ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Implementierung einer flexiblen Prozess-Engine auf Basis aktueller Webtechnologien

Masterarbeit an der Universität Ulm

Vorgelegt von:
Julian Sebastian Schregle
julian.schregle@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert
Dr. Rüdiger Pryss

Betreuer:
Johannes Schobel

2016

Fassung 28. November 2016

Satz: PDF-LaTeX 2ε

Kurzfassung

In der heutigen Zeit arbeiten immer mehr Unternehmen prozessorientiert. Arbeitsabläufe werden in grafischen Notationen erstellt, dokumentiert und anschließend nach diesem Schema ausgeführt. Um jedoch besser und schneller auf Kundenwünsche reagieren zu können, müssen Unternehmen flexibler werden. Hierfür werden zunehmend mehr anpassungsfähige, mobile Ausführungsplattformen für Geschäftsprozesse benötigt. Dazu ist eine Ausführung der Geschäftsprozesse direkt auf dem Gerät wünschenswert. Dadurch wird die Ausführung der Geschäftsprozesse unabhängig von einer Internetverbindung. Diese mobilen Ausführungsplattformen, auf denen die Geschäftsprozesse ausgeführt werden sollen, besitzen ein großes Problem: Werden sie plattformspezifisch entwickelt, können diese nur auf dem mobilen Betriebssystem ausgeführt werden, für das sie entwickelt wurden. Eine Ausführung auf einer anderen Plattform ist aufgrund der verwendeten Technologien nicht möglich. Eine Lösung für dieses Problem wäre eine plattformunabhängige Implementierung.

Im Zuge dieser Arbeit wird eine flexible, mobile Ausführungsplattform entwickelt. Dabei liegt der Fokus auf der Prozessausführung und der leichten Portierbarkeit auf andere Betriebssysteme. Zudem soll die Prozess-Engine leicht erweiterbar sein, sodass sich neue Funktionen ohne große Änderungen hinzufügen lassen. Die Prozess-Engine soll in der Lage sein konditionale Verzweigungen und parallele Ausführungen behandeln zu können. Durch diese Prozess-Engine lassen sich Geschäftsprozesse mobil auf unterschiedlichen Plattformen ausführen. Dies trägt dazu bei, dass prozessorientierte Unternehmen eine weitere Möglichkeit geboten bekommen, ihre Flexibilität zu erhöhen. Somit können sie in Zukunft schneller und besser auf Kundenwünsche reagieren.

Danksagung

Zuerst möchte ich mich bei meiner Familie bedanken, bei meinen Eltern Karl und Barbara und bei meiner Schwester Elisabeth, die mich immer unterstützt haben. Ein besonderer Dank geht auch an meine Freundin Tanja, die mich stets motiviert und unterstützt hat. Zusätzlich bedanken möchte ich mich bei meinen Freunden aus der Laufgruppe, die durch diese Arbeit das eine oder andere Mal ein etwas härteres Training ertragen mussten.

Ein großes Dankeschön geht an Herrn Prof. Dr. Manfred Reichert, Dr. Rüdiger Pryss und Johannes Schobel, ohne die diese Arbeit nicht möglich gewesen wäre. Vor allem für die Betreuung durch Johannes Schobel möchte ich mich bedanken. Die unermüdlichen und konstruktiven Diskussionen, die Anregungen und die immer motivierenden Rückmeldungen haben diese Arbeit in dieser Form erst möglich gemacht.

Inhaltsverzeichnis

1

Einleitung

Mobile Endgeräte liegen voll im Trend, egal ob Smartphone, Tablet oder E-Book Reader. Immer mehr Menschen setzen auf mobile Geräte anstatt eines stationären PCs oder Laptops. Gerade Firmen nutzen verstärkt mobile Endgeräte, um direkt beim Kunden Zugriff auf benötigte Daten zu bekommen und um Geschäftsprozesse in Gang zu setzen. Als Beispiel kann ein Schadensgutachter einer Versicherung genannt werden. Er kann mit seinem mobilen Gerät an Ort und Stelle auf die benötigten Daten zugreifen und direkt ein Gutachten erstellen. Hierbei würden die handschriftlichen Aufzeichnungen und der Übertrag in das Versicherungssystem entfallen. Außerdem würde bereits der Prozess der Schadensregulierung in Gang gesetzt. Die Anwendungsmöglichkeiten scheinen fast unendlich, mobile Endgeräte in den Geschäftsalltag zu integrieren. Man stelle sich vor, dass hierdurch immer effizienter und kundenfreundlicher gearbeitet werden kann. Somit erschließt sich ein riesiger Markt im privaten wie auch im geschäftlichen Sektor. Viele größere Unternehmen besitzen mittlerweile eine eigene mobile Applikation und mehr als die Hälfte der Angestellten arbeiten zum Teil auch mobil [1].

Somit wird es immer komplexer, Inhalte für die verschiedenen mobilen und stationären Plattformen bereitzustellen. Dies beginnt schon bei der Auswahl der mobilen Plattformen. Im Mai 2016 lag in Deutschland der Marktanteil der Smartphones mit Android bei ca. 67 % [2]. Immerhin noch mit ca. 29 % gefolgt von Apple's iOS. Mit ca. 2,5 % ist Windows in diesem Markt vertreten. In Summe beherrschen diese drei mobilen Betriebssysteme ca. 99 % des gesamten Marktes. Jedes dieser Betriebssysteme nutzt dabei eine andere Technologie. Android basiert auf Java, iOS auf Objective C und Windows auf C++/C#. Es kristallisiert sich hier bereits heraus, dass für diese drei Plattformen drei Implementierungen nötig sind, um 99 % des Marktes abdecken zu

können. Somit muss sich jeder, der eine Applikation entwickeln möchte, fragen, welche Betriebssysteme unterstützt werden sollen.
Die Herausforderung liegt hierbei darin, dass die Implementierungen ähnlich aussehen sollen. Außerdem sollen die internen Abläufe identisch sein, sodass bei einer Funktion, auf verschiedenen Plattformen, stets dasselbe Ergebnis resultiert. Nimmt man nun das Beispiel eines Geschäftsprozesses, so soll die Prozessausführung auf Android, iOS und Windows identisch sein. Jedoch soll auch das individuelle Look-and-Feel der jeweiligen Plattform mit berücksichtigt werden. Diese Anforderungen an die Applikation haben den kostspieligen Nachteil, dass parallel drei Implementierungen entwickelt werden müssen.

1.1 Problemstellung

Während immer mehr Menschen sich privat ein mobiles Gerät zulegen und der Markt für mobile Applikationen stetig wächst, war es nur eine Frage der Zeit, bis die Geschäftswelt auf diese Entwicklung reagiert. Da die Menschen nun vermehrt mobile Applikationen den Applikationen auf stationären Geräten vorziehen, müssen auch die Firmen nachlegen und flexibler werden. In einigen Firmen wird bereits mit wohldefinierten Geschäftsprozessen gearbeitet. Teilweise werden diese auch schon mobil genutzt. Jedoch gibt es oft noch gewaltige Einschränkungen in der Funktionalität, zum Beispiel in der offline Nutzung. Dies beginnt bereits bei der Entwicklung einer solchen Applikation. Im Stadium der Entwicklung müssen die Zielplattformen, auf denen die Applikation später laufen soll, festgelegt werden. Erlaubt die Firma auch noch den Bring-Your-Own-Device Ansatz, ist es fast unmöglich, mit einer Implementierung alle Geräte abzudecken. Ein weiteres Problem ist, dass die Ausführungskomponente oft auf nur einen vorliegenden Anwendungsfall angepasst wird und somit nicht wiederverwendbar ist. Daher wird es oft schwierig, nachträgliche Änderungen, zum Beispiel an der Notation des Prozessmodells, ohne hohen Aufwand umsetzen zu können. Ein weiteres Problem tritt, auf sobald eine Applikation mit mehreren Implementierungen für verschiedene Plattformen entwickelt wird: Soll eine Kleinigkeit an der Applikation geändert werden, muss diese für alle Implementierungen umgesetzt werden, was einen erheblichen Aufwand bedeutet. Zudem benötigen viele dieser Applikationen eine stetige Internetverbindung. In Städten

mag dies kein oder nur ein sehr geringes Problem darstellen. Auf dem Land oder in Gebäuden mit schlechtem Empfang ist die mangelnde Konnektivität immer noch ein großes Problem. Somit ist es wünschenswert, dass eine solche Applikation nur an wenigen Stellen eine Internetverbindung benötigt.

1.2 Zielsetzung

Um das diskutierte Problem anzugehen, soll eine mobile Multi-Device Applikation entwickelt werden. Dabei soll der Fokus auf der Ausführungskomponente und deren Flexibilität liegen. Um eine hohe Flexibilität und Wiederverwendbarkeit gewährleisten zu können, soll die Ausführungskomponente von der Benutzerschnittstelle komplett getrennt werden. Die Ausführungskomponente und die Benutzerschnittstelle sollen über eine wohldefinierte Schnittstelle miteinander kommunizieren. Darüber hinaus soll die Ausführungskomponente möglichst generisch aufgebaut sein. Dies soll sicherstellen, dass diese nicht nur die Ausführung eines bestimmten Anwendungsfalls beherrscht, sondern möglichst jeden strukturierten Ablauf. Zudem soll die Ausführungskomponente sehr modular aufgebaut sein, sodass sich alle Komponenten sehr leicht austauschen lassen. Dies hätte den Vorteil, dass bei Änderungen an einem Ablaufmodell nicht die gesamte Ausführungskomponente neu geschrieben werden muss, sondern lediglich die Import-Komponente umgebaut bzw. verändert werden muss. Die Multi-Device Fähigkeit soll über die Verwendung von entsprechenden Frameworks realisiert werden. Hierfür werden moderne Webtechnologien verwendet, die zunehmend beliebter werden. Da diese Technologien sich teilweise noch in der Entwicklung befinden, wird eine Diskussion erfolgen. Diese befasst sich damit, ob sich diese Technologien bereits für diese Art der Applikationen eignen beziehungsweise welche Kompromisse im Moment bei der Entwicklung mit diesen Technologien noch eingegangen werden müssen. Außerdem soll die Applikation in der Lage sein, strukturierte Abläufe von einem Server herunterzuladen, diese zu speichern und letztlich dann auch offline ausführen zu können. Darüber hinaus soll die Applikation erhobene Daten eines Ablaufs direkt auf einen Server übertragen können. Ferner soll sichergestellt werden, dass die Prozess-Engine an

möglichst wenigen Stellen eine Verbindung zum Internet benötigt und somit auch offline das Ausführen von strukturierten Abläufen zulässt.

1.3 Struktur der Arbeit

Der weitere Ablauf der Arbeit ist wie folgt:
Zunächst werden im Kapitel 2 die theoretischen Grundlagen aufgezeigt, die für das Verständnis dieser Arbeit notwendig sind. Unter anderem werden die Grundlagen zu hybriden mobilen Applikationen, zu Geschäftsprozessen und zu Representational State Transfer (REST) betrachtet. Im Kapitel 3 werden verwandte Arbeiten zu der hier vorliegenden Thesis vorgestellt und diskutiert. In Kapitel 4 werden die Anforderungen an die Implementierung erörtert, die in den darauf folgenden Kapiteln besprochen werden. Im Kapitel 5 wird die Architektur der entwickelten mobilen Applikation erklärt. Hierbei liegt der Fokus auf der Ausführungskomponente. In Kapitel 6 werden die verwendeten Frameworks behandelt und interessante Details der Implementierung vorgestellt. Anschließend werden Probleme diskutiert, die während der Implementierung aufgetreten sind. Kapitel 7 zeigt die Anwendungsmöglichkeiten der entwickelten mobilen Applikation anhand von Beispielen auf. Dies wird insbesondere am Beispiel des digitalen Fragebogen-Frameworks „QuestionSys“ verdeutlicht. Im letzten Kapitel werden die Erkenntnisse dieser Thesis zusammengefasst und es wird ein Ausblick gegeben, an welchen Stellen sinvolle Erweiterungen möglich wären.

2

Grundlagen

In diesem Kapitel werden die Grundlagen für die spätere Applikation gelegt. Zunächst wird der Begriff der hybriden Applikation näher erläutert. Anschließend wird in Abschnitt 2.2 auf das Prozessmodell und die Ausführungslogik eingegangen. Danach wird in Abschnnitt 2.3 das Konzept des Representational State Transfer (REST) behandelt.

2.1 Hybride Applikationen

Damit der Begriff der hybriden Applikation angemessen erläutert werden kann, müssen zudem die Begriffe der mobilen Webapplikation und der nativen Applikation behandelt werden. Zunächst wird in 1. die mobile native Applikation erläutert. Anschließend wird in 2. die mobile Webapplikation erklärt. Zu guter Letzt wird in 3. die mobile hybride Applikation behandelt (siehe Abbildung 2.1).

1. **Native Applikation:** Bei einer nativen Applikation handelt es sich um eine auf die entsprechende Plattform zugeschnittene Applikation. Sie verwendet direkt die Programmierschnittstelle der Plattform und besitzt somit direkten Zugriff auf die geräte-spezifische Hardware. Hierzu gehören zum Beispiel GPS, Beschleunigungssensoren oder die Kamera. Auch auf Software-Komponenten wie zum Beispiel das Dateisystem kann über die Programmierschnittstelle zugegriffen werden. Für solche nativen Applikationen ist deutlich mehr spezifisches Wissen über die entsprechende Plattform nötig als bei allen anderen hier vorgestellten Typen. Allerdings bieten native Applikationen mit Abstand die beste Leistung und die größtmögliche Funktionsvielfalt [4].

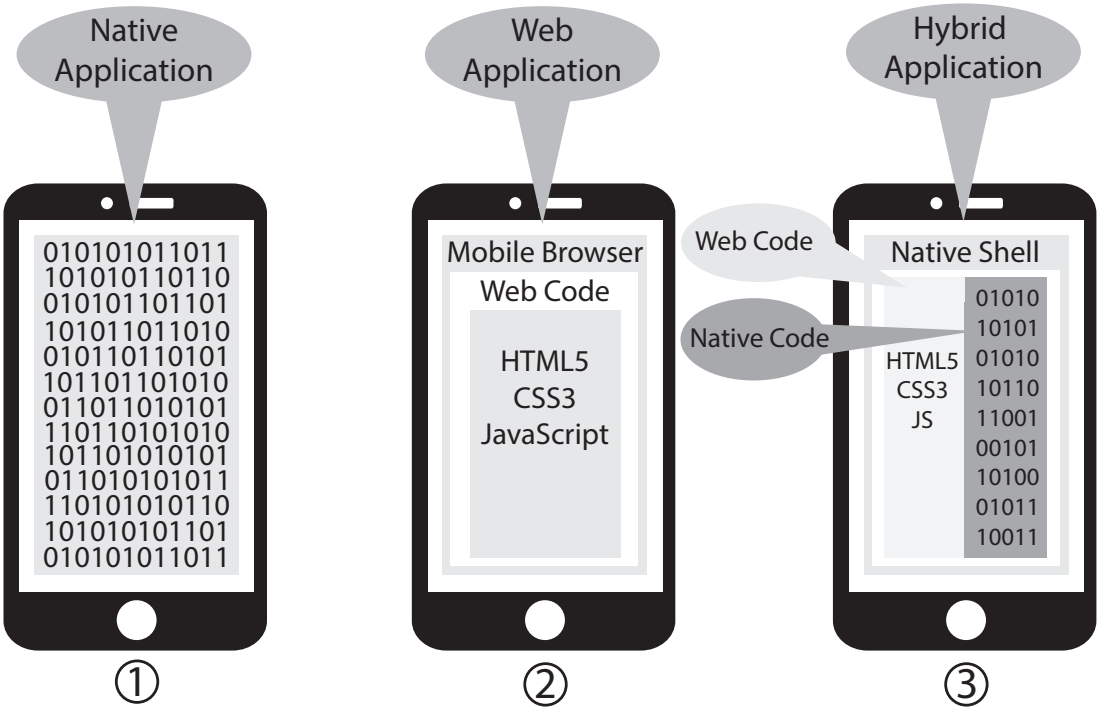

Abbildung 2.1: Verschiedene Applikations-Typen für mobile Endgeräte (nach [3]).

