtig, Wissen wiederverwendbar und insbesondere weiter verteilbar zu machen. Damit sollen Prozesse möglichst unabhängig von der Verfügbarkeit bestimmter Personen mit dem benötigten Wissen gemacht werden.

Der „*Informal Process Essentials*“-Ansatz [6] versucht das Problem der fehlenden Modellierungsmöglichkeiten zu lösen und bietet ein Metamodell zur Beschreibung von informellen Prozessen. Da die Business Logik von informellen Prozessen wegen der Unvorhersehbarkeit nicht explizit definierbar ist, nutzt der Ansatz eine implizite Beschreibung der Business Logik. Das heißt, es werden die menschlichen Akteure beschrieben, die durch ihre Entscheidungen die Business Logik definieren. Für informelle Prozesse sind Ressourcen (zu denen auch die Akteure gehören) häufig von großer Bedeutung und deshalb wurde ein Ressourcen-zentrierter Ansatz vorgeschlagen [6]. Der Fokus eines Modells liegt damit auf den Ressourcen, die im jeweiligen informellen Prozess verwendet werden. Neben den unterschiedlichen Ressourcen, sind der Kontext und die Intention wichtige Bestandteile eines Informal Process Essentials Modells. In der Intention wird das Ziel des informellen Prozesses definiert. Der Kontext wird einmal vor Beginn des informellen Prozesses und einmal nach dem Ende beschrieben, um die Auswirkung des informellen Prozesses deutlich zu machen. Außerdem können unterschiedliche informelle Prozesse durch die Angabe der Kontexte verbunden werden. Das heißt, der eine Prozess kann den Endkontext des anderen Prozesses als Startkontext nutzen.

Die in informellen Prozessen verwendeten Ressourcen können in vier verschiedene Gruppen unterteilt werden: *IT-Ressourcen*, *Material-Ressourcen*, *Wissens-Ressourcen* und *menschliche Ressourcen* [7]. Zu den IT-Ressourcen zählen zum Beispiel verschiedene Programme oder Dateien. Den Material-Ressourcen werden alle physischen Materialien eines Prozesses, wie Bauteile oder Rohmaterial, zugeordnet. Wissens-Ressourcen umfassen das gesamte Wissen, welches innerhalb eines Prozesses verwendet wird. Die menschlichen Ressourcen beinhalten alle am informellen Prozess beteiligten Personen, wie zum Beispiel Entwickler oder Systemadministratoren. Für die teilweise automatische Ausführung von informellen Prozessen, müssen die Ressourcen automatisch initialisiert werden können [7]. Die Ressourcen werden dann genutzt, um die Ziele des Prozesses zu erreichen

Die vorliegende Arbeit beschäftigt sich mit der automatischen Initialisierung von unterschiedlichen Informations-Ressourcen. Zu den *Informations-Ressourcen* zählen zum einen die Wissens-Ressourcen und zum anderen die Daten-Ressourcen, die eine Untermenge der IT-Ressourcen bilden.

1.1 Motivierendes Szenario

Im Folgenden wird ein motivierendes Szenario beschrieben, für das im Verlauf der Arbeit eine *Software zur Integration von Informations-Ressourcen* entworfen wird. Diese soll das automatische Integrieren der im Szenario beinhalteten Informations-Ressourcen ermöglichen. Abb. 1 listet die Schritte des Szenarios grafisch auf, in denen Ressourcen oder Relationen erzeugt, gespeichert, gelöscht oder transferiert werden. Die Zahlenverweise in Klammern innerhalb des Szenarios beziehen sich auf diese Grafik. Die Grafik beinhaltet aus Übersichtlichkeitsgründen nur zwei Beteiligte, ist aber für eine beliebige Anzahl erweiterbar.

Als motivierendes Szenario wurde der Prozess der Wartung einer Software ausgewählt. Die Wartung einer Software entspricht einem informellen Prozess, da die meisten Aktivitäten von Menschen ausgeführt werden müssen und die einzelnen Schritte nicht genau vorhersagbar sind. Je nach Art des Wartungsauftrags werden unterschiedliche Fachleute, wie zum Beispiel Datenbank- oder Netzwerkspezialisten, in den Prozess involviert.

Um das gewonnene Wissen während einer Durchführung des Prozesses teilbar zu machen und für zukünftige Prozesse zu erhalten, werden die Erkenntnisse der Mitarbeiter während des Prozesses in einem für alle Beteiligten zugänglichen MediaWiki[1] gespeichert. Dazu wird im ersten Schritt vom Administrator eine MediaWiki Instanz für den neu gestarteten Prozess erstellt (1). Hierbei kann entweder eine leere Instanz erzeugt werden, falls ein Prozess dieser Art zum ersten Mal ausgeführt wird, oder eine in einem anderen Prozess gespeicherte Instanz neu erzeugt werden. Im zweiten Schritt erhalten alle Beteiligten des Prozesses einen Account mit Administrator Rechten auf der erstellten MediaWiki Instanz, um darüber ihre Erkenntnisse austauschen zu können (2).

Nach der erfolgreichen Erstellung der MediaWiki Instanz können alle Beteiligten mit der Arbeit am Wartungsprozess beginnen. Dafür überträgt zunächst jeder Beteiligte das Code Projekt aus einem globalen Git Repository in sein Dateisystem (3). Anschließend nimmt jeder Beteiligte Änderungen an Komponenten vor, für die er zuständig ist und speichert diese Änderungen in seinem Dateisystem. Nach der Fertigstellung der Änderungen testet der Beteiligte, ob die gewünschte Funktionalität erreicht wurde. Dazu erzeugt er eine MySql[2] Datenbank (4) und fügt dort die für die Tests benötigten Tabellen ein (5). Wenn die Tests fehlschlagen, überarbeitet der Beteiligte die Änderungen, bis die Tests erfolgreich abgeschlossen wurden. Nach dem erfolgreichen Abschluss der Tests, lädt der Beteiligte die von ihm bearbeiteten Komponenten in einen Ordner eines Repositories, in welchem alle Änderungen gesammelt werden (6). Außerdem wird die erzeugte Datenbank mitsamt der Tabellen gelöscht, um keine Ressourcen zu verschwenden (7).

[1] `https://www.mediawiki.org/wiki/MediaWiki/de`
[2] `https://www.mysql.de/`

Nachdem alle Mitarbeiter ihre Änderungen gespeichert haben, ist der informelle Prozess beendet und die MediaWiki Instanz wird passiv gespeichert und gelöscht (8). Das bedeutet, auf die Instanz kann nicht mehr zugegriffen werden, aber sie kann jederzeit im selben Zustand neu erzeugt werden. Durch die passive Verwaltung werden Ressourcen gespart. Neben dem Austausch von Wissen zwischen Beteiligten über die MediaWiki Instanz, können andere Dokumente während des Prozesses über Dropbox ausgetauscht werden (9). Dies ist jedoch optional und nur nötig, falls zusätzliche Dokumente vorhanden sind.

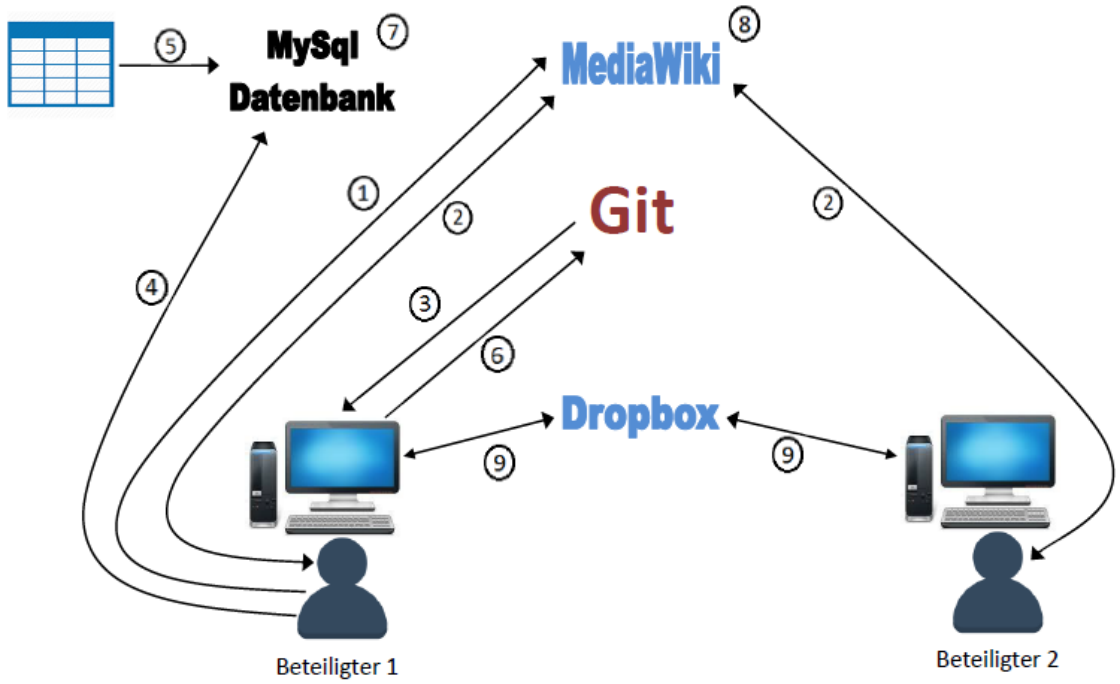

Abbildung 1: Motivierendes Szenario

1.1.1 Liste der verwendeten Ressourcen und Relationen

Nachfolgend werden alle Ressourcen und Relationen, die im motivierenden Szenario benötigt werden und damit in der Software zur Integration von Informations-Ressourcen initialisierbar sein sollen, aufgelistet:

- Ordner im lokalen Dateisystem
- Ordner in Git Repositories
- Ordner in Dropbox Accounts
- Git Repositories
- MySql Datenbanken
- Tabellen in MySql Datenbanken
- MediaWiki Instanzen
- Administrator Relation zwischen Nutzern und MediaWiki Instanzen

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist es, unterschiedliche Informationsquellen, Operationen und Relationen im Kontext der informellen Prozesse zu untersuchen. Aufbauend auf dieser Analyse, soll eine Software zur Integration von Informations-Ressourcen entworfen werden, die es ermöglicht, Datenquellen automatisiert in informelle Prozesse zu integrieren. Diese Software soll anschließend für die Datenquellen des motivierenden Szenarios implementiert werden (siehe Abschnitt 1.1.1). Da die Integration von Informations-Ressourcen im Kontext der informellen Prozesse bisher nicht untersucht wurde, soll die Arbeit diese Lücke schließen.

Abb. 2 zeigt das Vorgehen beim Integrieren von Ressourcen in informelle Prozesse [7]. Die für die Integration interessanten Ressourcen (I1) entsprechen den im motivierenden Szenario aufgelisteten Ressourcen. Die verschiedenen Operationen, welche auf den Ressourcen ausführbar sein sollen (I3), wurden bereits im System, das die Software später als Komponente verwenden soll, definiert. Eine Beschreibung dieser Operationen findet sich in Abschnitt 4.2. Die Implementierung der Software zur Integration von Informations-Ressourcen unterteilt sich in zwei Bereiche. Zum einen die Erstellung von *Domain Managers* (I4) für die verschiedenen Ressourcen, die dafür verwendet werden, alle verfügbaren Ressourcen einer Domäne aufzulisten und alle Informationen zu liefern, welche für den Zugriff auf die Ressourcen benötigt werden. Zum anderen die Realisierung von *Execution Environment Inegrators* (I5). Die Execution Environment Integrators nutzen die von den Domain Managers gelieferten Informationen, um die definierten Operationen auf den Ressourcen auszuführen. Am Ende der Arbeit können die erstellen Domain Manager und Execution Environment Integrator beim System registriert (I6) und verwendet werden.

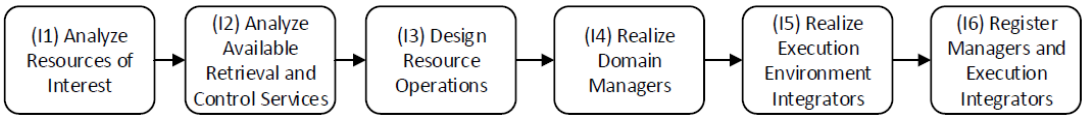

Abbildung 2: Integration von Ressourcen in informelle Prozesse[3]

[3]Übernommen aus Sungur et al. [7]

1.3 Aufbau der Arbeit

Diese Arbeit ist folgendermaßen strukturiert:

Kapitel 2 - Grundlagen und verwandte Arbeiten: In diesem Kapitel werden zunächst Grundlagen eingeführt, die für das Verständnis der weiteren Kapitel nötig sind.

Kapitel 3 - Analyse der Eigenschaften von Informations-Ressourcen: Hier werden Datenquellen untersucht, die im Kontext der informellen Prozesse verwendbar sein können. Es wird eine Taxonomie entworfen, durch welche sich die unterschiedlichen Datenquellen anhand interessanter Eigenschaften kategorisieren lassen. Außerdem werden Relationen untersucht, die zwischen Ressourcen in informellen Prozessen existieren können.

Kapitel 4 - Entwurf der Software zur Integration von Informations-Ressourcen: Die Software zur Integration von Informations-Ressourcen wird in diesem Kapitel konzeptionell ausgearbeitet. Dazu werden zunächst die Anforderungen anhand eines Use-Case-Diagramms analysiert. Anschließend werden die Schnittstellen des Systems, in das die Software integriert werden soll, beschrieben. Am Ende des Kapitels werden die verschiedenen Klassendiagramme der Software entworfen und die Abläufe der wichtigsten Methoden erklärt.

Kapitel 5 - Implementierung der Software zur Integration von Informations-Ressourcen: Dieses Kapitel beschäftigt sich mit der Implementierung der Software zur Integration von Informations-Ressourcen. Hierfür werden zunächst die Technologien und Libraries beschrieben, die für die Implementierung der Software verwendet wurden. Im darauffolgenden Abschnitt wird erklärt, mit Hilfe welcher Konfigurationsdaten die Software auf bestimmte Anwendungen und Datenquellen eingestellt werden kann. Als Abschluss des Kapitels wird eine Fallstudie eingeführt, welche die Verwendung der Software und die gelieferten Rückgaben verständlich darstellten soll.

Kapitel 6 - Zusammenfassung und Fazit: In diesem Kapitel werden die Kernthemen der Arbeit kurz zusammengefasst und es wird ein Fazit gezogen, ob das Ziel der Arbeit mit der implementierten Software zur Integration von Informations-Ressourcen erreicht wurde.

Kapitel 7 - Ausblick: Im letzten Kapitel wird ein Ausblick gegeben, welche Erweiterungen an der Software vorgenommen werden könnten und welche Schritte zur automatischen Integration von Ressourcen in informelle Prozesse noch nötig sind.

2 Grundlagen und verwandte Arbeiten

In diesem Kapitel werden Grundbegriffe, die für das Verständnis der späteren Kapitel elementar sind, eingeführt und gegebenenfalls voneinander abgegrenzt. Viele Begriffe werden dabei bewusst auf Englisch verwendet, da der Gebrauch der englischen Begriffe üblich ist oder keine geeignete deutsche Übersetzung existiert.

2.1 Topology and Orchestration Specification for Cloud Applications

Topology and Orchestration Specification for Cloud Applications (TOSCA) [10] ist ein OASIS-Standard (dieser Standard wird im Weiteren als TOSCA-Standard bezeichnet). Der TOSCA-Standard führt ein Metamodell ein, das zur Beschreibung von Cloud-Anwendungen genutzt werden kann. Das heißt, es wird beschrieben, wie ein Modell für eine Cloud-Anwendung aussieht und welche Elemente dafür verwendet werden. Der TOSCA-Standard nutzt die XML-Syntax zur Beschreibung von Modellen und Elementen. Für die Erstellung der Software zur Integration von Informations-Ressourcen ist der TOSCA-Standard wichtig, weil er *Portabilität*, *Automatisierung* und *Interoperabilität* der Software gewährleistet. Da der TOSCA-Standard sehr umfangreich ist, wird in diesem Abschnitt lediglich die grobe Struktur eines Modells, ein sogenanntes Service Template, und anschließend die für den Kontext der Arbeit wichtigen Definitions Documents, NodeTypes und RelationshipTypes beschrieben. Weitere Informationen zu TOSCA finden sich zum Beispiel im TOSCA-Standard [10] und in weiteren Veröffentlichungen zu TOSCA [11][12].

2.1.1 Service Template

Die Struktur eines *Service Templates* wird in Abb. 3 dargestellt. Die zwei Hauptbestandteile sind ein *Topology Template* und *Pläne*. Außerdem beinhaltet es eine Menge von Node Templates und Relationship Templates.

Das *Topology Template* beschreibt die Struktur der Anwendung, die mit dem Service Template modelliert werden soll. Es enthält dazu die verschiedenen Komponenten der Anwendungen und stellt außerdem die Beziehungen der Komponenten untereinander dar. Zur Darstellung einer Komponente wird ein Node Template erzeugt und falls eine Relation zu einer anderen Komponente besteht, wird ein Relationship Template verwendet, um die beiden Node Templates zu verbinden. Jedes Node und Relationship Template referenziert einen NodeType bzw. RelationshipType. Die Types definieren Operationen, Eigenschaften und Semantiken der zugehörigen Templates. Der Vorteil der Types ist die leichte Wiederverwendbarkeit in anderen Topology Templates [10].

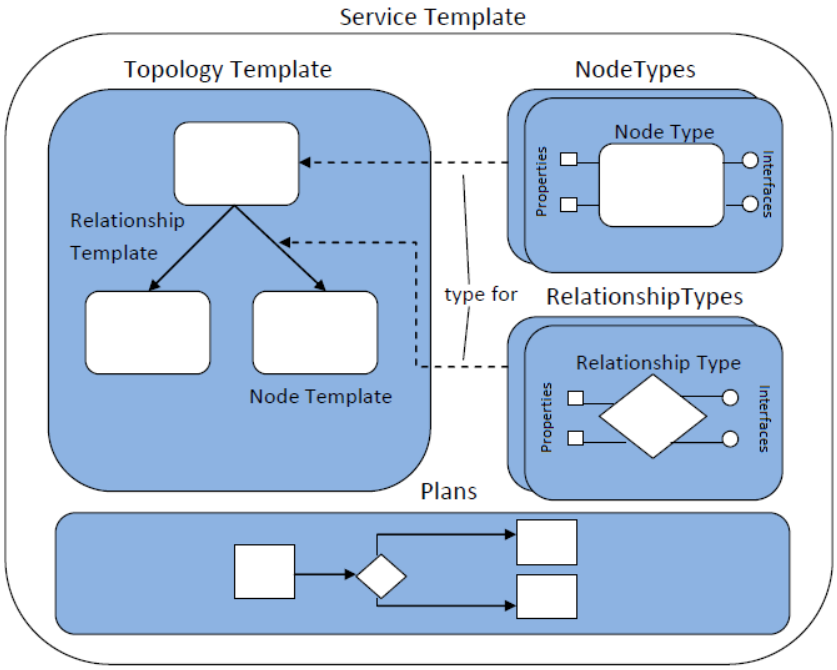

Abbildung 3: Aufbau eines Service Templates[4]

Eine Cloud-Anwendung kann zum Beispiel aus einem Online Warenhaus und einem Web Server bestehen, wobei das Warenhaus auf dem Web Server gehostet wird. Ein Service Template für diese Anwendung beinhaltet ein Topology Template mit jeweils einem Node Template für das Warenhaus und den Web Server und einem Relationship Template „hosted-on“. Das Relationship Template verbindet dabei die beiden Node Templates. Eine graphische Darstellung des Topology Template für dieses Beispiel ist in Abb. 4 gegeben.

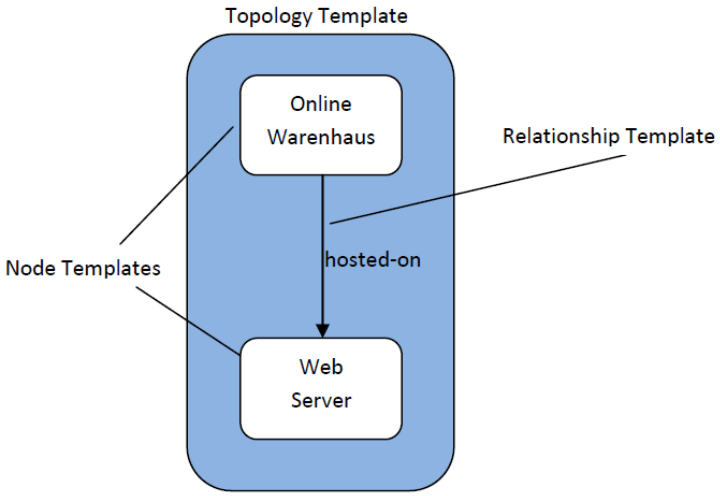

Abbildung 4: Beispiel „Topology Template“

[4]In Anlehnung an [10]

Die *Pläne* hingegen beschreiben die Managementaspekte der Anwendung und werden üblicherweise als Workflow in einer Sprache wie BPMN [4] oder BPEL [5] definiert. Zu den Managementaspekten gehören das Erzeugen der Anwendung, das Management während der Lebenszeit und die Terminierung der Anwendung. Pläne sind für den Kontext dieser Arbeit nicht relevant und werden deshalb nicht näher betrachtet.

2.1.2 Definitions Document

Ein TOSCA *Definitions Document* kann dazu verwendet werden, alle in einem bestimmten Kontext benötigen TOSCA Elemente in einem Dokument zu definieren. Der Inhalt eines Definitions Documents ist deshalb je nach Anwendungszweck sehr unterschiedlich und kann aus allen TOSCA Elementen wie zum Beispiel NodeTypes, RelationshipTypes, RequirementTypes oder Service Templates bestehen [10]. Außerdem können Definitions Documents Erweiterungen und Importe beinhalten. Über eine Erweiterung können die TOSCA Definitionen um zusätzliche domänenspezifische Informationen erweitert werden. Importe können dagegen zum Beispiel externe TOSCA Definitionen sein. Diese können dann im Definitions Document verwendet werden, ohne sie dort erneut definieren zu müssen. Außerdem können XML-Dateien durch die Importe in ein Definitions Document eingefügt und dann zum Beispiel in einem NodeType referenziert werden.

2.1.3 NodeType

Ein TOSCA *NodeType* definiert den Type von einem oder mehreren Node Templates [13]. Dazu beschreibt ein NodeType zum Beispiel die Struktur der Eigenschaften oder die Schnittstellen, die zugehörige Node Templates besitzen [10]. Der Vorteil der Types ist dabei, dass im NodeType definierte Eigenschaften für alle zugehörigen Node Templates gelten und nicht bei jedem einzelnen Template definiert werden müssen. In NodeTypes können viele verschiedene Elemente gesetzt werden, aber für den Kontext der Arbeit sind lediglich der *Name*, der *TargetNamespace* und die *PropertiesDefinition* interessant. Der Name und der TargetNamespace können verwendet werden, um einen NodeType eindeutig zu identifizieren. Über die PropertiesDefinition hingegen wird ein XML Element oder Type identifiziert, das die Struktur der Eigenschaften des NodeTypes beschreibt.