2. **Mobile Webapplikation:** Eine mobile Webapplikation ist eine Applikation, die nicht wie eine native Applikation auf dem mobilen Gerät des Benutzers installiert wird. Stattdessen befindet sich ein Großteil der datenverarbeitenden Schicht auf einem weit entfernten Webserver. Das Client-Gerät wird dabei zur Anzeige der Benutzerschnittstelle verwendet, welche in einem Webbrowser angezeigt und gerendert wird. Dabei werden Webtechnologien wie zum Beispiel HTML, CSS und JavaScript eingesetzt, um solche mobilen Webapplikation zu entwickeln. Darüber hinaus stellen mobile Webapplikation die einfachste Form der mobilen Applikationen dar und besitzen daher am wenigsten Funktionen [4].

3. **Hybride Applikation:** Eine hybride Applikation bettet eine Webapplikation in eine native Applikation ein, um so die Vorteile aus beiden Technologien verschmelzen zu lassen. Die Vorteile, die sich bei dieser Symbiose ergeben, sind einerseits eine plattformübergreifende Code-Basis, andererseits den Zugriff auf native Gerätekomponenten wie zum Beispiel auf die Kamera. Zudem besteht die Möglichkeit

der Offline-Nutzung. Für die Entwicklung hybrider Apps wird ein Framework, wie zum Beispiel Apache Cordova/Adobe PhoneGap, verwendet. Dabei verwenden diese Frameworks einen integrierten Webbrowser, bei dem bestimmte Funktionen, wie zum Beispiel den Zugriff auf die Adressleiste, deaktiviert sind [5]. Deshalb können Webtechnologien wie HTML, CSS und JavaScript zur Implementierung dieser hybriden Applikationen verwendet werden [6]. Zudem bietet der native Teil der Applikation einen direkten Hardwarezugriff [3].

Der größte Vorteil hybrider Applikationen ist, dass sie die Entwicklungskosten massiv senken. Dies folgt daraus, dass nicht parallel für mehrere Plattformen entwickelt werden muss. Es wird lediglich eine Code-Basis verwendet. Allerdings hat die Symbiose aus Webapplikation und nativer Applikation nicht nur Vorteile. Da die Applikation über eine zusätzliche Browserschicht verfügt, mit der interagiert werden muss, kann sich dies bei komplexen Anwendungen negativ auf die Leistung auswirken. Ein weiterer Nachteil ist jedoch, dass das jeweilige Plattform Look-And-Feel nicht immer komplett aufgegriffen werden kann. Jedenfalls ist dies nur als minimaler Nachteil zu werten, da dies plattformübergreifend zu einem einheitlicheren Erscheinungsbild der Applikation führt.

2.2 Prozessbeschreibung

Die Prozessbeschreibungssprache, sowie die Ausführungssemantik, die in diesem Abschnitt behandelt werden, sind für die hier vorliegende Thesis von enormer Bedeutung. Dabei wird eine blockbasierte Prozessbeschreibungssprache verwendet.
Die im folgenden Abschnitt vorgestellte Beschreibungssprache kann durchaus durch jede andere blockbasierte Sprache, zum Beispiel WS-BPEL [7], ersetzt werden. Im praktischen Teil müsste dafür lediglich ein zusätzlicher Wrapper implementiert werden. Dieser müsste die gewünschte, blockbasierte Sprache vorverarbeiten, sodass diese auf ADEPT abgebildet wird. Damit wäre es möglich, Prozessmodelle, die in einer anderen blockbasierten Sprache beschrieben wurden, zu verwenden.
In der hier vorliegenden Arbeit wird ADEPT als Prozessbeschreibungssprache verwendet.

2.2.1 Prozessmodell

Das Projekt *Application Development based on Encapsulated Pre-modeled Process Templates* „ADEPT“ wird seit 1997 [8] an der Universität Ulm entwickelt. ADEPT ist dabei eine Beschreibungssprache für Geschäftsprozesse, sodass diese durch ein Computersystem ausgeführt werden können. Die Prozessmodelle, die mit ADEPT modelliert werden, basieren auf azyklischen Graphen, bestehend aus Knoten, Kontrollflusskanten, Datenelementen und Datenflusskanten. Die Abbildung 2.2 zeigt eine Übersicht der Elemente, die in einem Prozessmodell enthalten sein können.

1. Blockstrukturierung: Jedes Prozessmodell ist blockstrukturiert. Das heißt, dass jeder Block nur einen Startknoten und einen Endknoten besitzt [9] und somit wohlgeformt ist. Wohlgeformt bedeutet in diesem Fall, dass jeder Block frei von Überlappungen sein muss. Das bedeutet, dass zum Beispiel auf einen AND-Split-Knoten kein OR-Join-Knoten folgen darf, sondern ein AND-Join-Knoten den Block abschließen muss.

2. Knoten/Aktivitäten: Wie erwähnt, gibt es in jedem ADEPT Prozessmodell nur einen Start- und Endknoten. Der Startknoten ist dabei vom Typ `NT_STARTFLOW`, der Endknoten vom Typ `NT_ENDFLOW` [9]. Die einfachste Form eines Knotens ist eine Aktivität die den Typ `NT_NORMAL` darstellt. Sie besitzt lediglich eine eingehende und eine ausgehende Kante. Ein Knoten kann nicht nur die oben genannten Basis-Typen annehmen, sondern auch einen Subprozess darstellen. Dies ist aufgrund der Blockstrukturierung problemlos möglich. Außerdem repräsentiert jeder Knoten in einem ADEPT Prozessmodell stets einen Prozessschritt [10].

3. Gateways: Es existieren drei Gateway-Typen: AND-, XOR- und LOOP-Gateways. Jedes dieser Gateways besteht aus zwei Teilen, einem Split und einem zugehörigen Join-Knoten [9]. Split-Knoten haben eine eingehende und mehr als eine ausgehende Kante. Join-Knoten hingegen besitzen mindestens eine eingehende, aber nur eine ausgehende Kante. Durch dieses Paradigma sind auch die Verzweigungen blockstrukturiert [9]. Es existieren dabei AND-Verzweigungen (`NT_AND_SPLIT`), sowie XOR-Verzweigungen

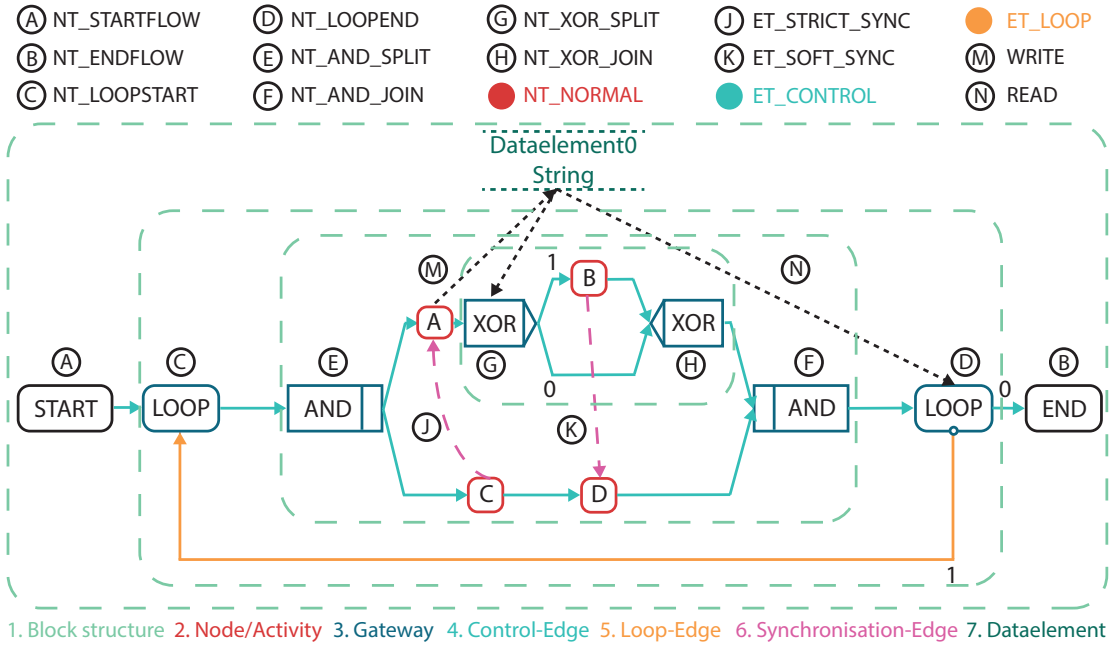

Abbildung 2.2: Übersicht der ADEPT-Elemente: Knoten, Kanten, Gateways, Schleifen, Datenelementen und Synchronisationskanten.

(`NT_XOR_SPLIT`) [11]. Aus der oben genannten Bedingung, dass jeder Split-Knoten einen korrespondierenden Join-Knoten benötigt, resultieren die Join-Knoten-Typen `NT_AND_JOIN` und `NT_XOR_JOIN`. Außerdem existiert ein Schleifen-Element, das den Typ `NT_STARTLOOP` für die Verzweigung und `NT_ENDLOOP` für den korrespondierenden Join-Knoten erhält. Das Besondere an dem Schleifen-Element ist dabei, dass der `NT_ENDLOOP` eine Bedingung für den Rücksprung auf den `NT_STARTLOOP` enthält, sowie eine zugehörige Kante für den Rücksprung [9]. Dabei besitzt der `NT_STARTLOOP`-Knoten zwei eingehende und eine ausgehende Kante. Der `NT_ENDLOOP`-Knoten besitzt zwei ausgehende und eine eingehende Kante.

4. Kontrollflusskante: Neben den Knoten existieren Kontrollflusskanten, die die Reihenfolge zwischen den Prozessschritten definieren. Auch diese können verschiedene Typen annehmen. Der einfachste Fall ist die Sequenz, die über den Typ `ET_CONTROL` repräsentiert wird. Dabei stellt beispielsweise eine Kante von Knoten A nach Knoten B

sicher, dass A vor B ausgeführt wird. Darüber hinaus muss Knoten A beendet sein bzw. nicht mehr zur Ausführung kommen, damit Knoten B gestartet werden darf.

5. Loopkante: Wie bereits erwähnt, besitzt eine Schleife eine Kante von dem Join-Knoten zum Start-Knoten. Durch die eben genannte Ausführungsbedingung benötigt die Schleife daher einen eigenen Kanten-Typ, für diesen Fall den `ET_LOOP`.

6. Synchronisationskante: Um komplexere Prozessmodelle abbilden zu können, existiert noch ein zusätzlicher Kanten-Typ, die Synchronisationskante. Auch diese Synchronisationskante besitzt zwei Typen. Zum einen die einfache `ET_SOFT_SYNC` und zum anderen die strikte `ET_STRICT_SYNC` Kante. Existiert zwischen Knoten A und Knoten B eine einfache Synchronisationskante, bedeutet dies, dass Knoten B erst ausgeführt werden darf, wenn Knoten A erfolgreich beendet wurde oder feststeht, dass Knoten A nicht mehr zur Ausführung kommt. Bei der strikteren Variante darf Knoten B nur ausgeführt werden, wenn Knoten A erfolgreich beendet wurde. Dies stellt eine Erfolgsabhängigkeit in der Prozessausführung dar [9].

7. Datenelemente: Datenelemente besitzen stets einen Datentyp. Dabei werden die typischen Basis-Datentypen (`STRING`, `INTEGER`, `FLOAT` und `BOOLEAN`) verwendet. Zudem existieren Ressourcenbezeichner `URI` (Uniform Resource Identifier), sowie der `UDT` (User-Defined Data Type für benutzerdefinierte Typen).
Datenelemente werden über Datenkanten mit Knoten verknüpft. Dabei gibt es zwei Typen: `WRITE` und `READ`. Eine Kante vom Knoten zum Datenelement handelt es sich um eine `WRITE`-Operation. Verläuft die Kante allerdings vom Datenelement zum Knoten handelt es sich um eine `READ`-Operation.

2.2.2 Ausführungslogik

In diesem Abschnitt wird nun die Ausführungslogik eines ADEPT Prozessmodells behandelt. Zunächst werden die verschiedenen Knoten-Zustände erläutert. Im nächsten Abschnitt wird die Aktivierungssemantik dieser Zustände besprochen.

Knotenzustände

Die Abbildung 2.3 gibt einen Überblick über die einzelnen Zustände, die ein Knoten in einem ADEPT Prozessmodell annehmen kann. Dabei existieren für die einzelnen Zustände zunächst Superklassen: `WAITING`, `RUNNING` und `TERMINATED`.
Initial beginnt jeder Knoten mit dem Zustand `NOT_ACTIVATED`. Hierbei sind die Vorbedingungen, die der Knoten benötigt, um seine Aufgabe erfüllen zu können, noch nicht erfüllt. Sind diese Vorbedingungen erfüllt, wechselt der Knoten in den Zustand `ACTIVATED`. Erfüllt der Knoten seine Aufgabe automatisch, dann wechselt er direkt in den Zustand `STARTED`. Benötigt er jedoch eine Interaktion durch einen (oder mehrere) Benutzer, bleibt er zunächst im Zustand `ACTIVATED`. Erst wenn der Knoten durch den Benutzer ausgewählt wird, wechselt dieser seinen Zustand. Dabei wird der Knoten in den Zustand `SELECTED` überführt. Hieraus kann der Benutzer die Ausführung des Knotens manuell starten, sodass dieser in den Zustand `STARTED` überführt wird. Befindet sich ein Knoten entweder im Zustand `ACTIVATED` oder `SELECTED`, kann dieser jederzeit wieder deaktiviert werden. Dabei wird der Knoten wieder in den Zustand `NOT_ACTIVATED` überführt. Die eben vorgestellten Zustände `NOT_ACTIVATED`, `ACTIVATED` und `SELECTED` werden der Superklasse `WAITING` zugeordnet.
Sobald ein Knoten in den Zustand `STARTED` wechselt, beginnt die eigentliche Ausführung der hinterlegten Aufgabe. Ausgehend von dem Zustand `STARTED` kann ein Knoten noch in den Zustand `SUSPENDED` überführt werden. Dies ist notwendig, da nicht bei allen Aufgaben davon auszugehen ist, dass sie an einem Stück bearbeitet werden kann. Er signalisiert dabei, dass es sich um eine Unterbrechung der Aufgabenbearbeitung handelt. Diese Zustände werden der Superklasse `RUNNING` zugeordnet.
Wenn sich ein Knoten im Zustand `STARTED` befindet, sind die folgenden Übergänge möglich. Beendet der Knoten seine Aufgabe erfolgreich, wechselt er in den Zustand

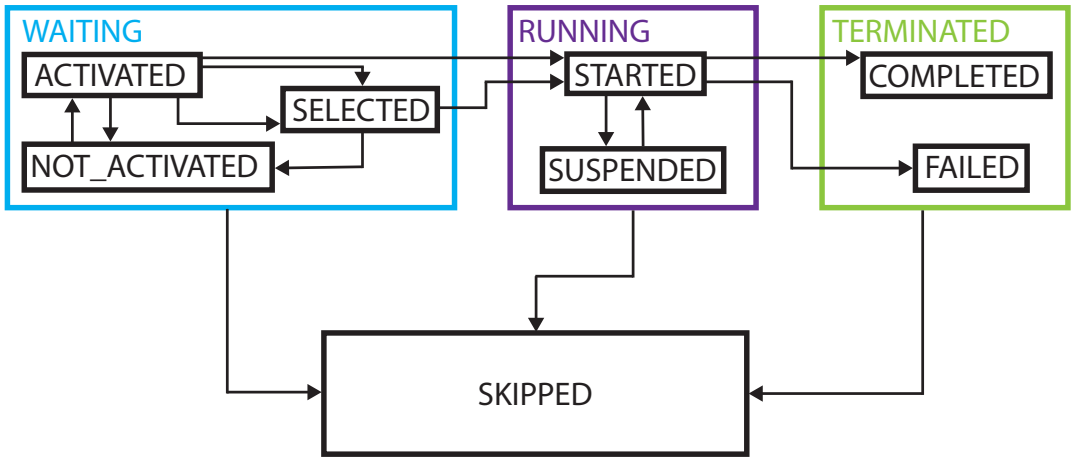

Abbildung 2.3: Zustands-Typen und die Übergänge zwischen den Zuständen (nach [9]).