In Listing 1 ist ein beispielhafter NodeType dargestellt. Dieser repräsentiert die Ressource „EigeneDateien“. Über den angegebenen Namen und TargetNamespace kann der NodeType eindeutig identifiziert und referenziert werden. Außerdem ist eine PropertiesDefinition vom Typ „element“ angegeben. Dies bedeutet die Eigenschaften des NodeTypes sind in einem XML-Element mit dem dort angegebenen *QName* (= qualified name) definiert.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <NodeType name="EigeneDateien" targetNamespace="http://www.uni-
      stuttgart/resources/data-resources/file">
3           <PropertiesDefinition element="http://www.uni-stuttgart/
              resources/data-resources/xsd/file/FileProperties"/>
4 </NodeType>
```

Listing 1: Beispiel „NodeType XML“

Im Kontext der Arbeit werden TOSCA NodeTypes verwendet, um verfügbare Ressourcen, also zum Beispiel Ordner oder Repositories, darzustellen. Wenn eine Ressource in einen informellen Prozess eingebunden werden soll, wird ein Node Template vom Type der gewünschten Ressource erstellt. Durch die Initialisierung des Node Templates, wird eine neue Instanz der Ressource des NodeTypes erzeugt. Das Node Template entspricht dann genau einer Instanz der Ressource des NodeTypes. Dadurch ist es möglich, beliebig viele Instanzen einer Ressource zu erstellen, indem Node Templates mit einem bestimmten Type erzeugt und initialisiert werden.

Wenn also der NodeType „EigeneDateien“ einen bestimmten Ordner im lokalen Dateisystem als Ressource definiert, können durch die Erzeugung und Initialisierung von Node Templates des Types „EigeneDateien“ beliebige Instanzen dieser Ressource für einen informellen Prozess kreiert werden.

2.1.4 RelationshipType

TOSCA *RelationshipTypes* stehen im selben Verhältnis zu Relationship Templates, wie NodeTypes zu Node Templates. Sie werden genutzt, um Definitionen für mehrere Relationship Templates durch nur eine einzelne Definition durchzuführen. Im Gegensatz zu NodeTypes, besitzen RelationshipTypes einige andere Elemente wie zum Beispiel SourceInterfaces und TargetInterfaces. Weitere Informationen dazu können im TOSCA-Standard [10] nachgeschlagen werden. Für die Verwendung in der Software zur Integration von Informations-Ressourcen sind jedoch nur die selben drei Elemente wie für NodeTypes interessant. Dazu gehören der Name, der TargetNamespace und die PropertiesDefinition.

Die RelationshipTypes werden im Kontext der Arbeit verwendet, um verfügbare Relationen zwischen Ressourcen darzustellen. Zur Erzeugung einer Relation kann ein Relationship Template des gewünschten Types erstellt und initialisiert werden. Das Relationship Template steht damit für eine spezielle Instanz einer Relation. Die Relation kann dabei, je nach Art, zwischen zwei beliebigen Ressourcen oder nur einer eingeschränkten Auswahl erzeugt werden.

2.2 Docker

Bei *Docker* handelt es sich um eine open-source Plattform zur Entwicklung und Ausführung von Anwendungen [14]. Das Besondere an Docker ist, dass die Anwendungen in komplett eigenständigen Containern ausgeführt werden [15]. Das bedeutet, der Container beinhaltet alles, was die Anwendung zur Ausführung benötigt. Diese spezielle Architektur ermöglicht es, die Anwendung von der Infrastruktur zu trennen und unabhängig zu betrachten [14].

Die Architektur von Docker ist in Abb. 5 dargestellt. Grundsätzlich unterteilt sich Docker in drei verschiedene Komponenten. Der Docker Client, der Docker Daemon und die Docker Registry.

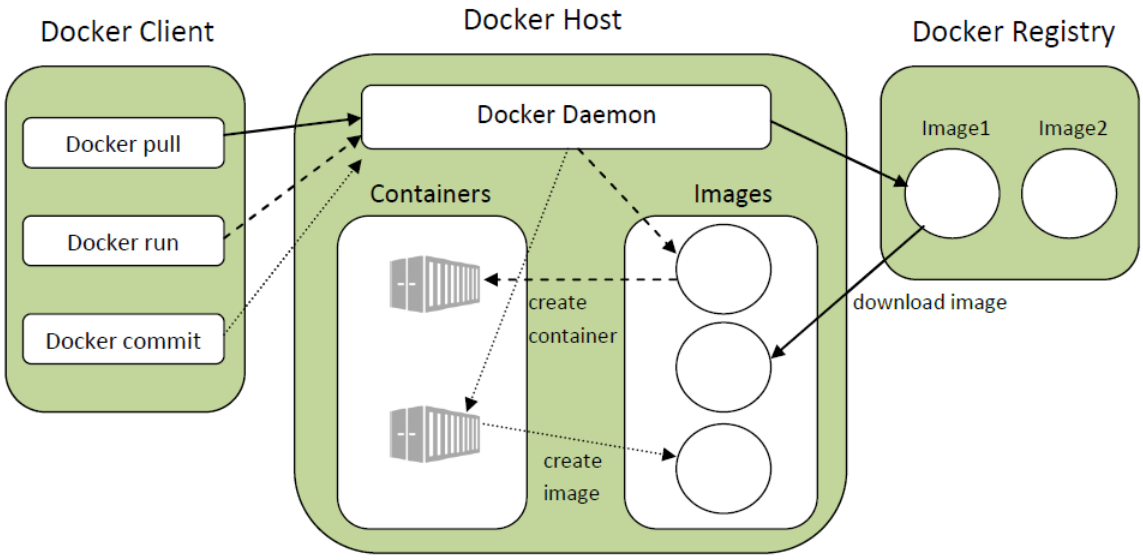

Abbildung 5: Docker Architektur[5]

Beim *Docker Client* handelt es sich um eine Nutzerschnittstelle zu Docker. Der Docker Client liegt lokal beim Nutzer vor. Er wird für die gesamte Kommunikation mit den Docker Diensten verwendet. Der Nutzer gibt die Befehle in den Docker Client ein oder nutzt dazu eine API und der Client leitet diese Befehle mittels eines REST-based Web Services an den Docker Daemon bzw. die Docker Registry weiter [16].

Der *Docker Daemon* ist einer der wichtigsten Bestandteile von Docker und führt die eigentlichen Anwendungen aus. Er kann sowohl lokal vorliegen, als auch remote genutzt werden. Dies wird durch eine REST-based API ermöglicht, über welche vom Docker Client auf den Docker Daemon zugegriffen werden kann [16]. Im Docker Daemon werden Images und Container verwendet. *Docker Container* entsprechen den *aktiven* Anwendungen in einem

[5]In Anlehnung an [14]

Docker Daemon. Sie können erzeugt, gestartet, gestoppt und gelöscht werden [14]. *Docker Images* hingegen sind *passive* Komponenten. Sie repräsentieren die Baupläne von Docker Containern. Jeder Container wird aus einem bestimmten Image erzeugt. Ein Image kann dazu verwendet werden, eine beliebige Anzahl an Containern zu erzeugen. Die Entwicklung von Anwendungen verschiebt sich damit zur Entwicklung von Images, die dann zur Erzeugung von Instanzen der Anwendung genutzt werden können. Neben dem durch das Image vorgegebene, grundsätzliche Verhalten eines Docker Containers, können zusätzliche Konfigurationsparameter, wie zum Beispiel welcher Port verwendet werden soll, bei der Erzeugung eines Containers übergeben werden.

Eine *Docker Registry* beinhaltet Docker Images. Es existieren viele verschiedene Registries, darunter die offizielle Docker Registry und Registries von privaten Entwicklern [15]. In einer Registry können Images gesucht und anschließend gedownloaded werden. Außerdem gibt es die Möglichkeit, eigene Images zu erstellen oder existierende Images zu bearbeiten und anschließend in eine Registry hochzuladen. Die Registry kann also dafür verwendet werden, Images bzw. Anwendungen, zum Beispiel innerhalb einer Firma, zu verteilen. Außerdem ist es möglich Images von aktiven Containern zu erstellen und damit den aktuellen Stand einer Anwendung zu speichern. Anschließend kann der Container gestoppt und gelöscht werden um Ressourcen zu sparen. Durch das Image in der Registry kann jederzeit eine Anwendung mit dem *exakt* gleichen Zustand neu erstellt werden.

Der Ablauf der Docker Operationen „pull“, „run“ und „commit“ ist in Abb. 5 beispielhaft abgebildet. Die Operationen werden zunächst alle im Docker Client eingegeben und von diesem an den Docker Daemon weitergeleitet. Für die „pull“ Operation (durchgezogene Linie) kontaktiert der Docker Daemon die Registry und downloadet von dieser das gewünschte Image. Die anderen beiden Operationen werden direkt im Docker Daemon ausgeführt. Die „run“ Operation (gestrichelte Linie) wird genutzt um einen neuen Container aus einem Image zu erzeugen. Die „commit“ Operation (gepunktete Linie) führt hingegen die gegengesetzte Operation aus und erstellt aus einem laufenden Container ein Image.

2.3 Dependency Injection

Unter *Dependency Injection* (DI) versteht man ein Entwurfsmuster für Software, bei dem die Abhängigkeiten der Objekte erst zur Laufzeit eingesetzt werden. Das Grundkonzept der Dependency Injection ist die „*Inversion of Control*“ [17], also die Umkehrung der Steuerung. Das heißt, die Abhängigkeiten eines Objekts werden nicht vom Objekt selbst konfiguriert, sondern werden von einem externen Objekt eingefügt. Der Vorteil der Dependency Injection ist, dass durch die Nutzung eine losere Kopplung der Komponenten möglich ist. Damit sind Änderungen an Komponenten vornehmbar, ohne dabei andere Komponenten ändern zu müssen. Außerdem ist es möglich verschiedene Implementierungen einer Funktionalität auszutauschen. Für die Nutzung von Dependency Injection wird üblicherweise eine *Dependency Injection Library* verwendet. Diese stellt die Objekte zur Verfügung, welche die Abhängigkeiten suchen und einsetzen.

Ein Beispiel für eine Dependency Injection Library für die Programmiersprache Java ist *Spring* [18]. Mit Spring ist es möglich verschiedene Implementierungen eines Interfaces zur Laufzeit zu laden. Dazu müssen lediglich alle Klassen, die das Interface implementieren mit „@Service“ annotiert werden. Damit wird den Spring Komponenten mitgeteilt, dass es sich um eine Implementierung handelt, die von Spring verwaltet werden soll. In der Klasse, welche die Implementierungen des Interfaces verwenden möchte, können dann zur Laufzeit alle verfügbaren Implementierungen geladen werden. Dafür muss nur der Name des Interfaces und außerdem alle Pakete, in denen Implementierungen liegen, bekannt sein.

Eine weitere Depenedency Injection Library ist *Guice*[6]. Während das Spring Framework neben der Dependency Injection auch andere Aspekte, wie z.B. die aspektorientierte Programmierung, unterstützt, adressiert Guice hauptsächlich das Problem der Dependency Injection. Für die Nutzung als Dependency Injection Library haben beide Frameworks Vor- und Nachteile [19]. Guice hat eine bessere Performanz als Spring und nutzt zudem keine String Bezeichner. Bei Spring werden die verschiedenen Komponenten über diese String Bezeichner identifiziert, was es unmöglich macht zwei Objekte mit demselben Namen zu verwenden. Dies kann in großen Systemen zu Problemen führen, wenn ein Name aus Versehen doppelt genutzt wird. Spring dagegen hat den Vorteil gegenüber Guice, dass Klassen von fremden Parteien leichter eingebunden werden können. Für die vorliegende Arbeit sind beide Frameworks gut geeignet, da es sich bei der entwickelten Software um kein großes System handelt und keine Klassen von fremden Parteien mittels Dependency Injection eingebunden werden müssen. Aufgrund der großen Bekanntheit wurde in der Software Spring als Dependency Injection Library verwendet.

[6] `https://github.com/google/guice`

2.4 InProXec Methode

Die *InProXec Methode* ist eine Methode zur Ausführung von informellen Prozessen [7]. Die Methode unterteilt sich in drei Phasen. In der ersten Phase werden die Services entwickelt, die Informationen über Ressourcen liefern und Operationen auf den Ressourcen ausführen. Zu den Operationen zählen zum Beispiel das Erzeugen einer neuen Instanz einer Ressource oder das Freigeben einer existierenden Instanz. In der zweiten Phase werden die informellen Prozesse modelliert. In dieser Phase entsteht ein sogenanntes *Informal Process Essentials Model* (IPE Model). Das entstandene IPE Model wird in der letzten Phase zunächst kompiliert, womit ein *Deployable Informal Process Essentials Archive* (DIPEA) erzeugt wird. Anschließend wird das DIPEA initialisiert und der informelle Prozess damit ausgeführt. Weitere Informationen zur InProXec Methode und den einzelnen Phasen finden sich in Sungur et al. [7].

Die vorliegende Arbeit beschäftigt sich mit der Erzeugung der Services in der ersten Phase der InProXec Methode. Die benötigten Services sind zum einen *Domain Managers* (DM) und zum anderen *Execution Environment Integrators* (EEI). Domain Managers werden verwendet, um Informationen über verschiedene Ressourcen in einer einheitlichen Art und Weise zu liefern und damit Heterogenität zu vermeiden. Diese Informationen können für die Modellierung von informellen Prozessen verwendet werden. Außerdem können die Informationen für EEIs bereitgestellt werden. Execution Environment Integrators werden genutzt, um Operationen auf Ressourcen auszuführen. Durch die Execution Environment Integrators soll ebenfalls Heterogenität vermieden werden und stattdessen ein einheitlicher Service zur Ausführung von Operationen entstehen.

3 Analyse der Eigenschaften von Informations-Ressourcen

In diesem Kapitel werden unterschiedliche Datenquellen, darin enthaltene Informations-Ressourcen und Relationen zwischen Informations-Ressourcen und anderen Ressourcen analysiert. Dazu wird in Abschnitt 3.1 eine Taxonomie für Datenquellen erstellt, welche wichtige Eigenschaften enthält und das Kategorisieren der Datenquellen im Kontext der informellen Prozesse ermöglicht. In Abschnitt 3.2 werden zur Nutzung der Taxonomie drei Beispiele gegeben. Abschließend werden in Abschnitt 3.3 Relationen zwischen Ressourcen analysiert, die im Rahmen der informellen Prozesse auftreten können.

3.1 Taxonomie für Datenquellen im Kontext informeller Prozesse

In diesem Abschnitt wird eine *Taxonomie für Datenquellen* im Kontext informeller Prozesse entworfen. Für den Entwurf der Taxonomie wurde eine Literaturrecherche durchgeführt und analysiert, welche Eigenschaften von Datenquellen für eine spätere Integration interessant sind. Die Taxonomie ist nicht vollständig und beschränkt sich zur besseren Übersicht und einfacheren Einordnung auf einige wenige wichtige Eigenschaften. Das Ziel der Taxonomie ist es, durch die Einordnung verschiedener Datenquellen Gemeinsamkeiten und Unterschiede festzustellen. Anhand dieser Gemeinsamkeiten und Unterschiede kann bewertet werden, ob die Erweiterung der Software zur Integration von Informations-Ressourcen um eine bestimmte Datenquelle möglich ist und welche Komponenten dazu in welcher Art erweitert werden müssen. Zum Beispiel ist eine Erweiterung sehr einfach, wenn das Protokoll, das von der neuen Datenquelle verwendet wird, schon von einer anderen, bereits in der Software integrierten, Datenquelle genutzt wird. Handelt es sich um ein neues Protokoll, ist die Erweiterung dagegen deutlich aufwendiger.

Abb. 6 zeigt die entworfene Taxonomie für Datenquellen als Baumstruktur. Dabei wurde die Taxonomie für „Data Grids“ von Venugopal et al. [20] für den Kontext der Datenquellen in informellen Prozesse angepasst.

Art der Ressourcen (E1):
Die erste Eigenschaft die betrachtet wird, ist die Art der Ressourcen, die von einer Datenquelle angeboten werden. Dabei wird zwischen strukturierten Daten und unstrukturierten Daten unterschieden. Bei strukturierten Daten handelt es sich um Daten, die ein bestimmtes Schema zugrunde liegen haben und damit dieselbe Struktur aufweisen [21][22]. Strukturierte Daten werden hauptsächlich in Datenbanken wie zum Beispiel DB2[7] oder

[7] `http://www-01.ibm.com/software/data/db2/`

MySql[8] gespeichert. Unstrukturierte Daten hingegen sind alle anderen Arten von Daten und entsprechen dem Großteil der Daten. Dazu zählen zum Beispiel Dokumente, Grafiken oder E-Mails. Diese Unterscheidung ist für das Integrieren der Datenquellen wichtig, da die Verarbeitung der Daten innerhalb der Software unterschiedlich durchgeführt werden muss. Zum Beispiel können unstrukturierte Daten nicht ohne Weiteres in Datenquellen gespeichert werden, die ausschließlich strukturierte Daten verwalten können. Sie müssen dafür erst in ein strukturiertes Format gebracht werden. Auf der anderen Seite macht die Darstellung von strukturierten Daten in Datenquellen für unstrukturierte Daten häufig ebenfalls nur wenig Sinn, da diese für den Menschen sehr schwer lesbar sind.

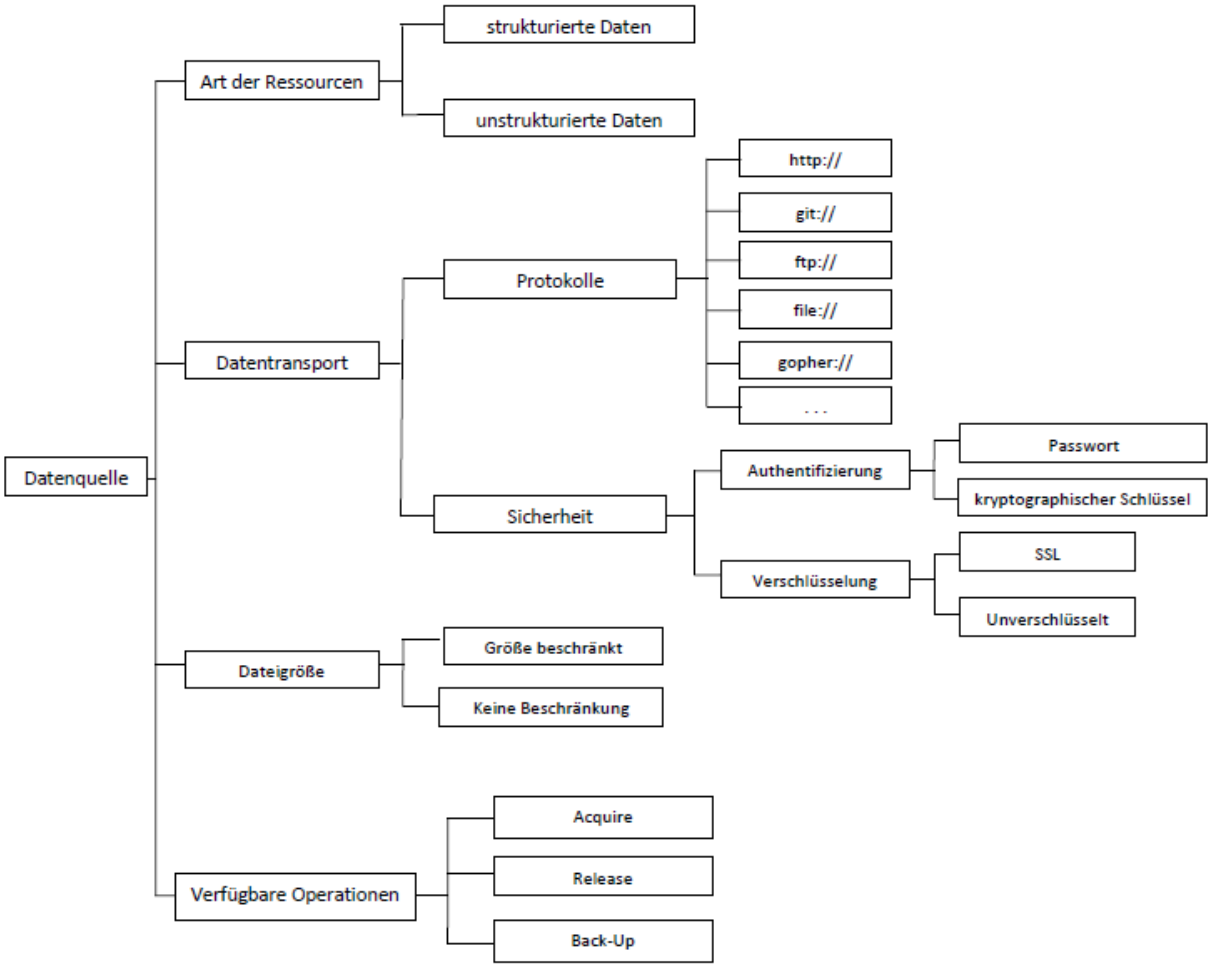

Abbildung 6: Taxonomie für Datenquellen

Datentransport (E2):
Die zweite Eigenschaft ist der Datentransport. Der Datentransport wird in Protokolle und Sicherheit unterteilt. Unter der Protokoll-Eigenschaft sind verschiedene Protokolle, wie zum Beispiel http:// oder ftp://, aufgelistet, die häufig verwendet werden. In der Liste

[8] `https://www.mysql.de/`

konnten nur standardisierte Protokolle aufgenommen werden und sie ist nicht vollständig. Für die Verwendung anderer Protokolle kann die Taxonomie an dieser Stelle jedoch leicht erweitert werden. Die Protokolle sind für die spätere Integration von Datenquellen interessant, da für jedes Protokoll ein neues Plugin für die Software erzeugt werden muss, welches den Zugriff ermöglicht. Wird hingegen ein Protokoll mit bereits vorhandenem Plugin verwendet, ist diese Erweiterung deutlich einfacher. Die Sicherheit teilt sich zum einen in die Authentifizierung, welche per Passwort oder kryptographischen Schlüssel durchgeführt werden kann, und die Verschlüsselung. Zur Verschlüsselung verwenden die meisten Datenquellen entweder Secure Sockets Layer (SSL) oder nutzen gar keine Verschlüsselung [20].

Dateigröße (E3):
Als nächste Eigenschaft wird die Dateigröße von Daten betrachtet, die eine Datenquelle bereitstellt. Dabei gibt es die Möglichkeit, dass die Datenquelle die Größe beschränkt oder beliebige Größen verwendet werden können. Die Größe von Daten kann die Art, in der die Daten bereitgestellt werden beeinflussen und sollte deshalb als gesonderte Eigenschaft betrachtet werden. Für sehr große Daten kann es zum Beispiel unmöglich bzw. sehr ineffizient sein, eine Kopie der Daten zu erstellen.

Verfügbare Operationen (E4):
Die letzte untersuchte Eigenschaft der Taxonomie ist die Verfügbarkeit bestimmter Operationen. Diese Eigenschaft ist im Hinblick auf die Integration von Informations-Ressourcen in informelle Prozesse von besonderem Interesse. Betrachtet werden dabei insbesondere die Operationen „*Acquire*“ (bereitstellen einer Kopie der Ressource zum Bearbeiten), „*Release*“ (freigeben der Ressource) und „*Back-Up*“ (speichern der bearbeiteten Ressource). Diese Operationen sind für das Integrieren der Ressourcen elementar. Datenquellen, welche in der Taxonomie so eingeordnet sind, dass sie eine oder mehrere Operationen nicht unterstützen, sind schwieriger zu integrieren. Für diese Datenquellen muss die Software zur Integration von Informations-Ressourcen die vorhandenen Operationen nutzen, um die nötigen Operationen nachzubilden. Ist dies nicht möglich, kann die betreffende Datenquelle nicht automatisch integriert werden. Die Taxonomie kann an dieser Stelle um weitere Operationen erweitert werden, falls diese bei der Integration gewünscht werden.

3.2 Einordnung von Datenquellen in die Taxonomie

Im folgenden Abschnitt soll die Einordnung in die erstellte Taxonomie exemplarisch für die drei Beispiele Apache Webserver[9], FileZilla Server[10]und das Linux Dateisystem durchgeführt werden, um die Verwendung der Taxonomie zu illustrieren. Neben der textuellen Einordnung der Datenquellen zeigt Abb. 7 eine graphische Möglichkeit der Nutzung der Taxonomie.

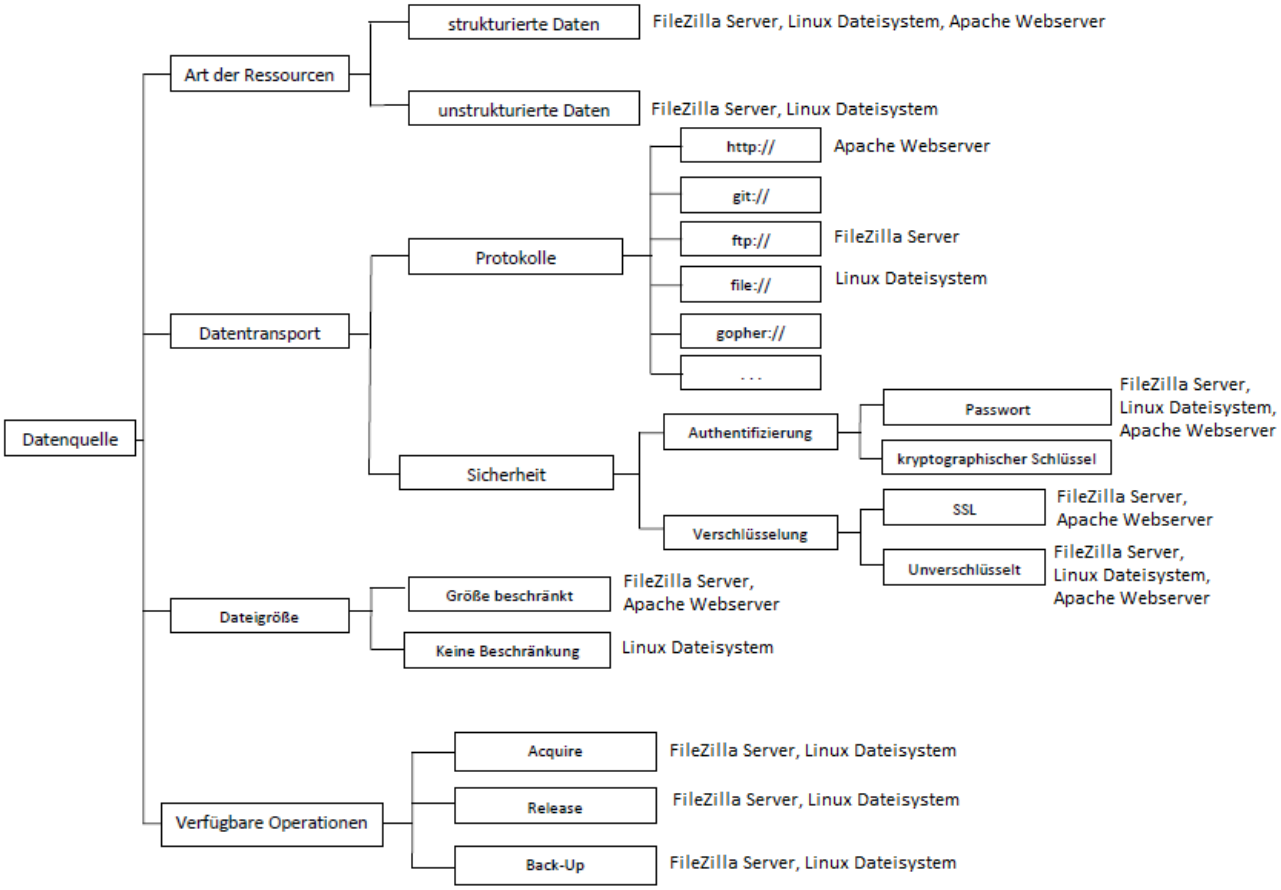

Abbildung 7: Beispiel „Einordnung von Datenquellen“

Apache Webserver:
Beim Apache Webserver handelt es sich üblicherweise um eine Datenquelle für strukturierte Daten. Auf dem Webserver werden HTML bzw. XML Dateien gespeichert, die beide ein strukturiertes Format besitzen. Als Protokoll zum Datentransfer wird http:// bzw. https:// verwendet. Bei Webservern ist der Zugriff auf viele Ressourcen häufig ohne Authentifizierung möglich. Bestimmte Ressourcen können aber durch eine Kombination von Benutzername und Passwort geschützt werden. Die Verschlüsselung hängt beim Apache Webserver von dem verwendeten Protokoll ab. Mit der Verwendung von http:// wird

[9] `https://httpd.apache.org/`
[10] `https://filezilla-project.org/`

beim Datentransport keine Verschlüsselung durchgeführt. Wird dagegen das Protokoll https:// verwendet, findet eine Verschlüsselung mittels SSL statt. Die maximale Dateigröße, welche auf den Server geladen werden darf, kann bei der Konfiguration des Webservers eingestellt werden. Wenn der Nutzer des Webservers gleichzeitig der Administrator ist, kann er die maximale Dateigröße an die benötigte Größe anpassen und es existiert damit praktisch keine Beschränkung. Da dies aber häufig nicht der Fall ist, muss allgemein von einer Beschränkung ausgegangen werden. Die Operationen „Acquire“, „Release“ und „Back-Up“ werden vom Apache Webserver nicht direkt implementiert und müssen deshalb zur Verwendung in der Software zur Integration von Informations-Ressourcen mittels vorhandener Operationen nachgebildet werden.

FileZilla Server:
Der FileZilla Server gehört zur Kategorie der Datenquellen für strukturierte und unstrukturierte Daten. Der Server kann zum Beispiel Textdateien speichern, die zu den unstrukturierten Daten zählen. Außerdem können aber auch strukturierte Daten, wie zum Beispiel XML Dateien, auf dem Server abgelegt werden. Zum Datentransport wird das ftp:// bzw. sftp:// Protokoll verwendet. Wie beim Apache Webserver auch, hängt die Verschlüsselung von der Wahl des Protokolls ab. Während beim Protokoll ftp:// keine Verschlüsselung genutzt wird, verwendet das sftp:// Protokoll eine SSL-Verschlüsselung. Die Authentifizierung wird üblicherweise per Benutzername und Passwort durchgeführt. Es gibt jedoch auch sogenannte anonyme FTP-Server, die den Zugriff ohne Authentifizierung zulassen. Die maximale Dateigröße kann ähnlich dem Apache Webserver vom Administrator des Servers festgelegt werden und ist damit allgemein beschränkt. Die Operation „Acquire“ entspricht dem Anfordern einer Datei des FTP-Servers mit der „Get“-Methode, weil dabei eine lokale Kopie der Datei erzeugt wird. Das Herunterladen der Datei ist aufgrund der beschränkten Größe problemlos möglich. „Back-Up“ kann mittels „Put“ realisiert werden, da die bearbeitete Datei beim Übertragen an den Server alte Vorkommen überschreibt. Für die Operation „Release“ muss lediglich die erzeugte Kopie gelöscht werden.

Linux Dateisystem:
Das Linux Dateisystem ist eine Datenquelle für strukturierte und unstrukturierte Daten, da dort alle Arten von Daten gespeichert werden können. Als Protokoll zum Zugriff auf Ressourcen wird file:// verwendet. Eine Verschlüsselung ist nicht nötig, da der Datentransport nur innerhalb des lokalen Rechners stattfindet. Zur Authentifizierung kann ein Benutzername und ein Passwort verwendet werden. Die Größe der Dateien, die verwaltet werden können, ist (abgesehen von der Festplattengröße) üblicherweise nicht beschränkt. Die Operationen „Acquire“, „Release“ und „Back-Up“ sind beim Linux Dateisystem nahezu identisch vorhanden, da sie dem Erzeugen, Löschen und Speichern einer lokalen Kopie entsprechen.

3.3 Relationen zwischen Ressourcen

Neben den Datenquellen und den zugehörigen Daten sind Relationen zwischen Ressourcen ein wichtiger Bestandteil zur Integration von Informations-Ressourcen. Eine Relation verbindet dabei immer zwei verschiedene Entitäten. Die Relationen können zum Beispiel genutzt werden, um zu notieren, dass zur Integration einer bestimmten Informations-Ressource eine andere Informations-Ressource ebenfalls benötigt wird. Betrachtet werden Relationen zwischen Informations-Ressourcen und menschlichen Ressourcen, sowie IT-Ressourcen. Relationen zwischen Informations-Ressourcen und menschlichen Ressourcen sind von besonderer Bedeutung, da die menschlichen Akteure den informellen Prozess ausführen. Durch eine Relation kann eine Informations-Ressource so eingestellt werden, dass der menschliche Akteur sie zum Erreichen des Ziels des Prozesses verwenden kann. Die Relationen zwischen Informations-Ressourcen und IT-Ressourcen sind wichtig, da durch sie die Beziehungen und Abhängigkeiten zwischen verschiedenen Services dargestellt werden kann. Relationen zu Material-Ressourcen sind dagegen im Kontext der Arbeit weniger interessant, da diese ebenfalls hauptsächlich Relationen zu menschlichen Ressourcen besitzen, welche die Materialen verwenden.

Relationen zwischen Informations-Ressourcen und menschlichen Ressourcen: Eine mögliche Relation zwischen einer menschlichen Ressource und einer Informations-Ressource ist „uses“. Die Relation verbindet einen menschlichen Akteur mit einer Ressource, wenn der Akteur die Ressource während des informellen Prozesses verwendet. Eine weitere Relation wäre „isAuthorized“. Diese Relation verbindet einen Menschen mit einer Ressource, falls er berechtigt ist diese zu lesen und zu bearbeiten. Eine Initialisierung der Relation erteilt dem Menschen die Berechtigung zum Lesen bzw. Schreiben und ermöglicht ihm dadurch die Ressource für den informellen Prozess zu verwenden. Die Relation „isResponsible“ verbindet Akteure mit den Ressourcen, für die sie verantwortlich sind. Eine Relation „isAdministrator“ könnte hingegen zwischen einem Akteur und einem Service erstellt werden, wenn der Akteur Administrator Rechte für den Service besitzt.

Relationen zwischen Informations-Ressourcen und IT-Ressourcen:
Zwischen zwei Informations-Ressourcen stellt „dependsOn“ eine mögliche Relation dar. Diese Relation stellt die temporale Ordnung zwischen den beiden Informations-Ressourcen her. Durch die Relation kann also zum Beispiel eine Reihe verschiedene Dateiversionen als Kette von Entitäten, die durch „dependsOn“-Relationen verbunden sind, dargestellt werden. Die Relation „uses“ kann eine Verbindung zwischen einer Softwarekomponente und für die Software benötige Dateien herstellen. Eine weitere mögliche Relation wäre „needs“. Diese Relation kann eine Informations-Ressource mit einer Software-Ressource verbinden, wenn die Informations-Ressource eine bestimmte Software benötigt, um gelesen werden zu können. Bei der Initialisierung der Informations-Ressource müssen alle Software-Ressourcen initialisiert werden, zu denen sie eine „needs“ Relation besitzt.

4 Entwurf der Software zur Integration von Informations-Ressourcen

In diesem Kapitel werden Anforderungen an die Software zur Integration von Informations-Ressourcen definiert und die Software wird entworfen. In Abschnitt 4.1 werden zunächst Anforderungen mithilfe eines Use-Case-Diagramms dargestellt. Dazu werden gewonnene Informationen aus der Taxonomie in Abschnitt 3 verwendet. Anschließend werden in Abschnitt 4.2 die Schnittstellen des Systems, in das die Software eingebunden werden soll, als Klassendiagramm beschrieben. Anhand dieser Schnittstellen wird definiert, welche Operationen die Software unterstützen soll, welche Parameter übergeben werden und welche Rückgabe die Software liefern soll. In Abschnitt 4.3 werden schließlich die Klassendiagramme für die Komponenten der Software entworfen, welche die nötigen Operationen realisieren.

4.1 Anwendungsfälle der Software

In Abb. 8 werden verschiedene Anwendungsfälle der zu implementierenden Software vorgestellt. Die Anwendungsfälle beziehen sich zum einen auf die Nutzung der Software und zum anderen auf die mögliche Erweiterung durch Administratoren.

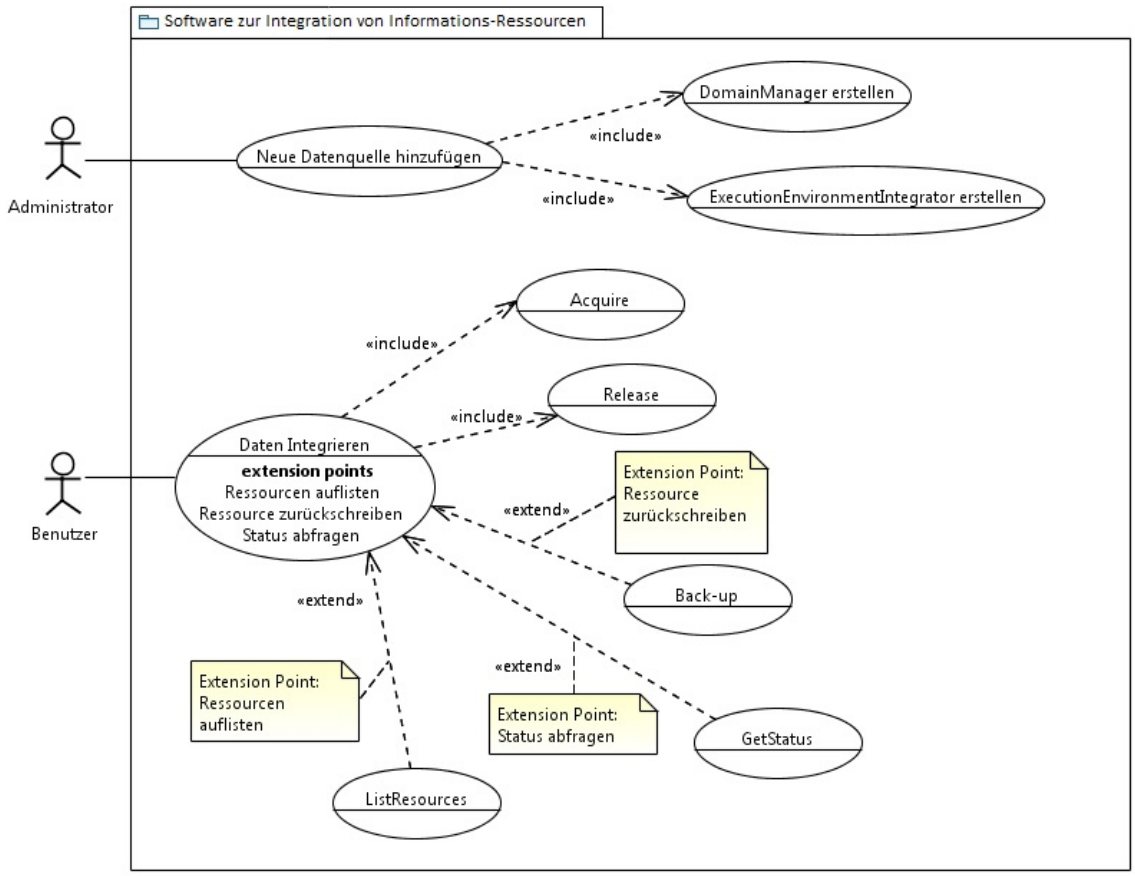

Abbildung 8: Use-Case-Diagramm der Software

In den beiden folgenden Abschnitten werden die verschiedenen Anwendungsfälle des Administrators bzw. des Benutzers genauer betrachtet und erläutert.

4.1.1 Anwendungsfälle des Administrators

Neue Datenquelle hinzufügen
Der Administrator erweitert die Software um eine zusätzliche Datenquelle, die zukünftig integrierbar sein soll. Um dies einfach zu ermöglichen, soll die Software über eine sogenannte Pluggable-Architektur verfügen. Diese ermöglicht es Erweiterungen separat zu schreiben und dann in die vorhandene Software „einzustecken“. Dieser Anwendungsfall schließt die Anwendungsfälle „Domain Manager erstellen“ und „Execution Environment Integrator erstellen“ ein. Vor dem Durchführen dieser beiden Anwendungsfälle, sollte der Administrator die neue Datenquelle in die in Abschnitt 3.1 erstellte Taxonomie einordnen. Damit sind die wichtigsten Eigenschaften der Datenquelle explizit dargestellt und das Erstellen der beiden Plugins wird erleichtert.

Domain Manager erstellen
Der Administrator erstellt einen Domain Manager (siehe Abschnitt 2.4) für die neue Datenquelle. Dieser stellt dem System die nötigen Ressourcen- und Relationen-Definitionen der neuen Datenquelle zur Verfügung (*ListResources* Operation). Außerdem liefert der Domain Manager die nötigen Informationen, wie zum Beispiel Zugangsdaten einer Datenquelle, die ein Execution Environment Integrator zum Ausführen der verschiedenen Operationen benötigt.

Execution Environment Integrator erstellen
Der Administrator erstellt einen Execution Environment Integrator (siehe Abschnitt 2.4) für die neue Datenquelle. Der Execution Environment Integrator kann die Ressourcen und Relationen der Datenquelle für menschenzentrierte Prozesse bereitstellen. Die Ausführung aller Operationen, außer ListResources, wird vom Execution Environment Integrator mithilfe der von einem Domain Manager gelieferten Informationen durchgeführt.

4.1.2 Anwendungsfälle des Benutzers

Daten Integrieren

Der Benutzer möchte mithilfe der Software zur Integration von Informations-Ressourcen eine Ressource oder Relation in einen informellen Prozess integrieren. Der Anwendungsfall beinhaltet die folgenden beiden Anwendungsfälle:

- „Acquire“
- „Release“

Außerdem können die drei folgenden Anwendungsfälle erweitert werden:

- „ListResources“, falls dem Benutzer zunächst die verfügbaren Ressourcen angezeigt werden sollen.
- „Back-up“, falls die bearbeiteten Ressourcen gespeichert werden sollen.
- „GetStatus“, falls der Status einer Ressource abgefragt werden soll.

Diese Operationen müssen für jede verfügbare Datenquelle zur Verfügung stehen. Für Datenquellen, welche die Operationen nicht direkt ermöglichen, muss die Software die Lücke zwischen den verfügbaren und benötigten Operationen schließen.

Acquire

Der Benutzer ruft die Funktion *Acquire* für eine bestimmte Ressource auf. Die Software erzeugt daraufhin eine neue Instanz von der Ressource und stellt diese dem Benutzer zur Bearbeitung bereit. Für die Bereitstellung der Ressource ist die Größe der Datei ein interessanter Aspekt (siehe Abschnitt 3.1 (E3)). Das heißt, für sehr große Ressourcen kann die Bereitstellung anders durchgeführt werden, als für kleinere Ressourcen, da von den großen Ressourcen keine Kopie erstellt werden kann. Stattdessen muss dem Benutzer in diesem Fall die vorhandene Kopie zugänglich gemacht werden. Da bei der Bearbeitung damit jedoch die alte Ressource überschrieben wird, muss der Benutzer über diesen Umstand informiert werden oder die Bereitstellung dieser Ressource muss abgelehnt werden. Die Funktion Acquire wird üblicherweise zum Beginn eines informellen Prozesses für alle benötigten Ressourcen aufgerufen.

Release

Der Benutzer ruft die Funktion *Release* für eine Ressource auf, die mit Acquire angefordert wurde. Die Software gibt die Ressource frei und löscht die erzeugte Instanz der Ressource. Release kann nur bei Ressourcen ausgeführt werden, die zuvor mit Acquire angefordert wurden. Die Release Funktion wird im Gegensatz zu Acquire am Ende eines informellen Prozesses aufgerufen, um überflüssige Ressourcen freizugeben.

Back-up
Der Benutzer möchte seine Änderungen an einer bestimmten Ressource speichern und ruft dazu die Funktion *Back-up* auf. Die Software speichert die bearbeitete Instanz und informiert den Benutzer über den Speicherort, um die zukünftige Benutzung der Ressource zu ermöglichen. Die Instanz wird dabei nicht zwangsläufig in derselben Datenquelle gespeichert wie die ursprüngliche Ressource, da die Software auch das Verwenden von Datenquellen ermöglichen soll, die nur Lesezugriff erlauben. Für die Back-up Operation ist die Unterscheidung wichtig, ob es sich um strukturierte oder unstrukturierte Daten handelt (siehe Abschnitt 3.1 (E1)), da dies die Auswahl einer Datenquelle zum Speichern der Ressource beeinflusst.

GetStatus
Der Benutzer möchte den Status einer bestimmten Datenquelle oder Ressource abfragen. Die Software ruft daraufhin bei der entsprechenden Datenquelle den Status ab und gibt ihn dem Benutzer aus.

ListResources
Der Benutzer möchte sich einen Überblick über verfügbare Ressourcen und Relationen verschaffen und ruft die Funktion *ListResources* auf. Die Software stellt dem Benutzer dazu ein TOSCA Definitons Document (siehe Abschnitt 2.1.2) zur Verfügung, welches die verfügbaren Ressourcen und Relationen als TOSCA NodeTypes bzw. RelationshipTypes beinhaltet. Der Benutzer kann gewünschte Ressourcen/Relationen aus dem Definitions Document auswählen und anschließend mit der Acquire Operation Instanzen der Ressourcen/Relationen erzeugen.

4.2 Klassendiagramm des umgebenden Systems

In diesem Abschnitt werden die Schnittstellen (Interfaces) des Systems, in das die Software zur Integration von Informations-Ressourcen eingebunden werden soll, eingeführt und erläutert. Dazu zählen zum einen die Interfaces, welche direkt von Komponenten implementiert werden sollen und zum anderen Interfaces, die als Parameter oder Rückgabewert verwendet werden. Abb. 9 zeigt den relevanten Ausschnitt des Systems als Klassendiagramm.

Unterteilt werden kann das System in zwei unterschiedliche Komponenten, die implementiert werden sollen (siehe auch Abschnitt 4.1). Die eine Komponente ist der Domain Manager, der die verfügbaren Ressourcen und Relationen auflistet und Informationen für den Zugriff bereitstellt und die andere Komponente ist der Execution Environment Integrator, der die verfügbaren Operationen auf den Ressourcen bzw. Relationen ausführt.

Domain Manager

Die Operationen des Domain Managers werden durch das DomainManagerOperations Interface definiert. Die listDomain() Methode wird dazu verwendet, die verfügbaren Ressourcen und Relationen aufzulisten. Der Rückgabeparameter Definitions entspricht einem TOSCA Definitions Document (siehe Abschnitt 2.1.2), das alle verfügbaren Ressourcen und Relationen als NodeTypes bzw. RelationshipTypes beinhaltet. Mittels der Methode getDeployable() gibt ein Domain Manager für zwei QNames (= qualified names) und eine TIntention die nötigen Informationen für den Zugriff auf eine Ressource oder Relation mittels eines Deployable aus. Der erste QName definiert dabei den eindeutigen Namen der Ressource oder Relation, der zweite QName den Typ des Deployables und die TIntention das Ziel der Nutzung der Ressource bzw. Relation. Ein Deployable besteht aus einem InputStream, der alle Informationen für den Zugriff auf eine Ressource beinhaltet und einem Typ. Der InputStream wird mit der Methode getDeployable() geliefert und der Typ lässt sich mit der Methode getType() abrufen.

Die Methode listDeployablesOfResource() verwendet den QName einer Ressource als Parameter und wird dazu verwendet, alle Typen von Deployables für diese Ressource als Liste von QNames auszugeben. Alle zurückgelieferten QNames können anschließend, zusammen mit dem QName der Ressource, in die Methode getDeployable() eingesetzt werden. Mit der Methode getTargetNamespace() kann der Namespace, in dem der Domain Manager verwendet wird, als URI ausgegeben werden.

Die letzte Methode des Interfaces ist getImports(), welche eine Liste von Import Objekten zurückliefert. Ein Import Objekt stellt eine importierte Datei in einem TOSCA Definitions Document dar. Die Methoden liefern den Namespace, den ImportType bzw. die Datei als InputStream zurück. Die Import Liste, die ein Domain Manager zurückliefert, entspricht allen Importen des Definitions Document, das mit listDomain() ausgegeben wird. Diese Liste von Importen wird benötigt, um die Software unabhängig von externen Dateien zu machen und dem Nutzer alle Informationen, die zur Verwendung des Definitions Documents benötigt werden, direkt zur Verfügung zu stellen. Dabei können der Namespace und der ImportType zur Identifizierung des richtigen Objekts genutzt werden und der InputStream beinhaltet den tatsächlichen Import als XML Datei.

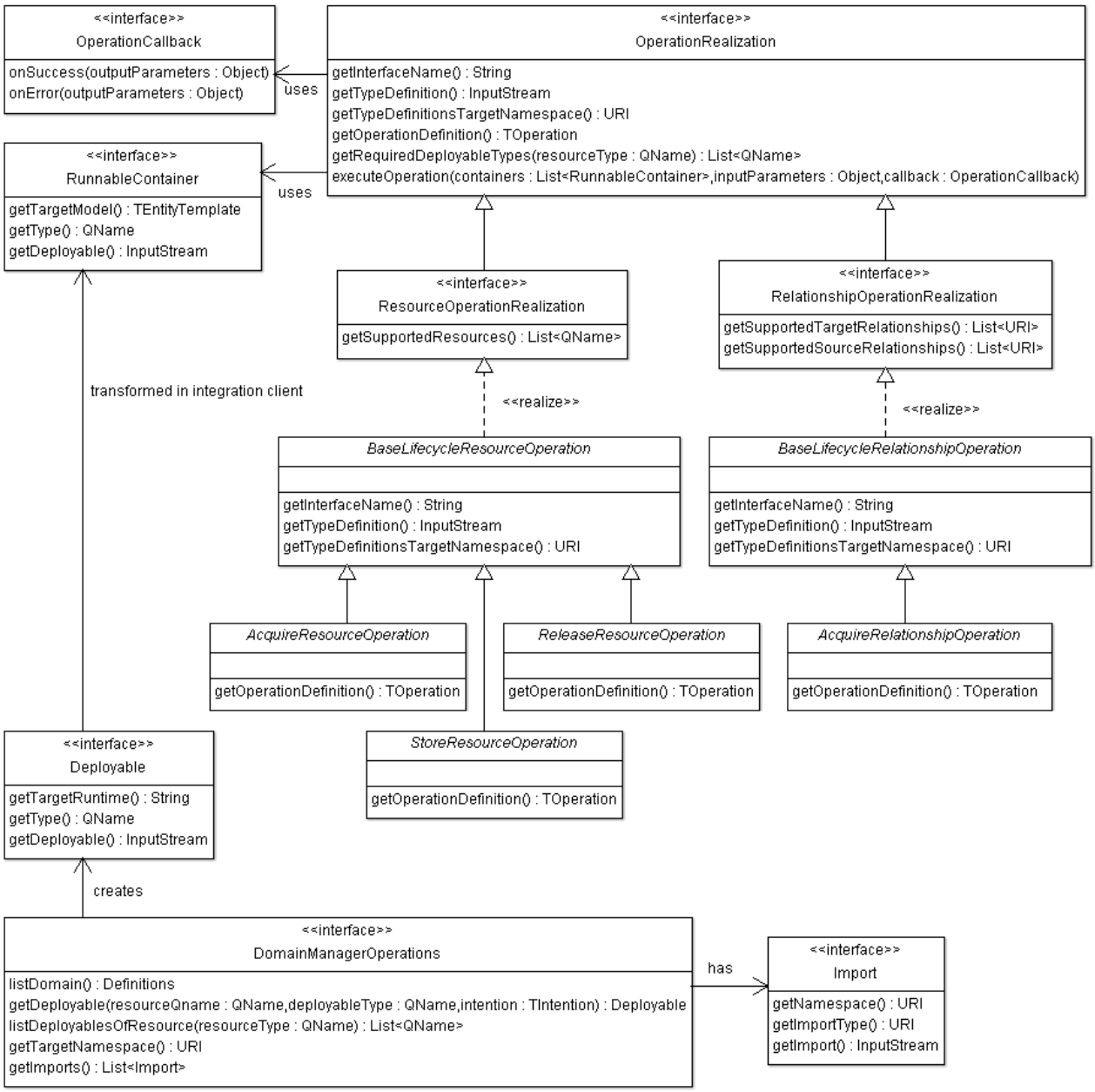

Abbildung 9: Klassendiagramm des Systems

Execution Environment Integrator

Ein Execution Environment Integrator besteht aus einer Menge von Operationen, die alle das Interface OperationRealization implementieren. Für die Software zur Integration von Informations-Ressourcen werden die vier Operationen AcquireResourceOperationen, StoreResourceOperation, ReleaseResourceOperation und AcquireRelationshipOperation verwendet. Ein Execution Environment Integrator kann jedoch um zusätzliche Operationen, wie zum Beispiel eine Ressource schreibgeschützt zu machen oder einen Account in einer Ressource zu erstellen, erweitert werden, wenn diese benötigt werden.

Das OperationRealization Interface definiert die grundlegenden Methoden einer Operation. Die ersten drei Methoden sind für die Erstellung der Software uninteressant, da diese bereits in den abstrakten Klassen BaseLifecycleResourceOperation bzw. BaseLifecycleRelationshipOperation implementiert sind und lediglich allgemeine Informationen über den Namespace bzw. das Interface liefern. Wichtig ist dagegen die Methode getOperationDefinition(), die eine TOperation zurückliefert. TOperation ist eine TOSCA Klasse und beinhaltet alle Informationen zur betreffenden Operation, wie zum Beispiel die Parameter oder Rückgabewerte. Implementiert wird diese Methode in den abstrakten Operations-Klassen (z.B. StoreResourceOperation).

Die Methode getRequiredDeployableTypes() liefert für den QName einer Ressource eine Liste von QNames an benötigten Deployables zurück. Diese müssen für die Ausführung der Operation bei einem Domain Manager angefordert, in RunnableContainer umgewandelt und anschließend als Parameter der executeOperation() Methode verwendet werden. Die letzte Methode des Interfaces ist executeOperation(). Sie wird verwendet um die Operation, welche durch die Klasse implementiert wird, auszuführen. Als Parameter werden der Methode eine Liste von RunnableContainers, ein Object und ein OperationCallback übergeben. Ein *RunnableContainer* beinhaltet, wie ein Deployable, alle Informationen, um auf eine Ressource zugreifen zu können. Zusätzlich beinhaltet der RunnableContainer aber auch ein TEntityTemplate. Das TEntityTemplate ist eine TOSCA Oberklasse, die zum Beispiel Node Template und Relationship Template als Unterklasse besitzt. Durch diesen Parameter wird bestimmt, welche Ressource durch die Operation bearbeitet werden soll und es können Eigenschaften mittels des Properties Elements gesetzt werden. Die Umwandlung von einem Deployable zu einem RunnableContainer findet innerhalb des Systems im sogenannten Integrations Client statt. Der zweite Parameter der executeOperation() Methode ist ein Object, das alle zusätzlichen Parameter der Operation beinhaltet. Diese zusätzlichen Parameter sind für die Einbindung von anderen Ressourcen Arten interessant. Für die Informations-Ressourcen werden sie hingegen nicht benötigt. Der dritte Parameter ist vom Typ OperationCallback. Eine Klasse, die das OperationCallback Interface implementiert, verfügt über die Methoden onSuccess() und onError(). Eine dieser beiden Methoden wird am Ende der Operation aufgerufen, um dem System damit die Rückgabe der Operation zu liefern.

Neben den Methoden des OperationRealization Interfaces implementiert eine Operation, je nachdem ob sie auf einer Ressource oder einer Relationship arbeitet, das ResourceOperationRealization Interface bzw. das RelationshipOperationRealization Interface. Die Methoden dieser Interfaces geben Listen aller in der Operation verwendbaren Ressourcen bzw. Relationen aus. Bei den Operationen auf Ressourcen ist dies nur eine Liste, wohingegen bei Operationen auf einer Relationship zwei Listen ausgegeben werden können, da eine Relationship immer zwischen zwei Ressourcen besteht.

4.3 Klassendiagramm der Software

In diesem Abschnitt werden die Klassen und Interfaces der Software zur Integration von Informations-Ressourcen als Klassendiagramm dargestellt. Außerdem wird für Teile der realisierten Methoden ein grober Ablauf beschrieben, um deren Funktionsweise zu verdeutlichen. Abb. 10 und Abb. 11 beinhalten das Klassendiagramm der Software. In Abschnitt 4.2 wurden bereits die Interfaces des Systems vorgestellt, die von der Software implementiert werden sollen. Die abstrakten Operations-Klassen (z.B. StoreResource-Operation) sind in diesem Abschnitt aufgrund der besseren Übersicht nur ohne Methoden und Vererbungshierarchie dargestellt, können aber in Abb. 9 nachgeschlagen werden.

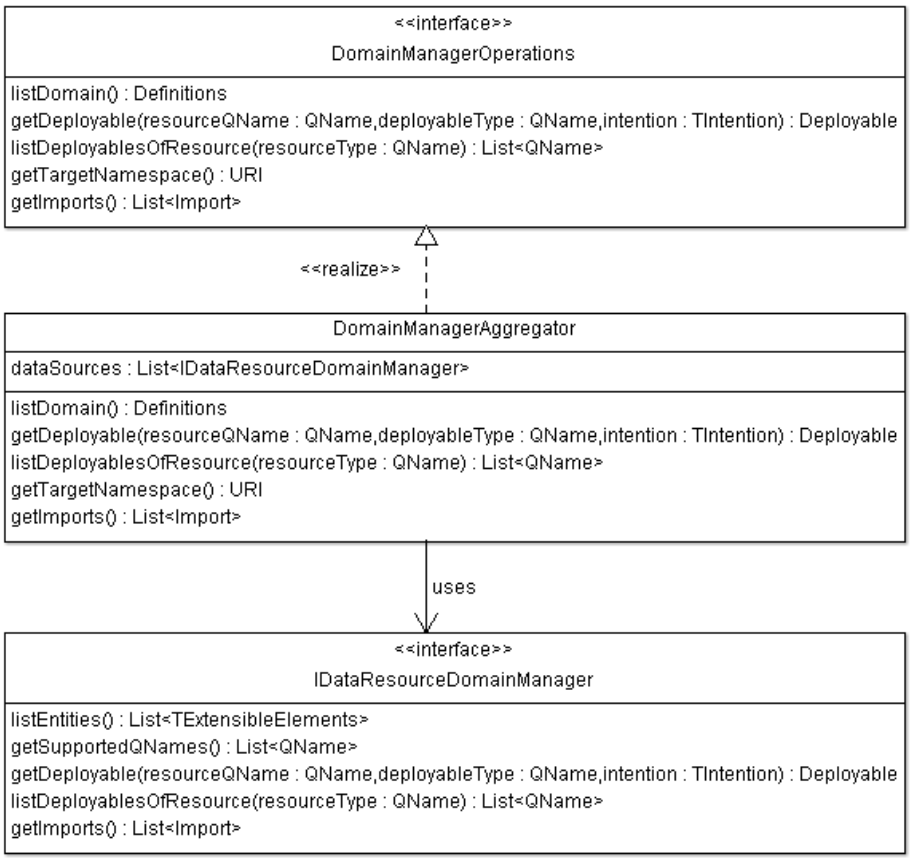

Abbildung 10: Klassendiagramm der Software (Teil 1)

Die Software zur Integration von Informations-Ressourcen beinhaltet zur Realisierung des DomainManagerOperations Interfaces die Klasse *DomainManagerAggregator*. Außerdem existiert für jede Operation eine Klasse, welche von der abstrakten Operation-Klasse erbt und die drei (bzw. bei Relationship-Operationen vier) noch nicht implementierten Methoden umsetzt. Zusätzlich wurden drei Interfaces (*IDataResourceDomainManager*, *IDataStorage*, *IDataLoader*) entworfen, die für eine Datenquelle implementiert werden müssen, wenn die Software um die Datenquelle erweitert werden soll.

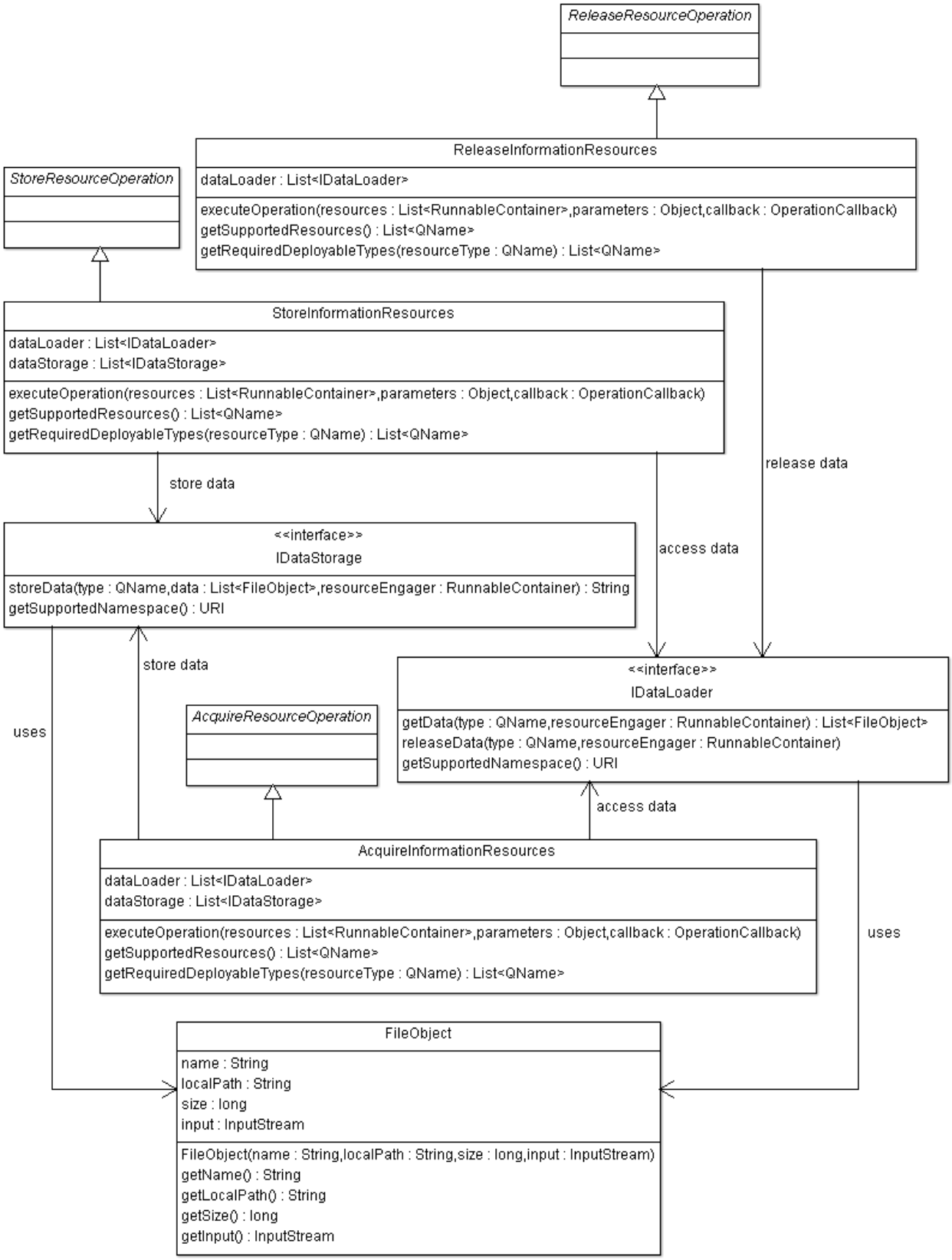

Abbildung 11: Klassendiagramm der Software (Teil 2)

Die Klassen DomainManagerAggregator, AcquireInformationResources, ReleaseInformationResources und StoreInformationResources wurden nach dem GoF-Pattern [23] der „*facade*“ entworfen. Unter einer „facade“ versteht man eine vereinfachte Schnittstelle zur Benutzung eines System. Sie ermöglicht es dem Benutzer, ausschließlich mit der Schnittstelle zu kommunizieren und dabei keine Informationen über die Abhängigkeiten und Klassen innerhalb des Systems zu benötigen. Die „facade“ leitet vom Benutzer erhaltene Anfragen an die Klassen und Schnittstellen weiter, die zum Bearbeiten benötigt werden. Durch dieses Vorgehen wird die lose Kopplung innerhalb des Systems gefordert. Damit wird vereinfacht, Komponenten auszutauschen bzw. neue Erweiterungen einzufügen. Insbesondere die Erweiterbarkeit ist für die Software zur Integration von Informations-Ressourcen von großer Bedeutung. Die „facade“-Komponenten nutzen Dependency Injection (siehe Abschnitt 2.3), um die verschiedenen Implementierungen der Interfaces im Konstruktor zu laden und in einer Liste zu speichern.

4.3.1 Interfaces zur Erweiterung der Software

Für jedes der drei entworfenen Interfaces wird im Folgenden beschrieben, was die einzelnen Methoden leisten müssen, um damit die Erweiterung der Software zu ermöglichen.

IDataResourceDomainManager

Eine Implementierung des IDataResourceDomainManager Interfaces wird für eine neue Datenquelle benötigt, um die beinhalteten Ressourcen und Relationen im DomainManager-Aggregator auflisten zu können. Außerdem muss die Implementierung für alle von ihr aufgelisteten Ressourcen und Relationen die Informationen liefern, die zur Ausführung von Operationen auf diesen Ressourcen bzw. Relationen benötigt werden. Für diese Funktionalitäten besitzt das Interface fünf verschiedene Methoden. Die getSupportedQNames() Methode liefert die QNames aller Ressourcen und Relationen, die von dieser Implementierung zur Zeit unterstützt werden. Diese Information wird vom DomainManagerAggregator verwendet, um die richtige Implementierung für eine bestimmte Ressource bzw. Relation zu finden. Dagegen liefert die listEntities() Methode die verfügbaren Ressourcen und Relationen als TOSCA NodeType bzw. RelationshipType, um dem DomainManagerAggregator die Erstellung des Definitions Documents zu ermöglichen. Die getImports() Methode liefert alle Imports, die von Ressourcen oder Relationen dieser Implementierung genutzt werden. Außerdem besitzt das Interface die Methode listDeployablesOfResource(), die für eine bestimmte Ressource oder Relation alle verfügbaren Deployables als Liste von QNames liefert. Die letzte Methode des Interfaces ist getDeployable() und diese wird dazu verwendet, das tatsächliche Deployable als Objekt zu liefern, welches die Informationen für den Zugriff auf die Ressource bzw. Relation beinhaltet. Diese Methode besitzt als Parameter zum einen den QName der Ressource bzw. Relation, für welche das Deployable aus-

gegeben werden soll und zum anderen einen QName, der von listDeployablesOfResource() ausgegeben wird und womit das gewünschte Deployable ausgewählt wird. Die gelieferten Informationen im Deployable müssen mit den benötigten Informationen der zugehörigen Implementierungen von IDataStorage und IDataLoader übereinstimmen.

IDataStorage

Das IDataStorage Interface wird verwendet um Ressourcen zu speichern. Das Interface besitzt zwei Methoden, die für eine neue Datenquelle implementiert werden müssen. Die erste Methode ist getSupportedNamespace(). Diese Methode liefert eine URI zurück, welche den Namespace der Ressourcen, die von der Implementierung unterstützt werden, darstellt. Die URI muss eindeutig unter allen Implementierungen sein und wird dazu verwendet, die richtige Implementierung für eine bestimmte Ressource zu finden. Die zweite Methode des Interfaces ist storeData(). Diese Methode führt die Speicherung der Ressource durch und hat drei Parameter. Der QName wird dazu verwendet, die Ressource eindeutig zu identifizieren. Der zweite Parameter ist eine Liste von *FileObjects*. Diese FileObjects werden sowohl von IDataStorage, als auch von IDataLoader verwendet und entsprechen dem Übertragungsformat innerhalb der Software. Jedes FileObject repräsentiert eine einzelne Datei und enthält alle benötigten Informationen über sie. Diese Informationen sind der Name, der lokale Pfad, die Größe der Datei und die Datei als InputStream. Der Name und der lokale Pfad werden dazu verwendet, interne Strukturen innerhalb einer Ressource zu erhalten. Der InputStream wird genutzt, um die Datei an einem neuen Ort erstellen zu können und die Größe der Datei wird für die Verwendung einiger Java Libraries benötigt. Der dritte Parameter ist ein RunnableContainer, der alle Informationen beinhaltet, um auf den gewünschten Speicherplatz der Ressource zugreifen und die Ressource dort erstellen zu können.

IDataLoader

Das IDataLoader Interface ist das Gegenstück zum IDataStorage Interface und ist für das Laden und Freigeben von Ressourcen zuständig. Es verfügt ebenfalls über die getSupportedNamespace() Methode, die den Namespace der zugehörigen Ressourcen als URI zurückliefert. Zusätzlich besitzt das Interface die Methoden releaseData() und getData(). Beide haben als Parameter einen QName, welcher die Ressource identifiziert, die für die Operation verwendet werden soll. Außerdem verwenden beide Methoden einen RunnableContainer als zweiten Parameter, der alle Informationen für den Zugriff auf die Ressource beinhaltet. Mit der releaseData() Methode wird die gewünschte Ressource gelöscht und kein Rückgabewert geliefert. Die getData() Methode hingegen lädt die Daten einer Ressource und gibt sie als Liste von FileObjects (siehe IDataStorage) zurück, die anschließend mit einer Implementierung von IDataStorage an einem anderen Platz gespeichert werden können.

4.3.2 Ablauf der wichtigsten Methoden

Im Folgenden soll der Ablauf der Methoden listDomain(), listDeployablesOfResource(), getDeployable(), getImports() des DomainManagerAggregators und für jede Operation die executeOperation() Methode beschrieben werden, da diese den interessanten zu implementierenden Methoden entsprechen. Neben den in Abb. 11 dargestellten Operationen Acquire, Store und Release für Ressourcen beinhaltet dies auch die AcquireAdminRelationshipOperation, die einen Benutzer zum Administrator einer MediaWiki Instanz macht. Diese Operation wurde im Klassendiagramm der Software nicht dargestellt, da sie von der AcquireRelationshipOperation Klasse des Systems (siehe Abschnitt 4.2) erbt und ansonsten keine Methoden oder Interfaces verwendet und keine weiteren Abhängigkeiten besitzt. Die Methoden der Klasse entsprechen also genau den Methoden der abstrakten Oberklasse.

Die weiteren Methoden, wie zum Beispiel getTargetNamespace() oder getSupportedResources(), werden hingegen nicht genauer betrachtet, da diese lediglich QNames von Ressourcen oder Namespaces zurückliefern und deren Ablauf vergleichsweise simpel ist. Zusätzliche Informationen zu den Methoden und die Implementierungen finden sich direkt im Git Projekt[11].

listDomain():

Das Ziel der listDomain() Methode des DomainManagerAggregators ist es, alle Ressourcen und Relationen, die von einem der IDataResourceDomainManager Implementierungen zur Verfügung gestellt werden, in einem einzelnen TOSCA Definitions Document (siehe Abschnitt 2.1.2) aufzulisten. Neben den NodeTypes bzw. RelationshipTypes, welche die Ressourcen und Relationen repräsentieren, soll das Definitions Document auch alle verwendeten Importe beinhalten. Der DomainManagerAggregator lädt im Konstruktor alle verfügbaren Implementierungen des IDataResourceDomainManager Interfaces und hat diese beim Aufruf von listDomain() als Liste vorliegen. Nach dem Aufruf der Methode kontaktiert der DomainManagerAggregator alle vorhandenen Implementierungen und ruft bei ihnen die Methode listEntities() auf. Die Implementierungen durchsuchen daraufhin ihre zugehörige Domäne und liefern die Ergebnisse als Liste von TExtensibleElements zurück. TExtensibleElements ist eine TOSCA Oberklasse, die unter anderem NodeTypes, RelationshipTypes und Imports als Unterklassen besitzt. Der DomainManagerAggregator empfängt alle Ergebnisse und fügt sie nach einer Überprüfung, ob keine unerwünschten Typen übergeben wurden, ins Definitions Document ein. Außerdem wird der Name, die ID und der Namespace des Definitions Document gesetzt. Am Ende der Methode wird das erzeugte Definitions Document zurückgegeben.

[11] `https://gitlab.com/timur87/integrating-data-resources`

listDeployablesOfResource():
Mit der listDeployablesOfResource() Methode sollen alle möglichen Deployables für eine bestimmte Ressource bzw. Relation als Liste von QNames ausgegeben werden. Mehrere Deployables für eine Ressource sind zum Beispiel sinnvoll, wenn mehrere Speicherorte verfügbar sind, an denen die Ressource dem Nutzer bereitgestellt werden kann. Damit kann der Nutzer entweder ein Deployable auswählen oder die Software eine zufällige Wahl treffen lassen. Der Vorteil der Auswahl aus mehreren Deployables ist, dass der Nutzer damit mehr Konfigurationsmöglichkeiten besitzt. Für eine Ordner Ressource im lokalen Dateisystem könnte zum Beispiel ein Deployable zur Verfügung stehen, das den Ordner in einem Git Repository liefert und ein Deployable, das Dropbox als Speicherort verwendet. Die Auswahl der Ressource bzw. Relation, für welche die Deployables gesucht werden sollen, wird durch einen QName als Parameter durchgeführt. Nach dem Aufruf der listDeployablesOfResource() Methode sucht der DomainManagerAggregator die Implementierung des IDataResourceDomainManager Interfaces, die für die Ressource bzw. Relation zuständig ist. Dazu wird über die Liste der verfügbaren Implementierungen iteriert und getestet, ob der QName der Ressource in der Rückgabe der Methode getSupportedQNames() vorhanden ist. Wenn die Implementierung gefunden wurde, wird die Anfrage an sie weitergegeben und ansonsten wird null zurückgegeben.

getDeployable():
Die getDeployable() Methode wird in der Software verwendet, um das Deployable einer bestimmten Ressource oder Relation zu erhalten. Die gewünschte Ressource bzw. Relation wird dabei durch den ersten QName „resourceQName“ identifiziert. Anhand dieses QNames ermittelt der DomainManagerAggregator die zuständige Implementierung des IDataResourceDomainManager Interfaces und leitet die Anfrage an diese mit beiden QNames und der Intention weiter. Wird keine passende Implementierung gefunden, liefert die Methode null zurück. Die Implementierung bestimmt zunächst die Informationen, die für den Zugriff auf die Ressource benötigt werden. Anschließend wird anhand des zweiten QNames „deployableType“ identifiziert, welches Deployable für die Ressource zur Verfügung gestellt werden soll. Der QName muss dabei in der von listDeployablesOfResource() gelieferten Liste sein. Je nach Art der Speicherung, welche das gewählte Deployable repräsentiert, sammelt die Implementierung die Informationen, die für die Speicherung der Ressource benötigt werden. Anschließend werden die Informationen zum Laden und Speichern der Ressource in einen InputStream eingefügt und damit das benötigt Deployable erzeugt. Das Deployable wird an den DomainManagerAggregator zurückgegeben und von diesem als Rückgabe der Methode verwendet.

getImports():
Mit der getImports() Methode wird es ermöglicht, dem Benutzer der Software alle Imports im durch listDomain() gelieferten Definitions Document direkt zur Verfügung zu stellen. Damit wird das Definitions Document von Abhängigkeiten nach außen bereinigt. Die zurückgegebene Liste der Methode bezieht sich immer auf den letzten Aufruf der listDomain() Methode. Der DomainManagerAggregator fragt beim Aufruf der getImports() Methode bei allen Implementierungen des IDataResourceDomainManager Interfaces die aktuellen Imports ab. Innerhalb der Implementierungen wird die Liste bei jedem Aufruf von listEntities() aktualisiert. Nach dem Erhalt aller Import Listen der Implementierungen, werden diese auf mehrfache Vorkommen eines Imports überprüft und vereinigt.

executeOperation() AcquireInformationResources:
Die executeOperation() Methode besitzt für alle Operationen drei Parameter. Der erste Parameter ist eine Liste von RunnableContainer Objekten. In diesen Objekten befinden sich alle Informationen über die Ressource, die zur Ausführung der Operation benötigt werden. Je nach Operation kann die Liste nur einen oder mehrere RunnableContainer beinhalten. Die Anzahl und der Typ der benötigten RunnableContainer wird durch die Methode getRequiredDeployableTypes() der Operation definiert. Zur Verwendung der Operation müssen diese RunnableContainer von einem Domain Manager geladen und anschließend an die Methode übergeben werden. Der zweite Parameter ist ein Object, welches für weitere Parameter verwendet werden kann, aber für den Kontext der Informations-Ressourcen nicht benötigt wird. Der letzte Parameter ist ein OperationCallback Objekt (siehe Abschnitt 4.2). Dieses Objekt bekommt die Rückgabe der Operation übergeben und behandelt diese mit den Methoden onSuccess() bzw. onError().

Nach dem Aufruf der Methode wird zunächst der QName der Ressource aus den RunnableContainer Objekten ausgelesen. Anhand dieses QNames kann bestimmt werden, welche Implementierungen des IDataLoader und IDataStorage Interfaces für die Ressource verwendet werden müssen. Die Implementierungen des IDataLoader bzw. IDataStorage Interfaces werden dabei von der Operations Klasse im Konstruktor geladen. Nachdem die Implementierung des IDataLoader Interfaces bestimmt wurde, wird die getData() Methode von dieser aufgerufen. Die Rückgabe der Methode ist eine Liste von FileObjects, die den Inhalt der Ressource repräsentieren. Anschließend wird mit dieser Liste die storeData() Methode der IDataStorage Implementierung aufgerufen und die Ressource damit an einer gewünschten Position bereitgestellt. Durch das Node Template, das im RunnableContainer beinhaltet ist, können mittels Properties zusätzliche Konfigurationsparameter für die Bereitstellung übermittelt werden. Dazu kann ein XML Element in das Node Template eingefügt werden, das alle Attribute enthält, die im zugehörigen NodeType

definiert wurden (siehe Abschnitt 2.1.3). Die storeData() Methode liefert am Ende der Ausführungen alle interessanten Informationen über die bereitgestellte Instanz zurück, die dann an das OperationCallback Objekt übergeben werden. Die Informationen beinhalten für jede Ressource die URI, über welche auf die Instanz zugegriffen werden kann. Außerdem können noch zusätzliche Informationen, wie zum Beispiel Benutzername und Passwort, ausgegeben werden, falls diese für den Zugriff benötigt werden.

executeOperation() StoreInformationResources:
Die executeOperation() Methode der StoreInformationResources Operation läuft fast gleich ab, wie die Methode der AcquireInformationResources Operation. Die Unterschiede zwischen den Operationen sind jedoch zum einen die Bedeutung innerhalb eines informellen Prozesses und zum anderen die unterschiedlichen Ressourcen, auf welche die Operationen angewendet werden können. Die Acquire Operation wird üblicherweise am Anfang eines informellen Prozesses ausgeführt, um die Ressourcen für den Prozess zu initialisieren. Die Store Operation hingegen wird verwendet, um eine während des Prozesses veränderte Ressource am Ende zu speichern und damit für nachfolgende Prozesse zu erhalten. Die Store Operation ist außerdem nur optional innerhalb eines informellen Prozesses. Die Bedeutung der Operationen spiegelt sich auch in den verwendeten Ressourcen wieder. Dies lässt sich am Beispiel einer MediaWiki Ressource verdeutlichen, die mit Docker (siehe Abschnitt 2.2) integriert wird. Die Acquire Operation lässt sich lediglich auf Docker Images anwenden, die passive Komponenten darstellen und erzeugt daraus eine aktive Instanz für den Prozess. Die Store Operation hingegen lässt sich nur auf aktive Container anwenden, welche durch die Operation als passives Image gespeichert werden und damit jederzeit wieder in neuen Prozessen erstellt werden können.

executeOperation() ReleaseInformationResources:
Mit der executeOperation() Methode der ReleaseInformationResources Operation können mit Acquire erstellte Ressourcen freigegeben werden. Dafür verwendet die Operation Implementierungen des IDataLoader Interfaces. Für die übergebenen RunnableContainer wird auf diesen Implementierungen die Methode releaseData() aufgerufen. Nach dem Abschluss der Operation wird dem OperationCallback Objekt durch den Aufruf der richtigen Methode mitgeteilt, ob die Freigabe der Ressourcen erfolgreich war oder ein Fehler aufgetreten ist.

executeOperation() AcquireAdminRelationshipOperation:
Diese Methode wird verwendet, um einen Benutzer zum Administrator einer MediaWiki Instanz zu machen. Dazu benötigt die Methode einen RunnableContainer in den Parametern, der die Informationen beinhaltet, um auf die MediaWiki Instanz zugreifen zu können. Außerdem muss das Relationship Template im RunnableContainer unter den Properties

ein XML Element mit den Attributen „userName“ und „password“ besitzen. Die unter diesen Attributen angegebenen Werte werden später als Benutzername und Passwort des Accounts in der MediaWiki Instanz verwendet. Um die Relation zu erzeugen, liest die Methode zunächst die Parameter aus. Anschließend führt sie mit den Parametern ein PHP Skript auf der MediaWiki Instanz aus, das den benötigten Account erzeugt. Abschließend wird dem OperationCallback Objekt die Information über die Location der MediaWiki Instanz und der Benutzername und das Passwort des erzeugten Accounts übergeben.

5 Implementierung der Software zur Integration von Informations-Ressourcen

In diesem Kapitel werden Details zur Implementierung und Nutzung der in Abschnitt 4 entworfenen Software zur Integration von Informations-Ressourcen dargestellt. Dazu werden zunächst in Abschnitt 5.1 die zur Implementierung genutzten Technologien und Libraries beschrieben. Anschließend wird in Abschnitt 5.2 zusammengefasst, über welche Parameter die Software an die Bedürfnisse des Nutzers bzw. die zur Verfügung stehenden Datenquellen angepasst werden kann. In Abschnitt 5.3 wird die konkrete Verwendung der Software anhand eines Beispiels erklärt. Dazu werden Code Ausschnitte und Rückgaben der Software zur besseren Verständlichkeit verwendet.

5.1 Verwendete Technologien und Libraries

Für die Implementierung der Software zur Integration von Informations-Ressourcen wurden viele verschiedene Technologien und Libraries verwendet, da der Zugriff auf jede Datenquellen auf eine andere Weise durchgeführt werden muss. Zudem wurden für die Implementierung der allgemeinen Logik der Software einige Technologien und Libraries benötigt. In diesem Abschnitt sollen die wichtigsten von ihnen eingeführt und ihre Anwendung in der Software erklärt werden.

Neben den hier aufgeführten Technologien und Libraries ist die objektorientierte Programmiersprache Java, in der die Umsetzung der Software erfolgte, von Bedeutung. Auf diese wird wegen der großen Bekanntheit jedoch nicht genauer eingegangen. Docker und Spring, als Dependency Injection Library, sind ebenfalls wichtige Technologien für die Implementierung. Diese wurden bereits in Abschnitt 2 eingeführt. Docker wird innerhalb der Software zur Integration von MediaWiki und von MySql Datenbanken verwendet. Spring hingegen wird in der Software genutzt, um eine möglichst lose Kopplung der Komponenten zu erreichen und alle Implementierungen von Interfaces zur Laufzeit zu laden.

EGit und JGit:
EGit[12] und *JGit*[13] sind zwei Java Libraries zur Verwendung von Git. Mit der JGit Library ist es möglich, Lese- und Schreiboperationen auf einem Git Repository auszuführen. Dazu zählen zum Beispiel die Operationen „pull“, „add“, „commit“ und „push“. Die Library wurde in der Software verwendet, um Ordner in Git Repositories als Ressourcen zu integrieren. Die EGit Library hingegen ermöglicht den Zugriff auf die GitHub API[14]. Über diese API lassen sich Repositories löschen und neu erstellen. Mit EGit und der Möglichkeit die Daten eines Repository mittels JGit zu lesen und schreiben, konnten GitHub Repositories als Datenquelle eingefügt werden.

[12] `http://www.eclipse.org/egit/`
[13] `http://www.eclipse.org/jgit/`
[14] `https://developer.github.com/v3/`

Dropbox Core SDK:
Das *Dropbox Core SDK* ist eine Library, die es ermöglicht, aus Java Anwendungen auf die Dropbox Core API[15] zugreifen zu können. Die Dropbox Core API bietet unter anderem die Funktionen, alle Ordner innerhalb eines Dropbox Accounts aufzulisten und Dateien oder Ordner hochzuladen, zu löschen oder zu downloaden. Mittels des Dropbox Core SDK wurden Ordner in Dropbox Accounts in die Software integriert.

JDBC:
Java Database Connectivity (JDBC) ist eine Datenbankschnittstelle für die Programmiersprache Java. Über diese Schnittstelle ist es möglich, eine Verbindung zu verschiedenen Arten von Datenbanken herzustellen. Zudem können SQL-Anfragen an die Datenbank gestellt werden und damit Tabellen eingefügt, geupdated oder gelöscht werden. Die Erstellung neuer Datenbanken ist mit der Schnittstelle jedoch nicht möglich. Mit ihr kann nur auf bereits existierende Datenbanken zugegriffen werden. In der Software wurde die Schnittstelle verwendet, um Verbindungen zu MySql Datenbanken zu ermöglichen und Daten aus Tabellen zu lesen bzw. in Tabellen einzufügen. Damit ist es möglich, Tabellen in MySql Datenbanken als Ressourcen in der Software zu verwenden. JDBC kann für die Nutzung anderer Datenbanken als Ressourcen ebenfalls verwendet werden, wenn die Software um diese erweitert werden soll. Dafür muss lediglich der zur Datenbank passende Treiber an die JDBC Komponenten übergeben werden.

Docker-Java und Docker-Client:
Docker-Java[16] und *Docker-Client*[17] sind zwei Libraries, die den Zugriff auf einen Docker Daemon (siehe Abschnitt 2.2) über die Docker Remote API ermöglichen. Damit kann eine Java Anwendung als Docker Client fungieren und Befehle, wie zum Beispiel das Erstellen eines Containers, auf dem Docker Daemon ausführen. Die zwei unterschiedlichen Libraries werden benötigt, da beide aktuell nur einen Teil der Funktionalität von der Docker Remote API abdecken. Mit Docker-Client ist zum Beispiel die Erstellung eines Images aus einem laufenden Container nicht möglich. Diese Funktionalität wird jedoch zum Speichern des Zustandes einer Ressource, die mit Docker initialisiert wurde, benötigt. Docker-Java eignet sich dagegen nicht gut um einen Befehl in einem laufenden Container auszuführen. Für die Initialisierung einer MediaWiki Instanz wird dies benötigt, da dort ein PHP Skript im Container ausgeführt werden muss. Die Kombination beider Libraries bietet alle Docker Funktionen, die für die Software benötigt werden.

[15] `https://www.dropbox.com/developers-v1/core`
[16] `https://github.com/docker-java/docker-java`
[17] `https://github.com/spotify/docker-client`

Apache Commons Configuration:
Die *Apache Commons Configuration Library*[18] wurde entworfen, um Konfigurationsparameter aus verschiedenen Quellen zur Laufzeit zu laden. Durch die Nutzung der Library müssen die Konfigurationsparameter nicht über den Konstruktor eines Objekts eingefügt werden, sondern können dynamisch geladen werden. In der Software wurde eine Properties Datei als Quelle der Konfigurationsparameter verwendet. Die Verwendung der Properties Datei und welche Parameter dort gesetzt werden können, wird im nächsten Abschnitt erklärt. Innerhalb der Software wurde diese Library verwendet, da damit die Konfiguration der Software auch durch Personen möglich ist, die keine Kenntnisse über die Implementierung der Software besitzen und dies die Anwendungsmöglichkeiten der Software erweitert.

[18] `https://commons.apache.org/proper/commons-configuration/`

5.2 Konfigurationsparameter der Software

Um die Komponenten der Software einfach verwenden zu können und keine Informationen im Konstruktor übergeben zu müssen, wird in der Software eine Properties Datei genutzt, welche die nötigen Konfigurationsparameter beinhaltet. Damit ist es möglich, dass alle Komponenten einen Konstruktor ohne Argumente besitzen. Zum Laden der Konfigurationsdaten, muss eine Datei mit dem Namen „data.properties“ im Projektordner existieren. In diesem Abschnitt wird anhand des Beispiels in Listing 2 gezeigt, welche Parameter in dieser Datei eingesetzt werden können und welche Auswirkung dies auf die Software hat.

```
1  # Meta data of the definitions document
2  definitions.name = InformationResources
3  definitions.namespace = http://www.uni-stuttgart.de/
4
5  # Meta data about the locations of implementations
6  IDataResourceDomainManager.locations =
       uni_stuttgart.integration.domain_manager.implementations
7  IDataLoader.locations = uni_stuttgart.integration.eei.implementations
8  IDataStorage.locations = uni_stuttgart.integration.eei.implementations
9
10 # Meta data about Git
11 GitDataResourceDomainManager.githubUsernameList = accountName
12 GitDataResourceDomainManager.githubPasswordList = testpass
13 GitDataResourceDomainManager.urlList = https://gitlab.com/user/repo
14 GitDataResourceDomainManager.repositoryUsernameList = testUser
15 GitDataResourceDomainManager.repositoryPasswordList = testpass
16
17 # Meta data about local file resources
18 FileDataResourceDomainManager.sourcePaths = C:\Users\User1\Data
19
20 # Meta data about docker clients
21 Docker.uri = tcp://192.168.99.100:2376
22 Docker.certs = /Users/User1/.docker/machine/certs/
23
24 # Meta data about dropbox accounts
25 Dropbox.accessToken = qXPJQq4y8JAAAAAAAAAACwNE3f7Z8Q9lT686lVFmQ9cuQT6
26 Dropbox.username = account@web.de
27 Dropbox.password = testpass
```

Listing 2: „data.properties“ Datei

Die ersten beiden Eigenschaften betreffen das Definitions Document, das vom DomainManagerAggregator dem Nutzer zur Verfügung gestellt wird. Der hier gesetzte Name und Namespace werden als Name bzw. Namespace des Definitions Documents verwendet.

Die nächsten drei Eigenschaften (Zeile 6-8) werden für die Dependency Injection mittels Spring genutzt. Damit Spring die Implementierungen der Interfaces laden kann, benötigt es die Java Package Namen, in denen die Implementierungen liegen. Die drei Eigenschaften werden dabei alle als kommaseparierte Liste verwendet. Das bedeutet, es können mehrere Package Namen angegeben werden, die dann alle von Spring nach Implementierungen durchsucht werden. Mit der konkreten Properties Datei in Listing 2 werden zum Beispiel alle Implementierungen des IDataLoader Interfaces aus dem Package „uni_stuttgart.integration.eei.implementations“ geladen.

Alle weiteren Eigenschaften beziehen sich auf die Verwendung bestimmter Ressourcen Arten. Für jede Ressourcen Art werden sogenannte „Quellinformationen“ angegeben. Die Quellinformationen bestimmen, von welchem Punkt aus die Software nach Ressourcen suchen soll. Sollte eine Datenquelle zum Beispiel mit einem Passwort geschützt sein, muss dieses hier ebenfalls angegeben werden. Der Domain Manager nutzt diese Informationen um ein Deployable für eine Ressource zu erzeugen und dieses muss alle Informationen beinhalten um auf die Ressource zugreifen zu können.

Git als Datenquelle unterteilt sich für die Nutzung in der Software zum einen in Ordner und zum anderen in Repositories als Ressourcen. Für Git Repositories müssen die beiden Eigenschaften „githubUsernameList“ und „githubPasswordList“ angegeben werden. Die Listen können eine beliebige Anzahl an GitHub Accounts und zugehörige Passwörter beinhalten. Die Software listet damit alle Repositories als Ressourcen, die innerhalb dieser Accounts existieren. Für Ordner in Git Repositories gibt es zum einen dieselbe Möglichkeit, durch die alle Ordner in allen Repositories eines Accounts aufgelistet werden. Zusätzlich ist es noch möglich, einzelne Repositories anzugeben, von denen dann alle beinhalteten Ordner als Ressourcen verwendet werden. Hierfür können die Eigenschaften in den Zeilen 13-15 verwendet werden, in denen die URL des Repositories der Username und das Passwort angegeben werden müssen.

Für Ordner im lokalen Dateisystem wird lediglich ein Ordner benötigt, von welchem alle beinhalteten Ordner aufgelistet werden sollen (Zeile 18). Dabei kann eine beliebige Anzahl an Ordnern angegeben werden. Im Beispiel werden also alle Ordner, die sich innerhalb des Ordners „C:\Users\User1\Data“ befinden, vom Domain Manager als Ressourcen aufgelistet.

Docker wird in der Software zur Integration von Informations-Ressourcen verwendet, um MediaWiki und MySql Instanzen zu erzeugen und zu speichern. Es können zum einen laufende MediaWiki bzw. MySql Container verwendet werden und damit neue Images erzeugt werden. Außerdem können gespeicherte Images verwendet werden, um damit neue Container zu erstellen. Dabei existiert für MediaWiki und MySql ein Image für eine leere Instanz, das aus der Docker Registry heruntergeladen und in jedem Client verwendet

werden kann. Alle anderen Images werden lokal in den zu den Docker Clients gehörenden Docker Daemons gespeichert. Die laufenden Container sind ebenfalls von dem Docker Daemon abhängig, in dem sie erstellt wurden. Deshalb sind die vorhandenen Docker Clients für die Software wichtig, um aktive und passiv gespeicherte Instanzen auflisten und verwalten zu können (siehe Abschnitt 2.2). Als Informationen, um auf einen Docker Client zugreifen zu können, benötigt die Software die URI und den Pfad zu den Zertifikaten des Client. Die URI besteht aus einem Protokoll, der IP-Adresse und dem Port des Clients. Der Pfad zu den Zertifikaten eines Clients wird benötigt, da nur damit der Zugriff auf ihn erlaubt ist.

Für die Verwendung der Ordner eines Dropbox Accounts als Ressourcen, müssen drei Eigenschaften in der Properties Datei gesetzt werden (Zeile 25-27). Die erste Information, die benötigt wird, ist ein sogenanntes Access Token. Das Access Token kann über die Dropbox Website[19] für einen Dropbox Account erstellt werden. Die beiden anderen benötigten Informationen sind der Benutzername und das Passwort des Accounts. Auf diese Weise kann eine beliebige Anzahl an Dropbox Accounts zur Verwendung in der Software hinzugefügt werden.

[19] `https://www.dropbox.com/`

5.3 Fallstudie

In diesem Abschnitt wird die mögliche Verwendung der Software am Beispiel der Bereitstellung einer Ressource in einem informellen Prozess illustriert. Das Beispiel umfasst den kompletten Ablauf vom Auswählen einer Ressource bis zur Durchführung einer Operation. Als Ressource wurde eine MediaWiki Instanz ausgewählt. Die Verwendung aller anderen Ressourcen ist jedoch auf dieselbe Art möglich. Dabei unterscheiden sich lediglich die möglichen Attribute im Node Template und die Informationen zur Bereitstellung der Ressource, die von der Software am Ende geliefert werden. Genutzt wird außerdem die AcquireInformationResources Operation, diese kann aber leicht durch jede andere Operation ersetzt werden. Neben den Code Ausschnitten, die zu den einzelnen Schritten gehören, werden auch die Rückgaben der Software beispielhaft angegeben, um zu verdeutlichen wie diese aussehen können.

Im ersten Schritt muss zunächst ein DomainManagerAggregator Objekt erzeugt werden. Anschließend wird mit der Methode listDomain() das aktuelle Definitions Document abgerufen, das alle verfügbaren Ressourcen und Relationen beinhaltet und dem Nutzer damit eine Auswahl anbietet. Der zugehörige Code wurde in Listing 3 abgebildet.