`COMPLETED`. Ist dies nicht der Fall, also wenn die Aufgabe fehlerbehaftet beendet, oder abgebrochen wurde, wird der Knoten in den Zustand `FAILED` überführt. Die beiden Zustände `COMPLETED` und `FAILED` werden der Superklasse `TERMINATED` zugeordnet. Abschließend gibt es noch der Zustand `SKIPPED`. Dieser findet Verwendung, wenn ein Knoten nicht mehr zur Ausführung kommt oder noch nicht gestartet wurde.

Zusätzlich zu den Knotenzuständen gibt es Zustände für die Kontrollflusskanten. Sie nehmen die Zustände `NOT_SIGNALED`, `TRUE_SIGNALED` und `FALSE_SIGNALED` an. Eine ausgehende Kontrollflusskante kann den Zustand `NOT_SIGNALED` nur erreichen, wenn der Zustand des Quellknotens den Superklassen `WAITING` und `RUNNING` zuzuordnen ist. Besitzt ein Knoten entweder den Zustand `SKIPPED` oder `FAILED`, besitzt die ausgehende Kontrollflusskante den Zustand `FALSE_SIGNALED`. Andernfalls ist die ausgehende Kontrollflusskante des Quellknotens im Zustand `TRUE_SIGNALED` [9].
Um nun einen konsistenten Ablauf der Ausführung sicherstellen zu können, muss die Aktivierung der Knoten noch näher spezifiziert werden. Der nächste Abschnitt beschäftigt sich aus diesem Grund mit der Aktivierungssemantik.

Aktivierungssemantik

Da ein Knoten je nach Typ mehrere eingehende und ausgehende Kontrollflusskanten besitzen kann, müssen Aktivierungsregeln aufgestellt werden, die eine konsistente Prozessausführung gewährleisten. Die Tabelle 2.1 gibt einen Überblick über die verschiedenen Typen sowie deren Aktivierungsregeln. Hierbei sticht zunächst der `NT_STARTFLOW` heraus. Dieser Typ besitzt keine eingehende Kontrollflusskante, somit existiert auch keine Aktivierungsregel. Dieser Knoten muss manuell durch den Benutzer aktiviert werden.
Die nächsten Knotentypen, die auffallen, sind `NT_XOR_JOIN` und der `NT_STARTLOOP`. Beide Typen besitzen mehrere eingehende Kontrollflusskanten und nur eine ausgehende Kontrollflusskante. Sie verwenden dazu die Aktivierungsregel `AT_LEAST_ONE` [9]. Sie signalisiert, dass mindestens eine eingehende Kontrollflusskante sich im Zustand `TRUE_SIGNALED` befinden muss, damit der Knoten aktiviert wird. Bei `AT_LEAST_ALL` müssen alle eingehenden Kontrollflusskanten diesen Zustand erreicht haben.

In diesem Abschnitt wurde der Aufbau des Prozessmodells mit der zugehörigen Ausführungslogik behandelt. Damit wurden die theoretischen Grundlagen für das Prozessmodell und für dessen Ausführung gelegt.

Knotentyp	Aktivierungsregel
NT_STARTFLOW	-
NT_NORMAL	AT_LEAST_ALL
NT_AND_SPLIT	AT_LEAST_ALL
NT_AND_JOIN	AT_LEAST_ALL
NT_XOR_SPLIT	AT_LEAST_ALL
NT_XOR_JOIN	AT_LEAST_ONE
NT_STARTLOOP	AT_LEAST_ONE
NT_ENDLOOP	AT_LEAST_ALL
NT_ENDFLOW	AT_LEAST_ALL

Tabelle 2.1: Aktivierungsregeln der einzelnen Knotentypen.

2.3 Representational State Transfer

Mit Representational State Transfer (REST) bezeichnet man ein Architektur-Paradigma für verteilte Systeme [12]. REST wird besonders im Kontext von Webservices eingesetzt. Dabei wird über den Universal Resource Identifier (URI) eine Information (Ressource) adressiert. Zu den Designprinzipien von REST gehört die Zustandslosigkeit [13]. Dabei sollen weder vom Server noch vom Client zwischen den Nachrichten Zustandsinformationen abgespeichert werden. Dies ist nicht nötig, da alle notwendigen Informationen bereits in der URI enthalten sind. Wie bereits erwähnt, werden Informationen bei REST über eine URI adressiert. Hieraus lässt sich ableiten, dass bei gleicher URI immer die gleiche Ressource zurückgegeben werden muss. Zudem fungiert REST als unabhängige Schnittstelle, da der Zugriff unabhängig von der verwendeten Programmiersprache aus erfolgt.

Hauptsächlich erfolgen Anfragen an einen REST-Webservice über HTTP oder HTTPS. HTTP-Methoden können dabei bestimmte Eigenschaften erfüllen. Zu den wichtigsten beiden Eigenschaften gehört die Idempotenz und die Sicherheit.

Sicherheit: Sichere Methoden dürfen am Zustand des Servers nichts verändern, da es sonst zu Seiteneffekten führen könnte [14].

Idempotenz: Idempotente Methoden besitzen bei mehrmaligen identischen Aufrufen stets dieselben Seiteneffekte wie bei einem einmaligen Aufruf [14].

Dabei werden Anfragen an den REST-Webservice, mit den in Abbildung 2.4 illustrierten HTTP-Anfragemethoden abgewickelt.

GET: ist die am meisten verwendete und wichtigste HTTP Methode. Außerdem zählt *GET* zu den idempotenten Methoden und erfüllt die Eigenschaft einer sicheren Methode. Mit *GET* wird lesend auf Ressourcen Zugegrifen.

PUT: Im Gegensatz zu *GET* werden mit *PUT* neue Ressourcen erstellt, sofern diese noch nicht existieren. Andernfalls wird die Ressource überschrieben.

POST: Mit *POST* wird eine neue Ressource mit einer eindeutigen URI angelegt. Handelt es bei der Ressource um eine Liste, werden die Daten als Sub-Ressource zur bereits existierenden Ressource hinzugefügt. Dem Client wird die URI der

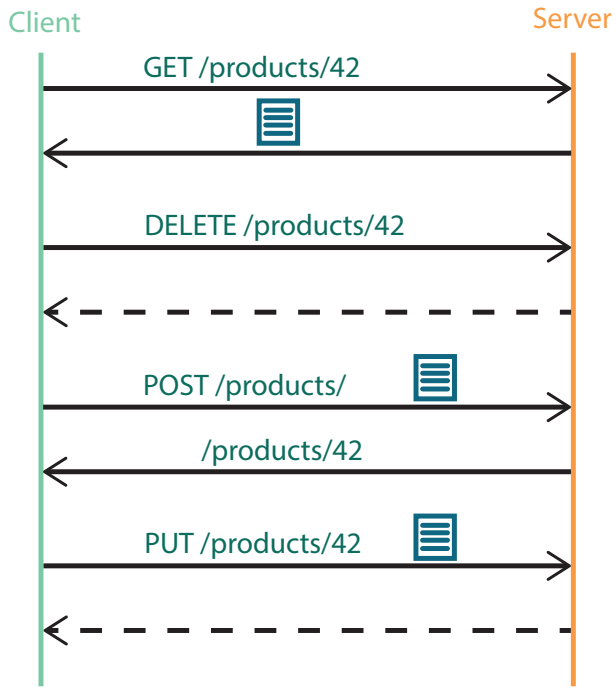

Abbildung 2.4: Übersicht der HTTP-Methoden, die bei REST verwendet werden.

neuen Ressource beziehungsweise der Sub-Ressource zurückgegeben. Außerdem gehört *POST* zu den idempotenten Methoden.

DELETE: Die Funktion von *DELETE* ist schon fast selbsterklärend. Dabei wird die über die URI identifizierte Ressource gelöscht. Wie auch *GET* und *POST* zählt *DELETE* zu den idempotenten Methoden.

In diesem Kapitel wurden die theoretischen Grundlagen für die nachfolgenden Kapitel gelegt. Dabei wurde zunächst eine Übersicht der unterschiedlichen mobilen Applikations-Typen gegeben. Anschließend wurden die Prozessmodelle behandelt. Hierbei wurde zunächst auf das eigentliche Prozessmodell eingegangen. Nachfolgend wurde die Ausführung des Prozessmodells näher betrachtet. Zuletzt wurde das REST Paradigma behandelt. Im nächsten Kapitel werden nun verwandte Arbeiten vorgestellt.

3

Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten vorgestellt, die sich mit mobiler Prozessausführung beschäftigen. Die Arbeiten unterscheiden sich in der Prozessbeschreibung, der Implementierungsbasis sowie in den unterschiedlichen Anwendungsbereichen. Während Sliver eine Ausführungs-Engine für BPEL bereitstellt, ist ROME4EU ein komplettes Prozess Management System für Krisensituationen, bei dem der Teamleiter sein Team mobil koordiniert. MARPLE bietet ebenfalls ein Prozess Management System, welches die Prozessausführung auf mobilen Geräten zulässt. DEMAC beschäftigt sich mit der Ausführung von verteilten Geschäftsprozessen. Das AllenaVita Projekt stellt eine mobile Lifestyle Coaching Applikation dar, die die Abläufe auf Prozessmodelle abbildet. Diese können dann auf mobilen Geräten ausgeführt werden. Zu guter Letzt bietet WOtAN eine auf Android basierende Prozessausführungs- und Analyse-Engine. Mit WOtAN können ADEPT basierte Prozessmodelle ausgeführt werden.

3.1 Sliver

„Sliver“ ist eine WS-BPEL Workflow Prozessausführungs-Engine für mobile Geräte [15], die an der Washington University in St. Louis entwickelt wurde. Dabei wurden bei dieser Engine die Kommunikation und die Prozessausführung strikt voneinander getrennt. Dies ist in Abbildung 3.1 ersichtlich, die die Sliver-Architektur illustriert. Sliver verwendet für die Kommunikation das Simple Object Access Protocol (SOAP) Netzwerkprotokoll. Für die Prozessbeschreibung wird WS-BPEL verwendet [15]. Ein großes Entwicklungsziel bei Sliver war die Leichtigkeit der gesamten Applikation, da diese auch auf mobilen

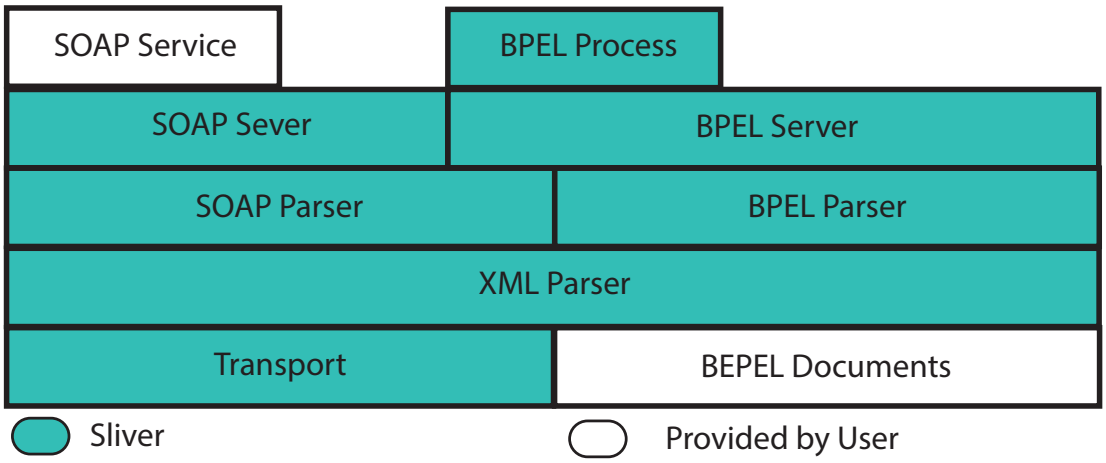

Abbildung 3.1: Übersicht der Sliver Architektur im Schichtmodell (nach [15]).

Geräten ausführbar sein sollte. Deshalb wurden auch nur zwei sehr leichtgewichtige, externe Bibliotheken bei der Entwicklung verwendet.

Transportschicht: Die Transportschicht ist verantwortlich für das Übertragen von Nachrichten, die von den oberen Schichten (SOAP und BPEL) produziert werden. Dabei unterstützt Sliver TCP/IP Sockets der Java Standard Edition (J2SE) oder auf Mobile Information Device Profile (MIDP) basierten Geräten. Des Weiteren bietet Sliver eine direkte HTTP Unterstützung. Diese steht nur auf J2SE basierten Geräten zur Verfügung. Möchte ein Entwickler ein anderes Protokoll für die Kommunikation verwenden, lässt sich Sliver einfach erweitern, da die benötigten Schnittstellen hierfür ausgelegt sind.

XML-Parser/Soap-Parser: Da sowohl SOAP für Nachrichten als auch BPEL für die Prozessbeschreibung auf XML arbeiten, besitzt SLIVER eine eigene Parsing Schicht für XML-Dokumente. Um alle Dokumente zu parsen wird kXML verwendet.

SOAP Server: SOAP stellt bereits einen Standard für Remote Procedure Calls (RPC) bereit. Diese werden vom SOAP-Server implementiert.

BPEL Parser: BPEL Prozessmodelle nutzen ein standardisiertes XML Schema. Der BPEL Parser assoziiert jedes dieser Elemente als eigene Java-Klasse. Dabei werden von Sliver nur die BPEL-Kernfunktionen implementiert und nicht der komplette

Standard. Dies ist nötig, damit die Applikation auch weiterhin sehr leichtgewichtig bleibt. Deshalb bietet Sliver einen sehr kleinen Funktionsumfang, der nur durch Aktivitäten repräsentiert ist. Das Prozessmodell wird auf Java-Klassen abgebildet, sodass daraus der eigentliche Prozess entsteht.

BPEL Server: Der BPEL Server stellt den BPEL Prozess, der vom Parser generiert wurde, als Service bereit. Dabei stellt ein BPEL Server eine Spezialisierung eines SOAP Servers dar, um Prozesse an Stelle der SOAP Services anbieten zu können.

Zusammenfassend lässt sich feststellen, dass „Sliver“ einen guten Ansatz für die mobile Ausführung BPEL-Basierter Prozesse bietet. Außerdem werden XOR- und AND-Gateways von Sliver unterstützt [15]. Jedoch bietet Sliver keine vollständige BPEL-Unterstützung, sondern lediglich die Kernfunktionen.