```
1 DomainManagerAggregator dm = new DomainManagerAggregator();
2 Definitions definitions = dm.listDomain();
```

Listing 3: Fallstudie: Code Teil 1

Als Rückgabe liefert die Software dem Benutzer das Definitions Document in Listing 4. Das Definitions Document verfügt über die Attribute id, name und targetNamespace. Die Werte für name und targetNamespace werden aus der data.properties Datei (siehe Abschnitt 5.2) geladen. Die id wird dafür verwendet, um das Definitions Document klar von früheren Versionen abzugrenzen. Deswegen wird in dieses Attribut neben dem Namen die aktuelle Zeit bei der Erstellung des Documents angegeben. Außerdem beinhaltet das Definitions Document insgesamt vier Elemente, wobei zwei vom Typ Import und zwei vom Typ NodeType sind.

Die beiden Import Elemente werden genutzt, um zwei externe Dateien in das Definitions Document einzubinden. Da für beide Elemente unter dem Attribut „importType“ der Wert „http://www.w3.org/2001/XMLSchema“ angegeben wurde, repräsentieren sie XML Schema Dateien. Der Namespace der Importe stimmt mit dem Namespace der Dateien überein, die mittels getImports() von einem Domain Manager geliefert werden. Diese können genutzt werden, um die richtige Datei zu identifizieren. Deshalb benötigen die Importe kein „location“ Attribut, sondern können direkt vom Domain Manager ange-

fordert werden. Die zwei NodeTypes innerhalb des Documents zeigen dem Nutzer, dass derzeit zwei verschiedene Ressourcen zur Initialisierung in einem informellen Prozess zur Verfügung stehen. Die Eigenschaften „name“ und „targetNamespace“ sind für jede Ressource eindeutig. Außerdem besitzen die NodeTypes ein Element PropertiesDefinition. In diesem Element wird unter der Eigenschaft „element“ ein XML Element definiert, das die Eigenschaften des NodeTypes definiert. Im vorliegenden Definitions Document sind die XML Elemente in den importierten XML Schema Dateien beinhaltet und können über diese zugegriffen werden.