3.2 ROME4EU

Das Projekt „Roman Orchestration Mobile Engine for Emergency Units“ (ROME4EU) [16] wurde von 2006 bis 2009 an der Universität Rom entwickelt. Ziel war es, ein Prozess Management System für mobile Endgeräte zur Koordination von Arbeitern, Geräten, Robotern und Sensoren in Krisenfällen zu entwickeln. Um beispielsweise die Auswirkungen von Naturkatastrophen schneller beseitigen zu können, werden die Teams zum Aufräumen mit PDAs ausgestattet. Damit diese nicht über das normale Mobilfunknetz arbeiten müssen (nach Hurrikan Catrina war die Kommunikation über das Telefonnetz nur schlecht möglich), wird ein Ad-Hoc-MANET über dem Gebiet aufgebaut [17]. Jedes Team ist einem Team-Leiter unterstellt, der Aufgaben koordiniert und an seine Team-Mitglieder verteilt. Diese Aufgaben müssen dann von den Team-Mitgliedern abgearbeitet werden. Bei der Ausführung dieser Aufgaben bewegen sich die Team-Mitglieder unter Umständen auch außerhalb des Empfangsbereichs. Deshalb muss die ROME4EU auch mit solchen Szenarien zurechtkommen.
ROME4EU wurde nach service-orientierten Paradigmen entworfen. Im Prinzip wird zwischen zwei Typen von Aufgaben unterschieden. Kann die Aufgabe automatisch ausgeführt werden, spricht man von einer Applikation. Werden menschliche Interaktionen

benötigt, spricht man von einem Task [17]. Applikationen und Tasks werden dabei für die einzelnen Netzwerkknoten als Service angeboten. Tasks werden in einem Task-List-Handler verwaltet. Zur Koordination und zur Ausführung wird ein Prozess als Web-Service benötigt, der in WS-BPEL definiert wird. Dabei kann ROME4EU bedingte Ausführungen (XOR-Split/Join) ebenso ausführen wie parallele Ausführungen (AND-Split/Join) [17]. Um die dynamische Zuordnung der Tasks an Arbeiter gewährleisten zu können, werden Web-Services verwendet, die nur dann aufgerufen werden, bevor der eigentliche Web-Service ausgeführt wird.

ROME4EU bietet ein Prozess Management System, das speziell für mobile Geräte entwickelt wurde. Es kann autonom Prozesse ausführen und kommt mit Verbindungsabbrüchen zurecht. Die größte Schwachstelle ist jedoch das Gerät des Team-Leites. Fällt das Gerät aus oder ist dauerhaft außer Reichweite, wird das komplette Team lahmgelegt.

3.3 Marple

Das Projekt „MAnaging Robust mobile Processes in a compLEx world“ (MARPLE) wird seit dem Jahr 2009 an der Universität Ulm entwickelt. Ziel dieses Projekts ist die Entwicklung einer leichtgewichtigen Prozess-Engine, die die Ausführung von zentral gespeicherten Prozessmodellen erlaubt. Zudem ist die Applikation für verschiedene mobile Plattformen verfügbar. MARPLE verwendet eine Client-Server Infrastruktur [18]. Des Weiteren werden die ADEPT Design Prinzipien, zum Beispiel „correctness-by-construction“ oder „dynamic process adaptability“ verwendet. MARPLE implementiert das Konzept von ADEPT und modifiziert es speziell für den Einsatz auf mobilen Geräten. MARPLE besteht aus zwei grundlegenden Komponenten, dem MARPLE Mediation Center und der MARPLE Mobile Engine [19]. Diese Komponenten werden in Abbildung 3.2 dargestellt.

MARPLE Mediation Center Das Mediation Center setz sich aus vier Komponenten zusammen: der Modellierungs-Komponente, dem Repository, der Kontrolleinheit und der Wartung, die nachfolgend beschrieben werden.

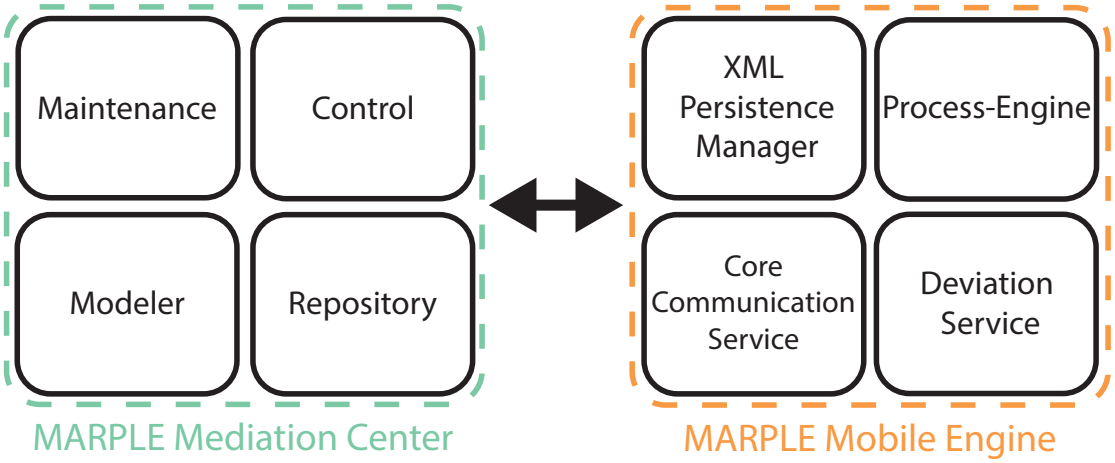

Abbildung 3.2: Übersicht der MARPLE Architektur (nach [19]).

Modellierungs-Komponente In der Modellierungs-Komponente können die eigentlichen Prozessmodelle erstellt, konfiguriert und getestet werden.

Repository Das Repository verwaltet sämtliche Daten des Mediation Centers. Dazu gehören unter anderem die PDA-Konfigurationen, die Prozessmodelle, die Aktivitätenmodelle und die Instanz-Daten.

Kontrolleinheit In der Kontrolleinheit lassen sich Prozesse den Benutzern zuordnen. Außerdem bietet sie die Möglichkeit der Ad-hoc Änderungen am laufenden Prozess. Dazu lassen sich Prozesse überwachen und diese auf andere Geräte migrieren.

Wartung In der Wartung lassen sich unter anderem die PDAs Konfigurieren und die MARPLE Mobile Engine installieren.

MARPLE Mobile Engine Die MARPLE Mobile Engine besteht ebenfalls aus vier Komponenten: der Prozess-Engine, dem Deviation Dienst, dem Core Communication Dienst und dem XML Persistence Manager.

Prozess-Engine Die Prozess-Engine ist zuständig für die Ausführung der Prozesse.

Deviation Dienst Der Deviation Dienst wird verwendet, um dynamische Adaptionen einer laufenden Prozessinstanz zu verwalten.

Core Communication Dienst Der Core Communication Dienst ist für die Kommunikation mit dem MARPLE Mediation Centre zuständig. Dazu gehört auch, dass dieser Dienst nur die Teile vom MARPLE Mediation Center herunterlädt, die auch tatsächlich benötigt werden [19].

XML Persistence Manager Der XML Persistence Manager verwaltet die heruntergeladen Prozess- und Activity-Modelle.

Somit bietet MARPLE eine mobile Prozess-Engine für iOS und Android [20].

Im Rahmen des MARPLE-Projektes wurde unter anderem der MEDo-Ansatz entwickelt. Dabei handelt es sich um eine Single Task Ausführungsplattform, die auf die Anwendung im Bereich der medizinischen Visite spezialisiert ist [21].

3.4 DEMAC

Das Projekt „Distributed Environment for Mobility-Aware Computing“ (DEMAC) wird seit 2005 an der Universität Hamburg entwickelt. Dieses Projekt beschäftigt sich mit der Ausführung von verteilten Geschäftsprozessen, die auch mobil ausgeführt werden sollen. Solche verteilten Geschäftsprozesse sind Prozesse, die räumlich oder organisatorisch voneinander getrennt verlaufen. Verbunden werden sie nur über einen asynchronen Kommunikationskanal, der eine übergeordnete Schicht der Transportprotokolle darstellt [22]. Diese verteilt ausgeführten Prozesse benötigen aufgrund der hohen Komplexität des Zusammenspiels der Systeme und der Mobilität von Komponenten einen hohen Bedarf an Anpassungsfähigkeit, um auch auf unerwartete Situationen reagieren zu können [23]. Im Rahmen dieses Projektes wird eine komponentenbasierte Middleware entwickelt. In [24] wird die Prozessausführungssprache „DEMAC Process Description Language“ (DPDL) und ein zugehöriges Ausführungsmodell für mobile, verteilte Geschäftsprozesse vorgeschlagen. Eine mobile Prozessausführungs-Engine, die die entsprechenden Modelle interpretieren und ausführen kann, wurde ebenfalls umgesetzt. Sie unterstützt bedingte und parallele Ausführungen [25].
Das DEMAC Projekt bietet eine komponentenbasierte Middleware, die zur Laufzeit Anpassungen des Prozesses zulässt [23].

3.5 AllenaVita

Das Projekt „AllenaVita“ wurde zwischen 2013 und 2014 an der Universität Trento in Kooperation mit einem lokalen Software-Unternehmen entwickelt. Dabei handelt es sich um eine mobile Lifestyle Coaching Applikation, mit der medizinisches Personal mit Patienten kommunizieren kann. Hierzu kann beispielsweise eine Krankenschwester die Vitalparameter oder Ziele eines Patienten überwachen und ihm Feedback oder einen medizinischen Bericht zusenden [26]. Dabei können die Ziele eines Patienten sämtliche Vitalparameter umfassen: Zum Beispiel das Wunschgewicht, der Fitnesszustand oder die Herzfrequenz können hier überwacht werden. Der Patient benutzt eine Android Applikation, die eine mobile Prozess-Engine besitzt. Das medizinische Personal verwendet eine Webapplikation und einen graphischen Prozess-Designer [27]. Als Prozessbeschreibung verwendet AllenaVita die BPMN 2.0. Hierbei wird während der Ausführungsphase ein mobiler Prozess von einer entfernten Quelle auf das mobile Gerät geladen. Während der Ausführungsphase wird für die Kommunikation mit Web-Services eine Internetverbindung benötigt. Die Applikation kann außerdem mit externen Peripherie-Geräten, wie einem Blutdruckmessgerät kommunizieren [27]. Sind Benutzer-Interaktionen während der Ausführung nötig, existiert auch eine Benutzeroberfläche für die verschiedensten Anwendungen. Die Aufgaben, die von einem Patienten ausgeführt werden sollen, können auch offline ausgeführt werden.
AllenaVita bietet eine mobile Prozess-Engine,die BPMN 2.0 unterstützt. Jedoch ist die Offline-Fähigkeit dieser Applikation beschränkt, da für die Ausführung von Aktivitäten, auch ohne Benutzerinteraktion, direkt ein Web-Service verwendet werden kann. Zudem ist die Applikation nur für Android Geräte verfügbar.

3.6 WOtAN

Das Framework „Workflows on Android “ (WOtAN) wurde 2016 im Rahmen einer Master-Thesis entwickelt [28]. Bei der Entwicklung von WOtAN lag der Fokus auf der Prozessausführung und der Prozessanalyse. Die Prozessmodelle, die auf WOtAN ausgeführt werden können, verwenden die ADEPT Prozessbeschreibungssprache [29].

WOtAN bietet eine mobile Prozess-Engine mit der Schwäche, dass es sich um eine native Applikation für Android handelt. Soll dieses Framework auch auf anderen Plattformen verwenden, wäre eine neue Implementierung für die jeweilige Plattform nötig.

3.7 Diskussion

In diesem Abschnitt werden die vorgestellten Arbeiten miteinander verglichen. Tabelle 3.1 liefert einen Vergleich der eben vorgestellten Arbeiten. Dabei werden unterstützen Plattformen, offline Ausführbarkeit, Prozessbeschreibung und die unterstützten Gateways (XOR, AND, LOOP) gegenübergestellt und verglichen.
Als erstes wurde die mobile Prozess-Engine Sliver vorgestellt. Sie basiert auf WS-BPEL und wurde speziell für mobile Geräte entwickelt [15]. Bei der Entwicklung wurde sich auf die Kernfunktionen von BPEL fokussiert, um die Applikation auf mobilen Geräten ausführen zu können. Es werden von Sliver konditionale Verzweigungen und parallele Ausführungen unterstützt. ROME4EU wurde speziell als Koordinations-Engine im Krisenfall für mobile Geräte entwickelt und kann mit einer instabilen Internetverbindung umgehen. ROME4EU basiert auf WS-BPEL und unterstützt Aktivitäten, konditionale Verzweigungen und parallele Ausführungen [17]. Der größte Schwachpunkt liegt beim Gerät des Team-Leiters. Fällt dieses Gerät aus oder besitzt es dauerhaft keine Verbindung, legt es das ganze Team lahm. MARPLE basiert im Gegensatz zu Sliver und ROME4EU auf ADEPT [18]. Allerdings werden die Prozesse bei MARPEL fragmentiert an die mobilen Geräte verteilt. Um MARPLE jedoch auf verschiedenen Plattformen ausführen zu können, sind, wie auch bei den anderen Projekten, jeweils plattformspezifische Implementierungen nötig. DEMAC ist eine Middleware zur Ausführung von verteilten Geschäftsprozessen, die auch mobil ausgeführt werden können. Dabei wird von DEMAC eine eigene Beschreibungssprache für die Prozesse verwendet. AllenaVita bietet eine mobile Prozess-Engine, die zur Prozessausführung jedoch eine Verbindung zum Internet benötigt. Zur Prozessbeschreibung verwendet AllenaVita BPMN 2.0. Die Applikation im Moment nur auf Android verfügbar. WOtAN ist ein Android-Framework, das eine offline Prozessausführung und Prozess-Analyse-Möglichkeiten bietet. Als Prozessbeschreibungssprache verwendet WOtAN ADEPT [28].

				Gateways		
Projekt	**Plattform**	**offline**	**Sprache**	**XOR**	**AND**	**LOOP**
Sliver	Java SE	✓	WS-BPEL	✓	✓	X
ROME4EU	Windows	X	WS-BPEL	✓	✓	✓
MARPLE	Android, iOS	✓	ADEPT	✓	✓	X
DEMAC	Java ME	✓	DPDL	✓	✓	✓
AllenaVita	Android	X	BPMN 2.0	✓	✓	✓
WOtAN	Android	✓	ADEPT	✓	✓	X

Tabelle 3.1: Vergleich der verwandten Arbeiten in Bezug auf deren Funktionen und deren Eigenschaften.

4

Anforderungen

In diesem Kapitel werden die Anforderungen an die zu entwickelnde mobile Applikation behandelt. Dabei werden zunächst die funktionalen Anforderungen und anschließend die nicht-funktionalen Anforderungen spezifiziert und erklärt.

4.1 Funktionale Anforderungen

In diesem Abschnitt werden die funktionalen Anforderungen besprochen.

FA1 (Import von Prozessmodellen): Die Applikation soll die Prozessmodelle, die über einen Webservice bezogen werden, importieren können.

FA2 (Ausführung der Prozessinstanzen): Die Ausführung eines Prozesses soll stets offline erfolgen können. Dabei soll keine aktive Internetverbindung benötigt werden.

FA3 (Protokollierung der Prozessausführung): Der Zustand der Prozessausführung soll bei jedem Zustandsübergang protokolliert werden.

FA4 (Verwendung von ausführbaren Geschäftsprozessen): Jeder der ausführbaren Geschäftsprozesse stellt ein individuelles Software-Modell dar, welches anhand der Verwendung entsprechend behandelt wird.

FA5 (Regelauswertung): Die im Prozessmodell angegebenen Entscheidungen sollen zur Laufzeit ausgewertet werden.

FA6 (Verwalten der Prozessinstanzen): Ein Benutzer soll die einzelnen Prozessinstanzen verwalten können.

FA7 (Hochladen der Daten mit Transportverschlüsselung): Damit die Daten einer Prozessinstanz vor Missbrauch geschützt sind, sollen sie während der Übertragung auf den Server verschlüsselt werden.

4.2 Nicht-funktionale Anforderungen

In diesem Abschnitt werden nicht-funktionale Anforderungen aufgestellt und erläutert.

NFA1 (Fehlermeldungen): Die Anwendung soll sinnvolle Fehlermeldungen ausgeben, sobald Fehler auftreten.

NFA2 (Programmiersprache): Die mobile Applikation soll mit der Programmiersprache TypeScript implementiert werden.

NFA3 (Ausführungsplattformen): Die mobile Applikation soll auf den Plattformen iOS, Android und Windows verfügbar sein.

NFA4 (Mehrsprachigkeit): Um die mobile Applikation für möglichst viele Benutzer benutzerfreundlich gestalten zu können, soll die mobile Applikation in den Sprachen Deutsch, Englisch, Italienisch, Französisch und Spanisch verfügbar sein.

NFA5 (Intuitive Bedienbarkeit): Die Applikation soll intuitiv bedienbar sein, ohne dass sich ein Benutzer in die Applikation einarbeiten muss.

NFA6 (Portierbarkeit der Prozess-Engine): Die Prozess-Engine soll sich einfach in andere Projekte beziehungsweise in andere Applikationen portieren lassen.

NFA7 (Sicherheit): Die Datensicherheit soll gewährleistet werden, sobald die Applikation Daten erhoben hat.

NFA8 (Erweiterbarkeit und Änderbarkeit): Damit die Anwendung einfach erweitert und abgeändert werden kann, soll sie möglichst modular aufgebaut sein.

In diesem Kapitel wurden die Anforderungen an die zu entwickelnde Applikation erläutert. Im folgenden Kapitel wird die Architektur der entwickelten mobilen Applikation vorgestellt.

5

Architektur

In diesem Kapitel wird ein Einblick in die konzeptionelle Architektur der Applikation gegeben, die im Zuge dieser Thesis entwickelt wurde. Zunächst wird die Architektur des Gesamtsystems näher betrachtet und die grundlegende Aufteilung in Komponenten erläutert. Anschließend werden die einzelnen Komponenten im Detail diskutiert. Den Anfang macht dabei die Prozess-Engine. Im nächsten Schritt wird in Abschnitt 5.3 das Applikationsmodul besprochen. Danach wird auf das Managementmodul in Abschnitt 5.4 eingegangen. Zu guter Letzt wird in Abschnitt 5.5 das Einstellungsmodul besprochen.

5.1 Gesamtsystem

Die gesamte Applikation besteht aus vier Teilen: der Prozess-Engine, dem Applikationsmodul, dem Managementmodul und dem Einstellungsmodul (siehe Abbildung 5.1).
Das Gesamtsystem basiert auf einer Dreischichtarchitektur (siehe Abbildung 5.2). Hierbei übernimmt stets ein Service das Bereitstellen der Daten. Die Logik wird von einer Komponente übernommen. Diese bereitet die Daten auf und stellt diese für die Präsentationsschicht bereit. Die Präsentationsschicht besitzt zusätzlich noch einen eigenen Logik-Teil. Dieser Teil ist nötig, damit für Daten, die in gleicher Weise präsentiert werden sollen, lediglich eine Präsentationsstruktur benötigt wird. Dies wird genauer in Kapitel 6.1.2 diskutiert. Beim Entwurf der Applikation wurde auf eine strikte Trennung der Komponenten (siehe **NFA8**) geachtet, sodass diese nicht voneinander abhängig sind. Es existieren jedoch logische Abhängigkeiten.
So ist beispielsweise die Management Komponente ohne die Prozess-Engine nutzlos, da über die Prozess-Engine die Prozessmodelle und die Prozessinstanzen verwaltet

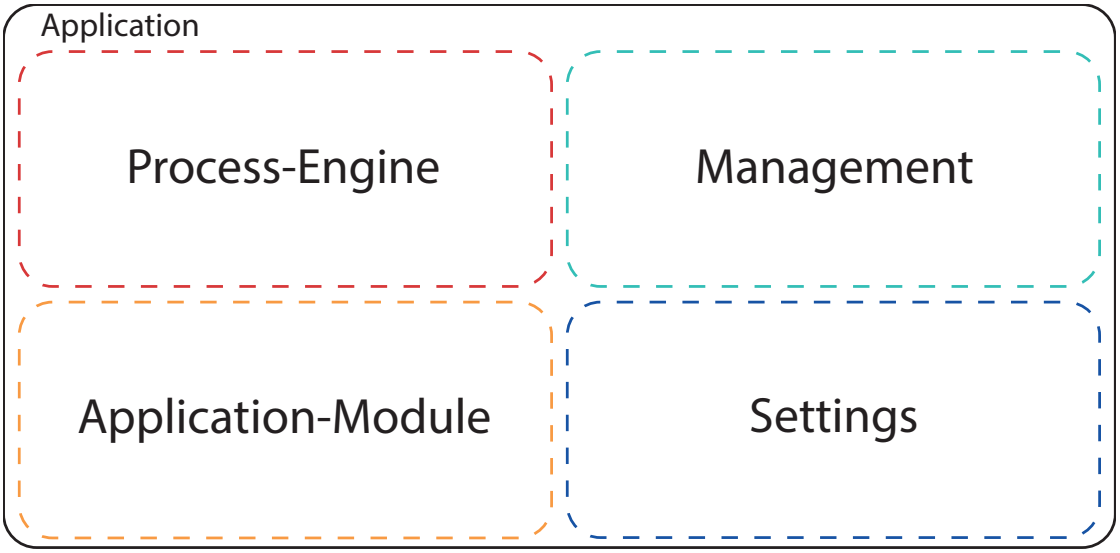

Abbildung 5.1: Architektur des Gesamtsystems.

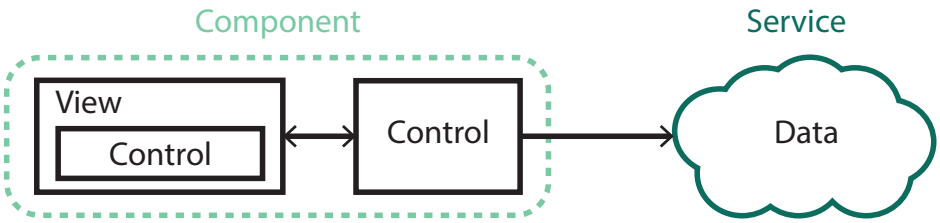

Abbildung 5.2: Dreischichtarchitektur der Applikation.

werden. Der Zugriff auf die Prozess-Daten erfolgt hierbei über die Prozess-Engine. Somit lässt sich die Prozess-Engine autark verwenden (siehe **NFA6**). Ähnlich ist die Situation auch zwischen dem Applikationsmodul und der Prozess-Engine. Hierbei stellt das Applikationsmodul Anwendungsbausteine für die Aktivitäten bereit.
Die Benutzeroberfläche der mobilen Applikation ist dabei intuitiv bedienbar, sodass sich der Benutzer nicht in die Applikation einarbeiten muss (siehe **NFA5**). Dazu ist die mobile Applikation in den Sprachen Deutsch, Englisch, Italienisch, Spanisch und Französisch verfügbar (siehe **NFA4**). Außerdem werden sinnvolle Fehlermeldungen ebenfalls mehrsprachig ausgegeben (siehe **NFA1**).

5.2 Prozess-Engine

In diesem Abschnitt wird die Prozess-Engine behandelt. Zunächst werden die Konzepte vorgestellt, die von der Prozess-Engine verwendet werden. Darauf aufbauend wird die Architektur der Prozess-Engine besprochen.

5.2.1 Konzepte

Im Folgenden werden die implementierten Konzepte vorgestellt. Zunächst wird hierbei das Prozessmodell und dessen Bestandteile erläutert. Im nächsten Schritt wird auf die Prozessausführung eingegangen. Der Fokus liegt dabei auf dem Knoten-Zustandsmodell. Der letzte Abschnitt befasst sich mit dem Konzept der ausführbaren Geschäftsprozesse, die im Folgenden als Executable Business Processes bezeichnet werden.

Prozessmodell

Ein Prozessmodell ist das Abbild eines realen Geschäftsprozesses (Beispiel: Abbildung 5.3). Dieses wird oft durch eine grafische Beschreibungssprache spezifiziert. Häufig sind diese Beschreibungssprachen blockstrukturiert. Das heißt, dass jeder Block eines solchen Prozessmodells einen Start und einen Endknoten besitzt [9]. Zudem dürfen sich diese Blöcke nicht überlappen, sodass zum Beispiel auf ein AND-Split kein XOR-Join folgen darf, sondern ein AND-Join folgen muss. Dieses Prinzip wird in diesem Zusammenhang als Wohlgeformtheit bezeichnet.

Ein Prozessmodell, auch Prozessschema genannt, das einen realen Geschäftsprozess abbildet, stellt einen gerichteten Graphen dar. Es besteht aus einer Menge von Knoten, die Aktivitäten und Kontrollkonnektoren (zum Beispiel AND Splits oder XOR Joins) repräsentieren [31]. Zudem existieren Kontrollflusskanten, die sich zwischen den Knoten befinden und die Knoten miteinander verbindet. Außerdem existieren Datenelemente, die durch einen Identifikator eindeutig identifiziert werden. Sie besitzen einen Datentyp, auf den lesend oder schreibend zugegriffen werden kann. Hierfür gibt es zudem spezielle Datenkanten, mit denen Lese bzw. Schreib-Zugriffe veranschaulicht werden können.

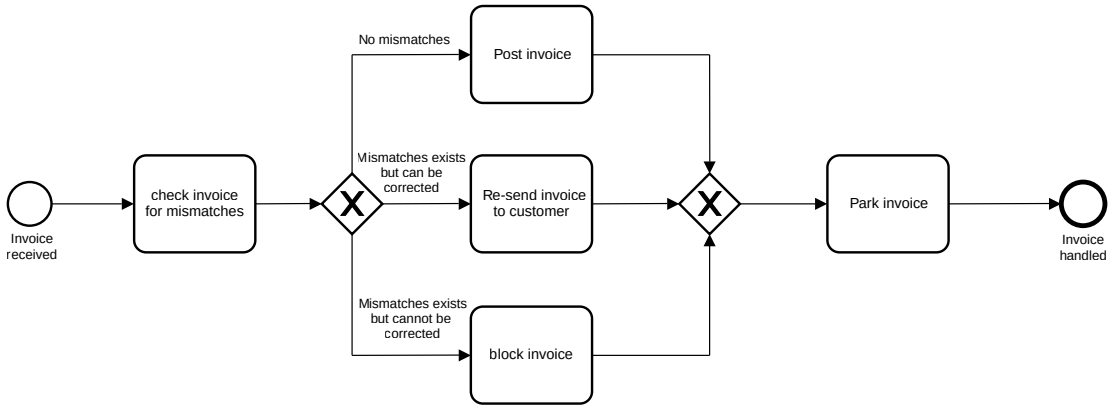

Abbildung 5.3: Ein Beispiel-Prozessmodell: Rechnungsprüfung (nach [30]).

Die Tabelle 5.1 zeigt eine Übersicht der möglichen Knoten-Typen. Dabei bleibt durch das Aufteilen von (AND, LOOP und XOR) in Split als auch Join die Blockbildung gewahrt. Hierfür kann der Split eines jeden Gateways als Einstiegspunkt betrachtet werden und der zugehörige Join als Ausstiegspunkt [31].

Prozessausführung

Damit ein Prozessmodell ausgeführt werden kann, wird zur Laufzeit eine Prozessinstanz erstellt. Diese Instanz wird anhand des zuvor definierten Prozessmodells ausgeführt [32]. Die Prozessausführung bezeichnet den Vorgang, bei dem der Prozessgraph Schritt für Schritt traversiert wird. Die unterschiedlichen Knotenzustände werden in Abbildung 5.4 dargestellt. Als Einstiegspunkt dient dafür der `START`-Knoten. Von ihm aus können die anderen Knoten bis zum `END`-Knoten erreicht werden. Der `END`-Knoten signalisiert das Ende des Prozesses und ist somit der Ausstiegspunkt.
Während der Prozessausführung durchlaufen Knoten verschiedene Zustände. Zunächst werden alle Knoten eines Prozesses mit dem Status `NOT_ACTIVATED` initialisiert. Ein Knoten wird in den Zustand `ACTIVATED` überführt, sobald alle Vorbedingungen erfüllt sind. Die Prozess-Engine kann Knoten automatisch starten und somit direkt in den Zustand `STARTED` überführen. Ist jedoch ein manueller Start durch den Benutzer notwendig, wechselt der Knoten zunächst in den Zustand `SELECTED`. An diesem Punkt

Knotentyp	Beschreibung
START	Signalisiert den Einstiegspunkt eines jeden Prozessmodells.
NORMAL	Stellt einen normalen Knoten dar. Er kann auch eine Aktivität enthalten.
AND_SPLIT	Stellt ein AND-Gateway dar. Die Prozess-Zweige werden dabei parallel ausgeführt.
AND_JOIN	Stellt ein AND-Gateway dar. Hier laufen die Prozess-Zweige nach einem AND_SPLIT wieder zusammen.
XOR_SPLIT	Stellt ein XOR-Gateway dar. Es enthält Bedingung(en) für die Auswahl des nächsten Prozessschrittes.
XOR_JOIN	Stellt ein XOR-Gateway dar. Hier laufen die Prozess-Zweige nach einem XOR_SPLIT wieder zusammen.
LOOP_START	Stellt ein Loop-Gateway dar. Hier ist der Wiedereinstiegspunkt für die Schleife.
LOOP_END	Stellt ein Loop-Gateway dar. Es enthält eine Bedingung nach der entschieden wird, ob die Schleife erneut durchlaufen wird.
END	Signalisiert den Aufstiegspunkt eines jeden Prozessmodells

Tabelle 5.1: Knotentypen und ihre Bedeutung.

wird auf den manuellen Start durch den Benutzer gewartet, bis dieser den Knoten explizit startet. Anschließend wird er in STARTED überführt.
Befindet sich ein Knoten im Zustand STARTED, existieren unterschiedliche Möglichkeiten für die weitere Bearbeitung. Unterbricht der Nutzer die Ausführung, wechselt der Knoten in den Zustand SUSPENDED. Durch einen erneuten Wechsel in den Zustand STARTED kann dieser weitergeführt werden. Sobald der Knoten ohne Fehler beendet wurde, wechselt er in den Zustand COMPLETED. Von dem Zustand COMPLETED aus kann der nächste Knoten aktiviert werden, sobald alle Vorbedingungen dieses Knotens erfüllt sind. Ist bei der Ausführung ein Fehler aufgetreten, wechselt er in den Zustand FAILED.

Executable Business Process

Ein Knoten kann einen Executable Business Process (EBP) enthalten, der eine Art Software-Modell darstellt. Ein EBP kann dabei wie eine Funktion Input- und Output-Parameter enthalten, die für lesenden beziehungsweise schreibenden Zugriff stehen. Ein EBP ist eine Aktivität, die automatisch oder durch den Benutzer ausgeführt wird.