```
1  <Definitions id="InformationResources - 2016/04/15 16:32:50"
2    name="InformationResources"
3    targetNamespace="http://www.uni-stuttgart.de/"
4    xmlns:mysql="http://www.uni-stuttgart.de/resources/data-resources/xsd/
         mysql"
5    xmlns:media="http://www.uni-stuttgart.de/resources/knowledge-resources/
         xsd/mediawiki">
6
7    <Import importType="http://www.w3.org/2001/XMLSchema"
8      namespace="http://www.uni-stuttgart.de/resources/data-resources/xsd/
           mysql">
9
10   <Import importType="http://www.w3.org/2001/XMLSchema"
11     namespace="http://www.uni-stuttgart.de/resources/knowledge-resources/
           xsd/mediawiki">
12
13   <NodeType name="newMySql"
14     targetNamespace="http://www.uni-stuttgart.de/resources/data-resources/
           mysql" />
15     <PropertiesDefinition element="mysql:MySqlProperties" />
16   </NodeType>
17
18   <NodeType name="newMediawiki"
19     targetNamespace="http://www.uni-stuttgart.de/resources/knowledge-
           resources/mediawiki" />
20     <PropertiesDefinition element="media:MediawikiProperties" />
21   </NodeType>
22
23 </Definitions>
```

Listing 4: Fallstudie: Definitions Document

Der Nutzer entscheidet sich im Beispiel dafür, eine MediaWiki Instanz mit der Acquire-InformationResources Operation zu erzeugen. Dazu müssen im nächsten Schritt die Eigenschaften des „newMediawiki“ NodeTypes ausgelesen werden. Der dazu notwendige Code ist in Listing 5 abgebildet.