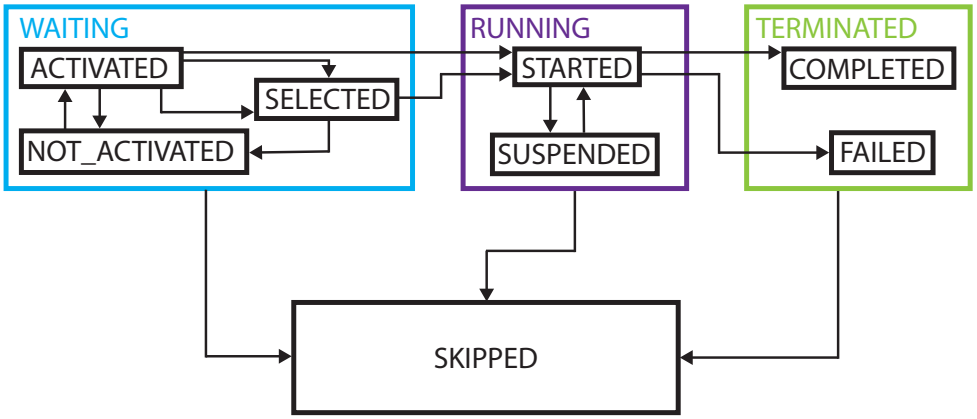

Abbildung 5.4: Zustands-Typen und die Übergänge zwischen den Zuständen (nach [9]).

5.2.2 Architektur

Die Abbildung 5.5 zeigt die Architektur der entwickelten Prozess-Engine. Dabei besitzt die Prozess-Engine drei Manager: den Execution Manager, den Instance Manager und den Runtime Manager. Zusammen kontrollieren sie eine Prozessinstanz. Zusätzlich zu den Managern existieren noch eine Reihe von Diensten: ein Datei-Speicher-Dienst, ein Kommunikations-Dienst, ein Antwort-Speicher und ein Datenbank-Dienst.

Execution Manager Zu den Aufgaben des Execution Manager gehört der Import von Prozessmodellen, das Speichern beziehungsweise Wiederherstellen einer angefangenen Prozessinstanz sowie der Zugriff auf die Funktionen des Instance Managers. Dabei besitzt der Execution Manager pro laufende Prozessinstanz einen Instance Manager. Deshalb muss er den Zugriff auf diese Instance Manager ebenfalls verwalten.

Instance Manager Der Instance Manager ist für den Kontrollfluss einer Instanz zuständig. Das heißt, er regelt die Ausführung der Prozessinstanz (siehe **FA2**). Dabei besitzt er genau einen Runtime Manager, der für ihn die Daten einer Prozessinstanz verwaltet. Hierfür ist er in der Lage die verschiedenen Knoten-Typen zu unterscheiden und diese entsprechend ihrer Semantik zu behandeln. Außerdem übernimmt der Instance Manager die Protokollierung der Knotenzustände (siehe **FA3**).

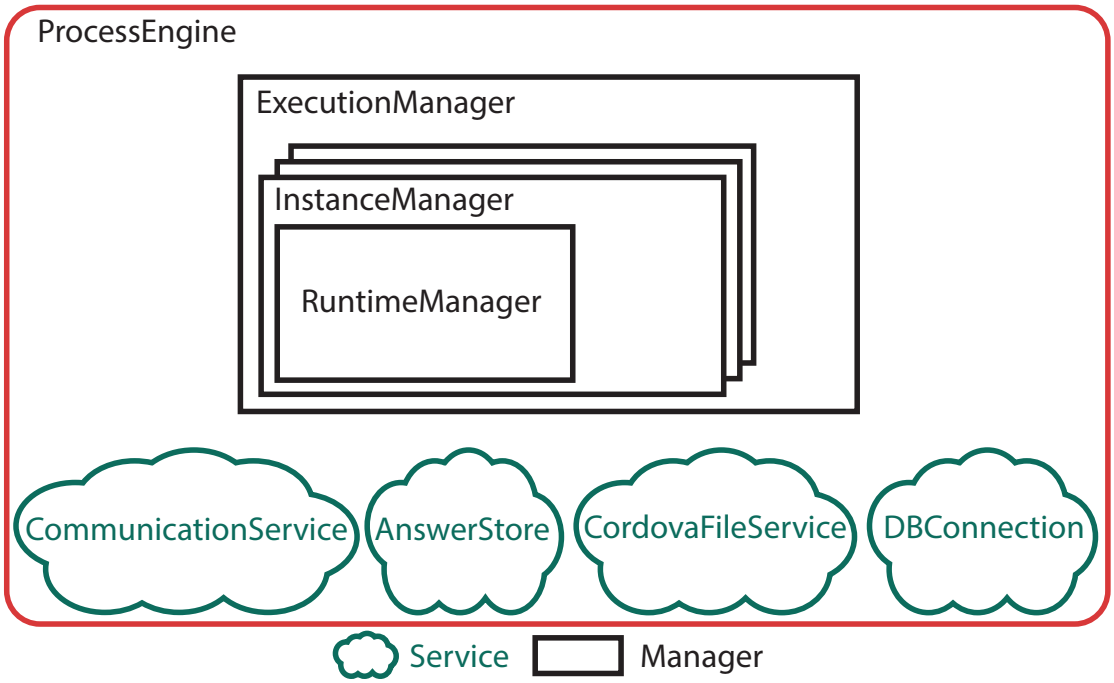

Abbildung 5.5: Architektur der Prozess-Engine.

Runtime Manager Der Runtime Manager ist zuständig für den Datenfluss innerhalb der Prozessinstanz. Er hält die angefallenen Daten und bietet Methoden für den Zugriff an. Außerdem regelt er die bidirektionale Kommunikation von den Managern zum Applikationsmodul. Er sendet also den Executable Business Process (siehe **FA4**) an den Kommunkations-Dienst, der den Executable Business Process an das Applikationsmodul weiterleitet. Dazu behandelt der Runtime Manager die durch den Kommunikations-Dienst empfangenen Daten indem diese im Antworten-Speicher abgelegt und gespeichert werden. Eine weitere Aufgabe des Runtime Manager ist der Zugriff auf die Datenbank. Hierfür hält er eine Referenz auf den Datenbank-Dienst. Außerdem besitzt jeder Runtime Manager einen eigenen Antworten-Speicher, damit die Antwort-Daten, die über den Kommunikationsdienst vom Applikationsmodul empfangen wurden, vor Zugriff von außen geschützt sind (siehe **NFA7**).

Dienste

In diesem Abschnitt wird der jeweilige Aufgabenbereich der Dienste näher erläutert. Die Dienste stellen Zugriffsmethoden auf das Dateisystem, den Kommunikationskanal oder auf die Datenbank bereit. Außerdem hält ein Dienst die benötigten Daten.

Datei-Speicher-Dienst: Dieser Dienst bietet Zugriff auf das Dateisystem der verschiedenen Plattformen. Er wird benötigt, wenn ein Prozessmodell Referenzen auf Media-Dateien besitzt. Diese werden zusammen mit dem Prozessmodell importiert und auf dem Dateisystem der jeweiligen Plattform gespeichert.

Kommunikations-Dienst: Der Kommunikations-Dienst stellt einen Kommunikationsbus dar. Er übernimmt die Weiterleitung der Executable Business Processe an den jeweiligen Dienst oder an den Applikations-Komponenten Manager des Applikationsmoduls. Der Kommunikations-Dienst regelt also die Kommunikation zwischen dem Runtime Manager und den Anwendungsbausteinen.

Antworten-Speicher: Dieser Dienst verwaltet die Daten, die von der Benutzeroberfläche über den Kommunikationsdienst an die Prozess-Engine gesendet werden.

Datenbank-Dienst: Dieser Dienst stellt Methoden für den Zugriff auf eine SQLite Datenbank bereit.

Da an diesem Punkt alle Komponenten der Prozess-Engine erläutert wurden, wird im folgenden Abschnitt auf das Applikationsmodul eingegangen.

5.3 Applikationsmodul

In diesem Abschnitt wird die Architektur des Applikationsmoduls näher behandelt. Im Prinzip besteht sie aus zwei Komponenten: dem Applikations-Komponenten Manager und den Anwendungsbausteinen (siehe Abbildung 5.6).

Applikations-Komponenten Manager: Prinzipiell besitzt der Applikations-Komponenten Manager zwei Aufgaben. Zum einen behandelt er die über den Kommunikations-Dienst erhaltenen Daten. Diese werden anhand der `ImplementationClass`-Eigenschaft des Executable Business Process an den entsprechenden Anwendung-

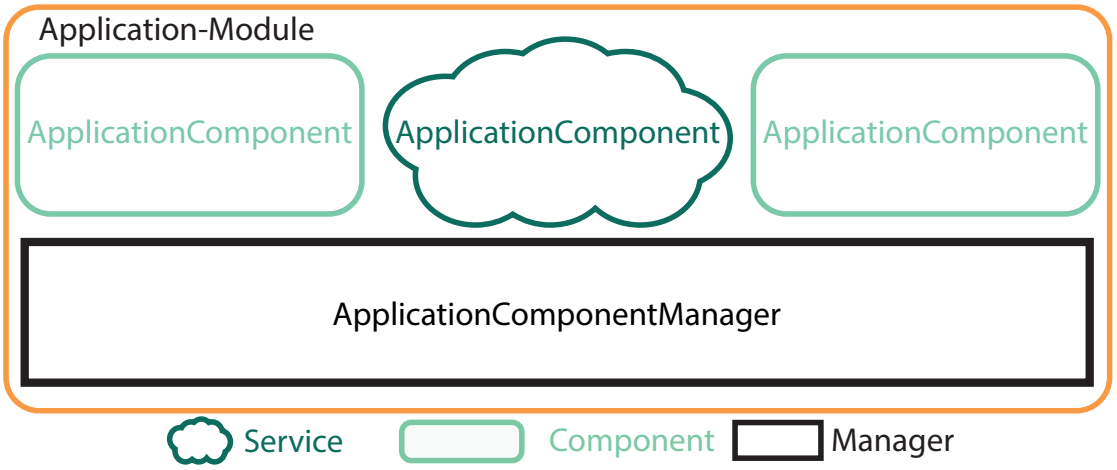

Abbildung 5.6: Architektur des Applikationsmoduls.

sbaustein weitergeleitet. Zum anderen kann er Antwort-Daten an den Runtime Manager senden. Hierfür wird das von dem Anwendungsbaustein erhaltene Objekt zur Verwaltung über den Kommunikations-Dienst an den Runtime Manager gesendet, der diese Antwort-Objekte in seinem Antworten-Speicher verwaltet. So wird dafür gesorgt, dass nicht jede andere Komponente aus diesem Modul direkten Zugriff auf die angefallenen Prozessdaten besitzt.

Anwendungsbaustein: Ein Anwendungsbaustein stellt eine Anwendungskomponente für einen Executable Business Process dar. Ein Baustein kann sämtliche Funktionen übernehmen. So kann ein Anwendungsbaustein eine Benutzeroberfläche implementieren oder einen automatisierten Dienst bereitstellen. Jedenfalls ist der Anwendungsbaustein vollständig von der Prozess-Engine entkoppelt, da sie nur über den Kommunikations-Dienst der Prozess-Engine kommunizieren.

5.4 Managementmodul

Mit dem Managementmodul lassen sich administrative Aufgaben bewältigen. Dabei besteht sie aus drei Komponenten: der Download-Komponente, der Administration-View-Komponente und dem Upload-Dienst (siehe Abbildung 5.7). Im folgenden Abschnitt werden diese Aufgaben nun detailliert spezifiziert.

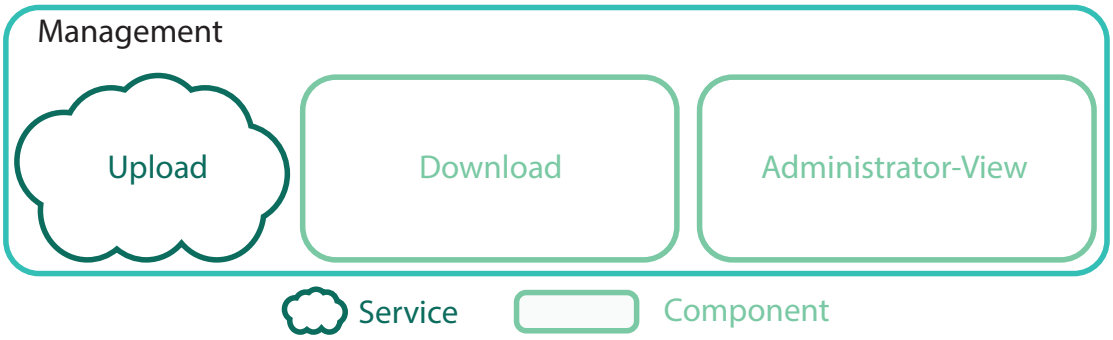

Abbildung 5.7: Architektur des Managementmoduls.

Download: Mit dieser Komponente lassen sich Prozessmodelle in die Applikation importieren (siehe **FA1**). Dabei werden diese von einem Server heruntergeladen und über die Prozess-Engine persistent gespeichert.

Administration-View: Mit der Administration-View-Komponente lassen sich Prozessmodelle und die bereits gestarteten bzw. abgeschlossenen Prozessinstanzen verwalten (siehe **FA6**). Dabei kann ein Prozessmodell gelöscht werden, wenn es nicht mehr benötigt wird. Für die Prozessinstanzen gibt es zwei Verwaltungsoptionen. Zum einen können sie gelöscht werden, zum anderen können die Daten, die bei der Prozessausführung gesammelt wurden, auf einen Server geladen werden. Diese Aufgabe übernimmt der Upload-Dienst.

Upload-Dienst: Der Upload-Dienst ist zuständig für das Hochladen der durch die Prozessausführung angefallenen Daten auf einen Server. Damit die Daten, die übertragen werden sollen, vor Missbrauch geschützt sind, überträgt der Upload Dienst diese Daten mittels Transportverschlüsselung (siehe **FA7**).

5.5 Einstellungsmodul

In diesem Abschnitt wird nun das Einstellungsmodul behandelt (siehe Abbildung 5.8). Sie besteht lediglich aus der Komponente Einstellungen.

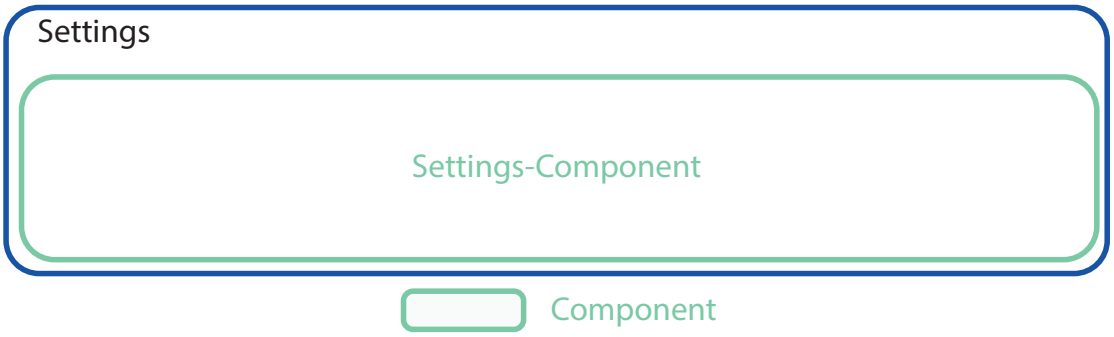

Abbildung 5.8: Architektur der Einstellungs-Komponente.

Einstellungen: Die Komponente Einstellungen verwaltet die Systemeinstellungen. Dazu gehören beispielsweise die Applikationssprache oder die Daten eines Benutzers. Diese können von anderen Diensten verwendet werden.

In diesem Kapitel wurde die Architektur der Applikation behandelt. Diese besteht aus vier Komponenten: der Prozess-Engine, dem Applikationsmodul, dem Managementmodul und dem Einstellungsmodul. Daraufhin wurde die Architektur jedes dieser Module im Detail behandelt. Im nächsten Kapitel wird die besprochene Architektur verwendet um die Applikation modular aufzubauen und schlussendlich zu implementieren.

6

Implementierung

In diesem Kapitel werden wichtige Aspekte der verwendeten Frameworks und interessante Details der Implementierungen vorgestellt und anschließend diskutiert. Zunächst werden die Frameworks, die bei der Implementierung eine wichtige Rolle spielen, in Abschnitt 6.1 vorgestellt und eingeordnet. Danach werden interessante Details der Implementierung präsentiert. Zu guter Letzt wird in Abschnitt 6.3 diskutiert, inwiefern sich die verwendeten Technologien für eine solche Implementierung eignen bzw. welche Kompromisse eingegangen werden müssen.

6.1 Verwendete Technologien

Um eine große Abdeckung der unterschiedlichen Plattformen durch eine Implementierung erreichen zu können, werden verschiedene Frameworks verwendet. Diese werden im folgenden Abschnitt mit samt ihrer Funktionen vorgestellt. Es wird zudem eingeordnet, welche Funktion jedes Framework im Gesamten übernimmt.

6.1.1 Apache Cordova

Apache Cordova ist ein Open-Source Framework zur Entwicklung hybrider Applikationen für mobile Endgeräte. Mit Cordova lassen sich Applikationen mithilfe von HTML5, CSS und JavaScript erstellen [6]. Diese Webtechnologien werden anstatt der plattformspezifischen Sprachen, wie zum Beispiel Objective C für iOS oder Java für Android, verwendet. Die Applikationen, die mithilfe von Cordova erstellt wurden, werden hybrid

genannt (siehe Abschnitt 2.1). Hybrid aus dem Grund, da sie nicht die plattformspezifischen Benutzeroberflächen Frameworks verwenden. Sie verwenden stattdessen HTML5 und CSS für die Benutzerschnittstelle. Zudem sind es keine reinen Webapplikationen, da die Distribution der Applikationen über die jeweiligen App-Stores verläuft. Im Moment unterstützt Cordova folgende Plattformen: iOS, Android, Windows (Phone), FireOs, LGwebOS, Ubuntu und Firefox OS. Über die Cordova-Native-Plugins lassen sich zudem JavaScript Schnittstellen, für die native Funktionen schaffen. Dadurch ist dann auch ein Zugriff nativen Komponenten möglich.

6.1.2 Angular 2

Angular 2, im Folgenden Angular gennant, ist ein Open-Source-Framework mit dem sich Single-Page-Anwendungen erstellen lassen. Es verwendet dabei das Model-View-ViewModel (MVVM) Konzept, welches eine Abwandlung des Model-View-Controller (MVC) Konzept darstellt. Hierbei werden die Benutzerschnittstelle und das Modell in einem gemeinsamen Controller definiert. Sie verfügen über eine bidirektionale Anbindung, sodass Interaktionen sowohl von der Benutzerschnittstelle als auch von der Modell-Seite aus möglich sind. Die Benutzerschnittstelle ist dabei ein HTML-Modell, welches zusätzliche Angular-Annotationen besitzt. Diese Annotationen weisen Angular an, wie die Oberfläche gerendert werden muss. Angular, in der Version 1, ist nur in JavaScript verfügbar. Die Version 2 bietet hingegen Unterstützung für Dart, JavaScript und TypeScript. Im Folgenden wird jedoch nur Angular 2 in der TypeScript-Variante (siehe **NFA2**) betrachtet, da diese Variante in dieser Arbeit Verwendung findet.

Architektur

Mit der Entwicklung von Angular 2 wurde die Version 1 nicht nur weiterentwickelt, das Framework wurde stattdessen komplett neu entwickelt. Deshalb wird im Folgenden nur auf die Architektur von Angular 2 eingegangen.

Module Angular ist modular aufgebaut. Jeder Teil der Implementierung ist ein sogenanntes Modul [33]. Ein Modul exportiert typischerweise eine Klasse. Diese Modularisierung bietet den Vorteil bei der Einbindung in andere Module, da sie über `import` in andere Module importiert werden können. Die häufigsten Module sind Komponenten.

Modelle, Komponenten und Metadaten Die Abbildung 6.1 zeigt den Aufbau und das Zusammenspiel zwischen dem Modell, den Metadaten und der Komponente. Dabei wird in 1. auf das Modell und in 2. die Metadaten eingegangen. Anschließend wird sich in 3. mit der Funktion der Komponente auseinandergesetzt.

1. : Ein Modell ist die Rohform der Benutzeroberfläche. Sie enthält HTML-Elemente und Angular Annotationen. Diese Annotationen haben ihre eigene Syntax (Angular Template Syntax). Sie beinhaltet die Datenbindung des HTML-Modells mit der Komponente und die Direktiven-Anbindung. Durch die Auswertung der Angular-Annotationen erhält Angular die Informationen die nötig sind, um die Benutzeroberfläche richtig darzustellen. Das Modell kann auch Kind-Komponenten enthalten. Diese werden über ein eigenes HTML-Element eingebunden.

2. : Damit eine Komponente weiß, welches Modell sie kontrolliert, welche Services und welche Kind-Komponenten in ihr enthalten sind, benötigt sie Metadaten. Zu diesen Metadaten gehört zum Beispiel der Pfad zum kontrollierten Modell. Neben dieser Angabe können auch neue Services erstellt werden. Sie müssen mit `providers` als Array angegeben werden. Da eine Komponente auch Kind-Komponenten enthalten kann, müssen diese explizit angegeben werden. Dies geschieht über `directives`. Sie müssen ebenfalls als Array angegeben werden. Am wichtigsten jedoch ist der Selektor, welcher mit dem Schlüsselwort `selector` angegeben wird. Der Selektor ist die Angabe für Angular, welches HTML-Element im übergeordneten Modell durch das Modell der Komponente ersetzt werden soll.

3. : Eine Komponente ist das Kontroll-Element eines Modells. Sie beinhaltet lediglich die Logik, jedoch keine Benutzeroberfläche.

Da Modell, Metadaten und Komponenten in dieser Arbeit immer im Verbund auftreten, wird dieser Verbund im Folgenden schlicht als Komponente bezeichnet.

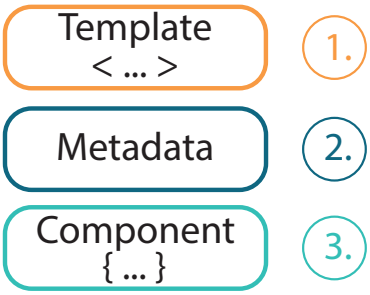

Abbildung 6.1: Aufbau einer vollständigen Komponente (nach [33]).

Datenbindung

Wie bereits angesprochen, ist es möglich die Komponente und das Modell über die Angular Template Syntax bidirektional anzubinden [33]. Jedoch ist die bidirektionale Anbindung nur eine von vier Möglichkeiten. Zudem existieren noch die Interpolation, die Eigenschafts-Bindung und die Ereignis-Bindung. Jede dieser Anbindungen hat einen bestimmten Verwendungszweck, der im Folgenden näher erläutert wird.

Zwei-Wege-Datenbindung: Eine bidirektionale Datenbindung wird in Angular Zwei-Wege-Datenbindung genannt [33]. Diese wird folgendermaßen angegeben: `<input [(ngModel)]= "hero">`. `NgModel` ist hier eine Besonderheit und wird gerne im Kontext von Input-Elementen verwendet, da hier bei einer Wertveränderung durch den Benutzer ein Ereignis ausgelöst wird. Jedoch kann auch eine programmatische Wertveränderung erfolgen, die sofort den geänderten Wert auf der Benutzeroberfläche sichtbar macht.

Interpolation: Die Interpolation ist die einfachste Variante um Variablen einer Komponente an die Benutzeroberfläche zu binden. Sie ermöglicht es, den Inhalt einer Variable im HTML-Modell anzuzeigen. Die Variable wird mit zwei geschweiften Klammern umschlossen `{{ name }}` . Die Variable `name` kann nur durch den Logikteil der Komponente verändert werden.

Eigenschafts-Bindung: Die nächste Variante ist die sogenannte Eigenschafts-Bindung [33]. Hierbei fungieren eckige Klammern als Identifikator. Sie wird meist zur Übergabe von Eigenschaften von der Vater-Komponente an die Kind-Komponente

verwendet. Dies sieht folgendermaßen aus: `<question [data]= "data">`. Im Beispiel erhält die Kind-Komponente `question` die der Variable `data`. Diese befindet sich in den eckigen Klammern. Sie bekommt den Inhalt der in der übergeordneten Komponente befindlichen Variable `data` übergeben.

Ereignis-Bindung: Zuletzt gibt es die Möglichkeit ganz gezielt spezielle Ereignisse mit einer Behandlungsmethode zu verknüpfen. Dies lässt sich wie folgt bewerkstelligen:
`<button (click)= "clickHandler()"></button>`
Bei jedem Klick auf den Button wird nun von Angular die Behandlungsmethode `clickHandler` aufgerufen, die die Behandlung ausführt. An diese Behandlungsmethode können auch Parameter übergeben werden.

Direktiven Ein wichtiger Bestandteil von Angular sind Direktiven [33]. Dabei wird zwischen strukturellen und Attribut-Direktiven unterschieden:

Strukturelle Direktiven: verändern das Document Object Model (DOM) der entsprechenden Seite. Hierzu zählen zum Beispiel `*ngFor` und `*ngIf`. Wie in Abbildung 6.2 illustriert wird, fügt `*ngFor` alle Elemente des Arrays `items` dynamisch zum DOM hinzu. Bei `*ngIf` wird der boolesche Ausdruck ausgewertet und wird erst dann zum DOM hinzugefügt, wenn die Bedingung als `true` evaluiert wurde.
`<p *ngIf="condition===42">is shown if condition is 42</p>`
In diesem Fall wird der Paragraph nur zum Document Object Model hinzugefügt wenn die Variable `condition` den Wert `42` enthält.

Attribut-Direktiven: fügen dynamisch Eigenschaften hinzu oder entfernen diese. Als Beispiel kann `ngStyle` genannt werden. `<div [style.color]="'orange'">`
Wie das Beispiel hier zeigt, kann jedes CSS-Style-Attribut eines jeden Elements verändert werden, indem die Attribut-Direktive `style` den Zugriff darauf ermöglicht.

Services Der nächste wichtige Angular-Bestandteil nennt sich Service [33]. Dabei kann ein Service so gut wie jede Funktion besitzen. Es kann sich um elementare Funktionen, wie einen einfache globale Variable oder eine zu berechnende Funktion handeln.

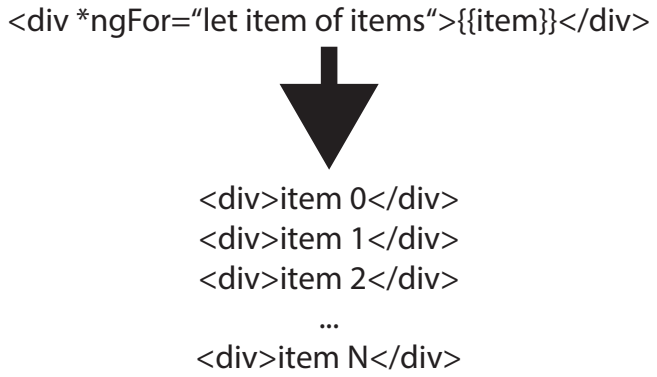

Abbildung 6.2: Beispielhafte Funktionsweise von `*ngFor`.

Der Service erfüllt jedenfalls immer einen wohldefinierten Zweck. Zum Beispiel wird eine Datenbank-Anbindung typischerweise in einen Service ausgelagert. Ein Service wird immer mit `@Injectable()` vor dem eigentlichen Modul markiert.

Abhängigkeitsinjektion Um einen Service in einer Komponente oder auch einem Service nutzen zu können, muss dieser zunächst injiziert werden [33]. Dafür benötigt jeder Service einen Provider. Dieser Provider ist in den meisten Fällen das Service-Modul selbst. Nun muss der Provider noch registriert werden. Dies kann in jeder Stelle der Applikation erfolgen. Oft geschieht dies direkt beim Laden der Anwendung, kann aber auch auf Komponenten-Ebene oder direkt in einer Methode geschehen. An welcher Stelle dies genau geschieht, ist abhängig von der gewünschten Nutzbarkeit.

Wird wie hier der Service `ExampleService` auf Komponenten-Ebene über die Metadaten mit dem Schlüsselwort `providers` registriert (siehe Listing 6.1 (1.)), so ist er für die Komponente und ihre Kind-Komponenten verfügbar und lässt sich über den Konstruktor injizieren. Dabei muss in TypeScript stets der Typ des jeweiligen Service angegeben werden, damit der Service identifiziert werden kann (siehe Listing 6.1 (2.)). Benötigt ein Service einen anderen Service, wird er mit einer Methode registriert. Hierfürl stellt Angular den `ReflectiveInjector` bereit, der den Service erstellt und registriert.

```
@Component({
  template: '<ion-nav [root]="rootPage"></ion-nav>',
  providers: [ExampleService],          // 1.
})
export class MyApp {
          constructor(private exampleService:ExampleService) { // 1.
                 let injector = ReflectiveInjector.resolveAndCreate([
                    ExampleService2]);     // 2.
                this.exampleService2 = injector.get(ExampleService2);
          }
}
```

Listing 6.1: Registrierung und Injektion eines Services auf Komponenten-Ebene.

Komponenten Lebenszyklen Jede Komponente, die in Angular zur Ausführung kommt, ist zustandsbehaftet. Diese Zustände nennen sich Lebenszyklus. Mit ihnen lassen sich Zustände abwarten, auf die reagiert werden soll. Die folgende Auflistung berücksichtigt bereits die Reihenfolge,in der die Zustände erreicht werden. [33, Lifecycle Hooks]:

ngOnChanges: wird noch vor ngOnInit ausgeführt, wenn eine Input-Bindung definiert wurde. Andernfalls wird ngOnChanges das erste Mal übersprungen.

ngOnInit: wird ausgeführt, wenn die Komponente initialisiert wird.

ngDoCheck: wird bei einer Änderung ausgeführt. Zum Beispiel im Fall, dass der Wert einer Variable geändert wurde.

ngAfterContentInit: wird ausgeführt, nachdem der Inhalt initialisiert wurde.

ngAfterContentChecked: wird ausgeführt, nachdem der Inhalt überprüft wurde.

ngAfterViewInit: wird ausgeführt, nachdem die Oberfläche initialisiert wurde.

ngAfterViewChecked: wir ausgeführt, nachdem die Oberfläche überprüft wurde.

ngOnDestroy: wird ausgeführt, bevor die Komponente zerstört wird.

Die Abbildung 6.3 liefert einen Gesamtüberblick der Angular Architektur. Besonders gut ersichtlich ist das Zusammenspiel des Modells, das über die Eigenschafts-Bindung

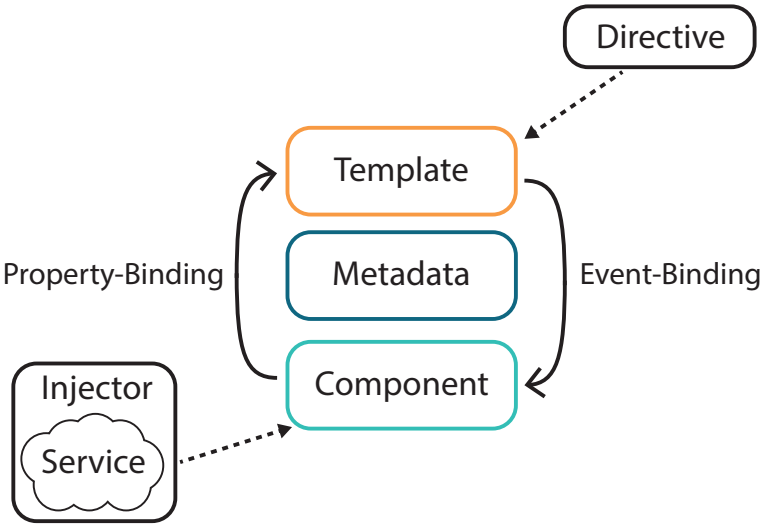

Abbildung 6.3: Übersicht der vorgestellten Angular 2 Architektur (nach [33]).

und Ereignis-Bindung an die Komponente gebunden ist. Zudem zeigt die Abbildung die Verbindung zwischen den Direktiven und dem Modell. Außerdem wird die Verbindung der Services mit der Komponente dargestellt.