```
1  TEntityType resource = (TEntityType) definitions.
       getServiceTemplateOrNodeTypeOrNodeTypeImplementation().get(1);
2   InputStream xsdFile = null;
3
4  // read related namespace
5  String xsdNamespace = resource.getPropertiesDefinition().getElement().
       getNamespaceURI();
6  for(Import importFile : dm.getImports()){
7    // search import with same namespace and use InputStream
8    if(importFile.getNamespace().toString().equals(xsdNamespace)){
9      xsdFile = importFile.getImport();
10     break;
11   }
12 }
```

Listing 5: Fallstudie: Code Teil 2

Zunächst wird der TEntityType aus dem Definitions Document ausgegeben. Der Index „1“ der get() Methode steht für die Position des „newMediawiki“ NodeTypes im Document. Für den „newMySql“ NodeType müsste an dieser Stelle der Index „0“ verwendet werden. Anschließend wird der Namespace des Elements in der PropertiesDefinition des NodeTypes angefordert. Dieser Namespace kann genutzt werden, um das richtige Import Objekt des DomainManagerAggregators zu identifizieren. Dazu wird über alle Imports iteriert, die mittels der getImports() Methode geliefert werden. Sobald das Import Objekt gefunden wird, das denselben Namespace besitzt, wird der InputStream in einer Variable gespeichert und die Suche beendet.

Der vorliegende InputStream kann zum Beispiel dafür genutzt werden, die XML Schema Datei dem Nutzer auf der Konsole auszugeben. Eine XML Schema Datei für den „newMediawiki“ NodeType ist in Listing 6 dargestellt. Die Schema Datei enthält das Element „MediawikiProperties“, auf das vom NodeType im Definitions Document verwiesen wird. Das Element beinhaltet drei Attribute, die für ein NodeTemplate des Types gesetzt werden können. Dabei entspricht „mediawikiName“ dem Namen der MediaWiki Instanz, die durch das NodeTemplate erzeugt werden soll. Wenn das Attribut nicht gesetzt wird, nutzt die Software einen zufälligen Namen. Die Attribute „mediawikiAdmin“ und „mediawikiAdminPass“ können dagegen dafür verwendet werden, einen initialen Administrator Account für die MediaWiki Instanz zu erstellen.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<x:schema xmlns:x="http://www.w3.org/2001/XMLSchema" targetNamespace="http
    ://www.uni-stuttgart.de/resources/knowledge-resources/xsd/mediawiki">
  <x:element name="MediawikiProperties">
    <x:complexType>
      <x:attribute name="mediawikiName" type="xs:string"/>
      <x:attribute name="mediawikiAdmin" type="xs:string"/>
      <x:attribute name="mediawikiAdminPass" type="xs:string"/>
    </x:complexType>
  </x:element>
</x:schema>
```

Listing 6: Fallstudie: XML Schema Datei

Nachdem dem Nutzer durch das XML Schema angezeigt wurde, welche Einstellungsmöglichkeiten er zusätzlich zum RunnableContainer für die Ressource besitzt, kann die Operation als nächstes ausgeführt werden. Der Code findet sich in Listing 7. Für die Anforderung der Deployables vom DomainManagerAggregator, muss zunächst der QName der Ressource aus dem NodeType ausgelesen werden. Danach kann das NodeTemplate für die RunnableContainer erzeugt werden und mit dem Properties Element versehen werden. Das XML Element wird mittels DOM[20] erstellt. Dazu wird ein Document und Element erstellt und anschließend werden die drei Attribute aus der XML Schema Datei im Element gesetzt. Die angegebenen Attribute sorgen dafür, dass die erzeugte Instanz den Namen „InformalProcessWiki“ trägt und ein Administrator Account erstellt wird, der mit dem Benutzername „admin“ und dem Password „password“ versehen wird. Im nächsten Schritt wird ein Objekt der Operation, in diesem Fall AcquireInformationResources, erstellt. Von der Operation kann für den QName der Ressource abgefragt werden, wie viele und welche Deployables benötigt werden. Jedes geforderte Deployable wird vom DomainManagaerAggregator abgerufen und zusammen mit dem erstellten Template zu einem RunnableContainer umgewandelt. Alle erstellten RunnableContainer werden in eine Liste eingefügt. Mit dieser Liste kann schließlich die Operation mittels der executeOperation() Methode gestartet werden.

[20] https://docs.oracle.com/javase/tutorial/jaxp/dom/index.html

```
1  QName resourceName = new QName(resource.getTargetNamespace(), resource.
       getName());
2  TNodeTemplate template = new TNodeTemplate();
3
4  // add attributes to the element
5  DOMImplementation impl = DOMImplementationImpl.getDOMImplementation();
6  Document doc = impl.createDocument(null, "testDoc", null);
7  Element element = doc.createElement("Element");
8  element.setAttribute("mediawikiName", "InformalProcessWiki");
9  element.setAttribute("mediawikiAdmin", "admin");
10 element.setAttribute("mediawikiAdminPass", "password");
11
12 // add Properties to template
13 Properties prop = new Properties();
14 prop.setAny(element);
15 template.setProperties(prop);
16
17 // create RunnableContainers and execute the operation
18 List<RunnableContainer> list = new ArrayList<RunnableContainer>();
19 AcquireInformationResources operation = new AcquireInformationResources();
20 for(QName depl : operation.getRequiredDeployableTypes(resourceName)){
21   Deployable deployable = dm.getDeployable(resourceName, depl, new
         TIntention());
22   list.add(new RunnableContainerObject(template, deployable.getDeployable()
         , deployable.getType()));
23 }
24 operation.executeOperation(list, null, new OperationCallbackObject());
```

Listing 7: Fallstudie: Code Teil 3

Zusätzlich zur Liste mit RunnableContainer Objekten, muss der executeOperation() Methode ein Objekt übergeben werden, welches das OperationCallback Interface implementiert. Nachdem die Operation beendet ist, wird auf diesem Objekt die onSuccess() bzw. onError() Methode mit den Rückgabewerten aufgerufen. Die aufgerufene Methode ist dann dafür zuständig, die Ergebnisse zu verarbeiten. Für die Fallstudie wurde eine Implementierung des Interfaces erzeugt, die lediglich die Rückgabe auf der Konsole ausgibt. Die Ausgabe findet sich in Listing 8. Sie beinhaltet den Status der Operation („Success“). Außerdem wird die Location ausgegeben, welche einer URI entspricht, über die auf die erzeugte MediaWiki Instanz zugegriffen werden kann. Der dritte Rückgabeparameter entspricht zusätzlichen Informationen über die erzeugte Instanz. Im Fall von MediaWiki sind dies der Nutzername und das Passwort des Administrator Accounts, sofern die zugehörigen Attribute im NodeTemplate gesetzt wurden.

```
1 Success
2 Location: http://192.168.99.100:32771
3 Username: admin Password: password
```

Listing 8: Fallstudie: Ausgabe

Damit ist die Bereitstellung der MediaWiki Ressource für einen informellen Prozess abgeschlossen. Bei der tatsächlichen Anwendung der Software werden Ressourcen in ein großes System integriert. Dabei werden die einzelnen Schritte, die in der Fallstudie dargestellt wurden, automatisch vom umgebenden System veranlasst und benötigen keinen Eingriff vom Benutzer. Zudem werden die von der Software erstellten Rückgaben von diesem System verarbeitet und es ist damit möglich, alle Ressourcen, die für einen informellen Prozess benötigt werden, automatisch zu initialisieren.

6 Zusammenfassung und Fazit

Um erworbenes Wissen innerhalb eines informellen Prozesses wiederverwendbar zu machen, muss eine Möglichkeit gefunden werden, die Prozesse zu modellieren. Das Problem bei der Modellierung informeller Prozesse liegt darin, dass sie im Gegensatz zu Business Prozessen hauptsächlich von menschlichen Akteuren durchgeführt werden. Durch die damit entstehende Unvorhersehbarkeit der Aktivitäten des Prozesses, können übliche Modellierungsverfahren für informelle Prozesse nicht angewandt werden. Stattdessen ist es möglich, die Business Logik eines informellen Prozesses implizit zu beschreiben, indem die Ressourcen modelliert werden, die im informellen Prozess verwendet werden. Damit neben der Modellierung auch die teilweise automatische Ausführung eines informellen Prozesses ermöglicht werden kann, müssen alle Ressourcen des Prozesses automatisch initialisierbar gemacht werden. Da die Integration von Informations-Ressourcen im Kontext der informellen Prozesse bisher nicht untersucht wurde, soll die Arbeit diese Lücke schließen.

Das Ziel der vorliegenden Arbeit war es deshalb, eine Software zu erstellen, welche die automatische Integration von Informations-Ressourcen in informelle Prozesse durchführen kann. Dazu wurde zunächst eine Taxonomie entworfen, die das Einordnen von Datenquellen, anhand interessanter Eigenschaften für die Integration, ermöglicht. Bei der Erstellung der Taxonomie wurde festgestellt, dass unter anderem die Art und die Größe der Ressourcen interessant sind. Die Art der Ressourcen unterteilt sich in strukturierte Daten und unstrukturierte Daten. Da eine Darstellung einer Ressource in einer Datenquelle einer anderen Art üblicherweise keinen Sinn macht, muss dies in der Software beachtet werden. Die Größe einer Ressource spielt dagegen eine Rolle, da extrem große Ressourcen nicht ohne größere Verzögerungen übertragen werden können und diese deshalb nur lokal kopiert werden sollten.

Nach der Analyse der interessanten Eigenschaften von Datenquellen wurde die Software konzeptuell ausgearbeitet. Dazu wurde ein Use-Case Diagramm mit möglichen Anwendungsfällen für die Software und ein Klassendiagramm erstellt. Die Software wurde so entworfen, dass sie einfach um neue Datenquellen erweitert werden kann, indem nur einige Interfaces implementiert werden müssen. Anschließend wurde die grundlegende Logik der Software in der Programmiersprache Java umgesetzt und die Interfaces für alle Ressourcen des motivierenden Szenarios implementiert.

Durch die Erstellung der Software zur Integration von Informations-Ressourcen wurde das Ziel, Informations-Ressourcen automatisch in informelle Prozesse zu integrieren, um damit eine teilweise automatische Ausführung der Prozesse zu ermöglichen, erreicht. Die Software ermöglicht es, alle in Abschnitt 1.1.1 definierten Ressourcen und Relationen in informelle Prozesse zu integrieren. Da für allgemeine informelle Prozesse jedoch viele verschiedene Informations-Ressourcen verwendet werden können, ist an dieser Stelle eine Erweiterung der Software für Prozesse mit anderen Ressourcen nötig.

7 Ausblick

Die im Laufe der Arbeit entwickelte Software wird für die Integration von Informations-Ressourcen in informelle Prozesse verwendet. Hierbei konnte von den verschiedenen Datenquellen jedoch nur ein kleiner Teil bereits in die Software aufgenommen werden. Der einfachste Weg, die Integration von Ressourcen fortzuführen, ist, die vorhandene Software um weitere Datenquellen mit Informations-Ressourcen, wie zum Beispiel Google Docs oder Redmine, zu erweitern.

Neben der Integration neuer Datenquellen ist auch die Erweiterung der Software um zusätzliche Operationen denkbar. Bisher werden lediglich die sogenannten Lifecycle Operationen (Aquire, Store, Release) unterstützt. Weitere domänenspezifische Operationen könnten jedoch einen zusätzlichen Mehrwert zur Nutzung in informellen Prozessen darstellen.

Das Ziel der Integration von Ressourcen ist es, informelle Prozesse teilweise automatisiert ausführen zu können, indem die Initialisierung der Ressourcen automatisch durchführbar wird. Um dieses Ziel zu erreichen, müssen jedoch alle vier verschiedenen Arten von Ressourcen integriert werden. Die Wissens-Ressourcen und ein Teil der IT-Ressourcen werden bereits durch den Ansatz der Software abgedeckt. Noch offen sind dabei aber die Material-Ressourcen und die menschlichen Ressourcen. Material-Ressourcen spielen in informellen Prozessen im IT Umfeld häufig keine große Rolle. Menschliche Ressourcen sind jedoch elementar für jeden informellen Prozess. Aufgrund dessen ist es ein wichtiger Schritt die Integration von menschlichen Ressourcen in Zukunft genauer zu betrachten.

8 Abbildungsverzeichnis

Abb. 1 Motivierendes Szenario 10
Abb. 2 Integration von Ressourcen in informelle Prozesse [7] 11
Abb. 3 Aufbau eines Service Templates [10] 14
Abb. 4 Beispiel „Topology Template“ 14
Abb. 5 Docker Architektur 17
Abb. 6 Taxonomie für Datenquellen 22
Abb. 7 Beispiel „Einordnung von Datenquellen“ 24
Abb. 8 Use-Case-Diagramm der Software 27
Abb. 9 Klassendiagramm des Systems 32
Abb. 10 Klassendiagramm der Software (Teil 1) 34
Abb. 11 Klassendiagramm der Software (Teil 2) 35

9 Listingverzeichnis

Lst. 1 Beispiel „NodeType XML“ 16
Lst. 2 „data.properties“ Datei 46
Lst. 3 Fallstudie: Code Teil 1 49
Lst. 4 Fallstudie: Definitions Document 50
Lst. 5 Fallstudie: Code Teil 2 51
Lst. 6 Fallstudie: XML Schema Datei 52
Lst. 7 Fallstudie: Code Teil 3 53
Lst. 8 Fallstudie: Ausgabe 54

10 Quellenverzeichnis

[1] W. v. d. Aalst, A. t. Hofstede, and M. Weske. Business process management: A survey. In *Business process management*. Springer, 2003.

[2] F. Leymann and D. Roller. *Production Work Flow: Concepts and Techniques*. Prentice Hall PTR, 2000.

[3] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer New York, Inc., 2007.

[4] Object Management Group. Business process model and notation (BPMN) version 2.0. Technical report, Object Management Group, 2011.

[5] OASIS Standard. Web services business process execution language version 2.0, 2007.

[6] C. T. Sungur, T. Binz, U. Breitenbücher, and F. Leymann. Informal process essentials. In *Proceedings of the 18th IEEE Enterprise Distributed Object Conderence*, 2014.

[7] C. T. Sungur, U. Breitenbücher, F. Leymann, and J. Wettinger. Executing informal processes. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*, 2015.

[8] W. v. d. Aalst, M. Weske, and D.Grünbauer. Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, 2005.

[9] P. Dadam and M. Reichert. The adept project: A decade of research and development for robust and flexible process support - challenges and achievements. *Computer Science - Research and Development*, 2009.

[10] OASIS Standard. Topology and Orchestration Specification for Cloud Applications Version 1.0, 2013.

[11] T. Binz, G. Breiter, F. Leyman, and T. Spatzier. Portable cloud services using tosca. *IEEE Internet Computing*, 2012.

[12] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann. Tosca: portable automated deployment and management of cloud applications. In *Advanced Web Services*. Springer, 2014.

[13] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann. Winery – modeling tool for TOSCA-based cloud applications. In *11^{th} International Conference on Service-Oriented Computing*, 2013.

[14] Docker. Docker Webseite. `https://www.docker.com/`, 2016. [Stand 29. April 2016].

[15] J. Turnbull. *The Docker Book: Containerization is the new virtualization*. J. Turnbull, 2014.

[16] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014.

[17] M. Fowler. Inversion of control containers and the dependency injection pattern. `http://www.martinfowler.com/articles/injection.html`, 2004. [Stand 03. Mai 2016].

[18] J. Arthur and S. Azadegan. Spring framework for rapid open source j2ee web application development: a case study. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2005.

[19] C. Walls. Guice vs. Spring JavaConfig: A comparison of DI styles. `http://www.jroller.com/habuma/entry/guice_vs_spring_javaconfig_a`, 2007. [Stand 3. Mai 2016].

[20] S. Venugopal, R. Buyya, and K. Ramamohanarao. A taxonomy of data grids for distributed data sharing, management and processing. *Technical Report*, 2005.

[21] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003.

[22] L. Kassner and B. Mitschang. MaXCept – Decision Support in Exception Handling through Unstructured Data Integration in the Production Context. An Integral Part of the Smart Factory. In *Proceedings of the 48th Hawaii International Conference on System Sciences*, 2015.

[23] J. Vlissides, R. Helm, R. Johnson, and E. Gamma. Design patterns: Elements of reusable object-oriented software. *Addison-Wesley*, 1995.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Datum: .....

(Unterschrift)