6.1.3 Ionic 2

Ionic ist ein hybrides open-source Webframework zur Erstellung von Applikationen mithilfe von Webtechnologien. Es verwendet dafür HTML5, CSS, SASS und TypeScript. Dabei basiert es auf Angular 2 sowie auf Apache Cordova und unterstützt die Plattformen iOS, Android und Windows (siehe **NFA3**). Wie Abbildung 6.4 zeigt, besteht Ionic aus drei Säulen: den Ionic Komponenten, der Ionic Native-Unterstützung und des Ionic CLI [34]. Derweil setzt Ionic am Angular Komponenten Konzept an und bringt eine große Menge von vorgefertigten Oberflächen-Komponenten mit. Diese reichen dabei von einfachen Komponenten wie Buttons, Input-Feldern oder Slidern bis hin zu komplexen Suchfunktionen oder Komponenten zur Strukturierung von Oberflächen. Zudem besitzt Ionic noch eine Native-Unterstützung, die dabei eine Sammlung verschiedener JavaScript/TypeScript Wrapper zur Verfügung stellt. Dabei können SQLite Datenbanken ebenso verwendet werden, wie zum Beispiel die Kamera eines mobilen Gerätes oder die direkte Integration

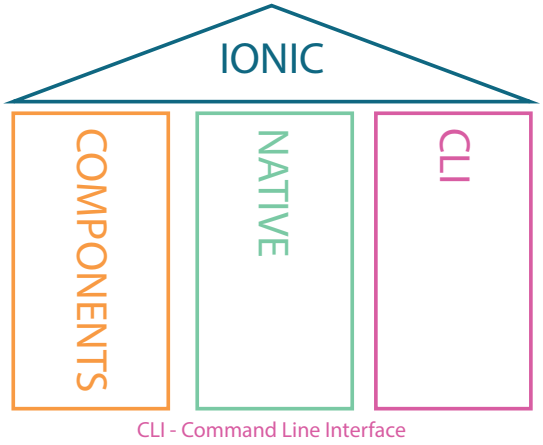

Abbildung 6.4: Übersicht der Ionic Architektur.

von Google Maps oder eine Facebook-Anbindung. Das Ionic CLI bildet dabei die Schnittstelle für die Kommandozeile. Sie bietet alle Funktionen, die zur Entwicklung einer Ionic Applikation nötig sind. Mit ihrer Hilfe lässt sich ein neues Projekt anlegen, dieses im Browser testen und die plattformspezifische Applikation bauen, sodass diese bereit ist, um in einen App-Store eingestellt zu werden.

Ionic Native

Mit Ionic Native werden Wrapper für Cordova bzw. PhoneGap Plugins bereitgestellt. Dies ermöglicht eine sehr einfache Integration der nativen Komponenten. Dabei liefert jeder dieser Wrapper ein sogenanntes `Promise` oder `Observable` zurück [34, Ionic Native], mit dem dann die entsprechende Komponente angesprochen werden kann. Damit ein solches Plugin verwendet werden kann, muss dies logischerweise installiert werden. Dies wird über den Packagemanager „npm“ bewerkstelligt. Ist ein Plugin installiert, muss es nur noch zu dem entsprechenden Projekt hinzugefügt werden. Dies ist über das CLI mit dem Befehl `ionic plugin add cordova-plugin` möglich.

Ionic CLI

Ionic vergleicht gerne das Command-Line-Interface als Schweizer Taschenmesser, da es alle Funktionen mitbringt, um eine Applikation zu entwickeln [34, Ionic CLI]. Dabei kann das CLI beim Projektstart und beim Browser-Test helfen. Außerdem kann mit dem CLI auch ein plattformspezifisches Projekt generiert werden, für iOS beispielsweise ein XCode Projekt. Zudem kann das CLI auch den plattformspezifischen Emulator laden, um die mobile Applikation auf diesem zu testen. Des Weiteren können über das CLI Plugins zu einem Projekt hinzugefügt werden.

6.1.4 Zusammenspiel der Frameworks

Wie in Abbildung 6.5 illustriert wird, werden alle oben genannten Frameworks im Verbund verwendet. Dabei wird die Benutzerschnittstelle mit Ionic erstellt. Die Logik der Applikation wird fast ausschließlich mithilfe von Angular bereitgestellt. Lediglich die Schnittstellen der nativen Funktionen werden von Ionic zur Verfügung gestellt.
Durch diese beiden Frameworks wird eine One-Page Applikation erstellt. Diese wird mithilfe von Cordova anschließend auf die plattformspezifischen Formate übersetzt. Als Resultat erhält man, für jede unterstütze Plattform, eine mobile Applikation, die in die diversen App-Stores eingestellt werden kann.

In diesem Abschnitt wurden die in der Implementierung verwendeten Frameworks vorgestellt. Im Folgenden wird näher auf die Implementierung eingegangen. Hierbei spielt die Logik, die in TypeScript geschrieben wurde, die Hauptrolle.

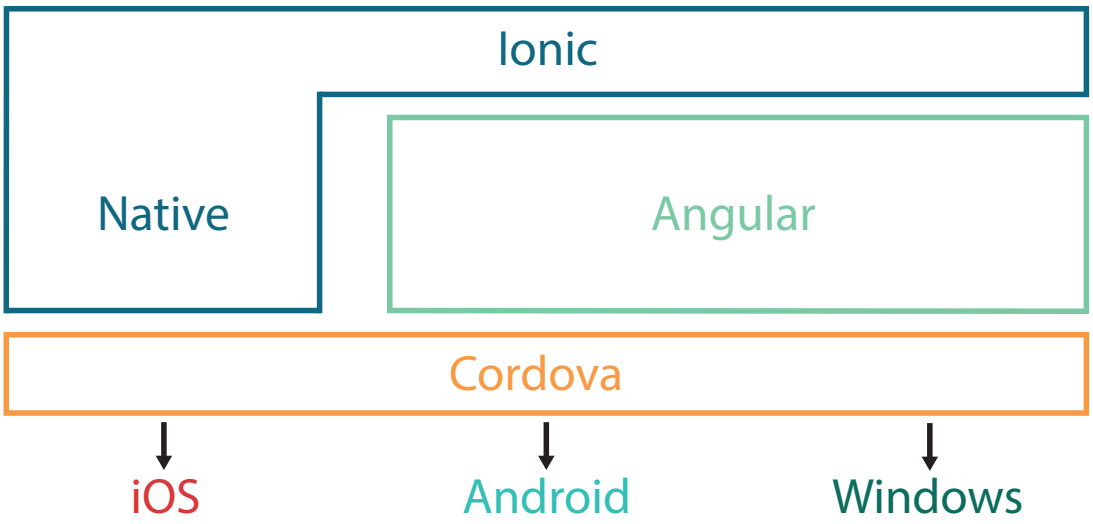

Abbildung 6.5: Zusammenspiel der vorgestellten Frameworks.

6.2 Ausgewählte Aspekte der Implementierung

In diesem Abschnitt werden ausgewählte Aspekte der Implementierung anhand von Anwendungsfällen vorgestellt. Zunächst wird in Abschnitt 6.2.1 ein Prozessmodell importiert. Anschließend wird das importierte Prozessmodell in Abschnitt 6.2.2 ausgeführt. Dabei werden zwei Anwendungsfälle demonstriert. Zunächst wird ein Knoten behandelt, der einen Executable Business Process besitzt, der eine Benutzeroberfläche repräsentiert. Im nächsten Schritt wird ein Knoten behandelt, der einen Executable Business Process enthält, der eine automatische Funktion darstellt.

6.2.1 Prozessmodell importieren

In diesem Szenario (siehe Abbildung 6.6) wird vom Benutzer ein Prozess-Modell aus dem Internet heruntergeladen. Anschließend wird das Prozess-Modell importiert, um danach mit der Prozess-Engine ausgeführt zu werden. Da ein Prozess-Modell zusätzlich Medien-Dateien benötigen kann, müssen diese Dateien ebenfalls importiert werden. Sie sind gegebenenfalls vorhanden und müssen über das Dateisystem der jeweiligen

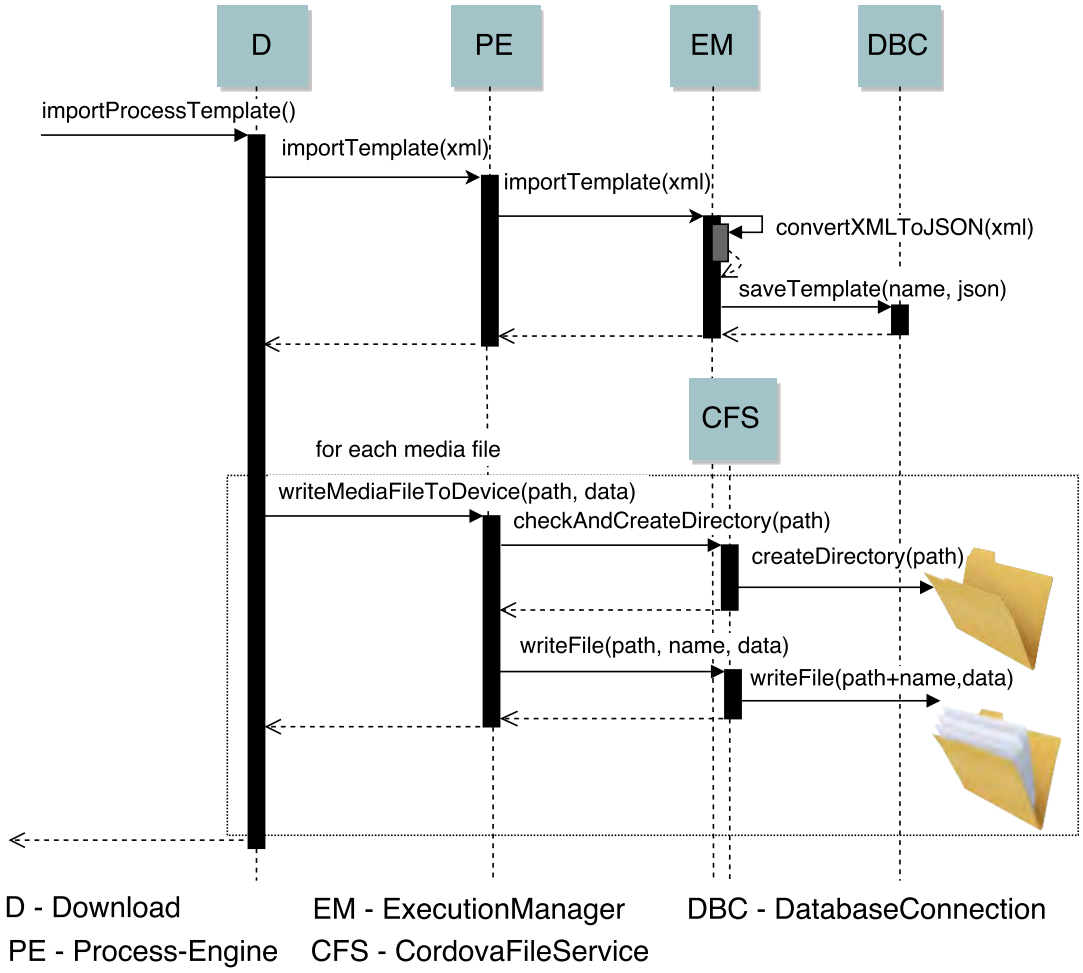

Abbildung 6.6: Anwendungsfall: Prozessmodell herunterladen und importieren.

Plattform auf dem Gerät gespeichert werden. Hierfür verwendet die Prozess-Engine einen Service, der den Zugriff auf das Dateisystem bereitstellt.

Grundlagen

Der Service, der den Zugriff auf das Dateisystem bereitstellt, basiert auf dem Cordova File Plugin [35]. Es ermöglicht den Lese- und Schreibzugriff auf das plattformindividuelle Dateisystem. Dabei basiert dieses Plugin auf der HTML5 File API und der FileWriter Spezifikation [35]. Um einen standardisierten Zugriff auf das Dateisystem bieten zu können, fungiert es als Schicht zwischen der Web-Oberfläche und dem Betriebssystem

des Gerätes und stellt die globale Variable `cordova.file.*` bereit. Mit ihr kann auf die betriebssystemunabhängigen Dateipfade zugegriffen werden.
Während der Implementierung war der TypeScript Wrapper, den Ionic normalerweise für den Zugriff auf native Plugins zur Verfügung stellt, noch nicht vollständig implementiert. Es fehlten einige Implementierungen für den Schreibzugriff. Die Methoden für den Schreibzugriff werden deshalb durch den `CordovaFileService` implementiert. Die Implementierung dieser Funktionen setzt ein detaillierteres Wissen über die Dateisysteme der unterschiedlichen Plattformen voraus. Im Folgenden wird eine kurze Einführung über die verschiedenen Dateisysteme gegeben.

iOS unterscheidet zwischen temporären, Cache und normalen Dateien. Temporäre Dateien können vom Betriebssystem gelöscht werden, wenn sie nicht mehr benötigt werden. Sie werden unter `/var/mobile/Applications/<UUID>/tmp/` gespeichert. Cache Dateien werden bei Applikations- oder Betriebssystem-Updates gelöscht. Sie werden unter `/var/mobile/Applications/<UUID>/Library/Caches/` gespeichert. Bei normalen Dateien wird zudem noch eine weitere Unterscheidung getroffen, ob diese Daten in der Cloud gespeichert werden sollen. Sollen sie in der Cloud gespeichert werden, müssen die Daten in das Synchronisations-Verzeichnis welches sich unter `/var/mobile/Applications/<UUID>/Library/Cloud/` befindet, gespeichert werden. Dieser Ordner dient zur Synchronisation mit dem Cloud-Dienst. Der iOS Cloud Dienst erkennt neue Dateien und spiegelt diese in die Cloud. Die Dateien dieser Implementierung sollen nicht in die Cloud hochgeladen werden und werden im Ordner `/var/mobile/Applications/<UUID>/Library/NoCloud/` gespeichert. Der iOS-Verzeichnispfad für dieses Verzeichnis kann direkt über die Cordova-File-Variable `cordova.file.dataDirectory` adressiert werden.

Android unterscheidet im Gegensatz zum iOS Dateisystem lediglich zwischen Cache und normalen Dateien. Cache Dateien werden unter Android laufend gelöscht. Sie werden unter `/data/data/<app-id>/cache/` gespeichert [35]. Normale Dateien werden unter `/data/data/<app-id>/files/` gespeichert. Dieser Pfad ist ebenfalls über `cordova.file.dataDirectory` verfügbar.

Windows unterscheidet, wie auch iOS, zwischen temporären, Cache und normalen Dateien. Allerdings werden Temporäre und Cache Dateien im selben Ordner unter `ms-appdata://temp/` gespeichert [35]. Dieses Verzeichnis kann allerdings vom Betriebssystem jederzeit gelöscht werden.
Normale Dateien werden beim Windows-Dateisystem unter `ms-appdata://local/` gespeichert. Über die Cordova-File-Variable lässt sich dieser Verzeichnispfad ebenfalls unter `cordova.file.dataDirectory` erreichen.

Implementierung

Damit der Pfad der Medien-Dateien mit dem aus dem Executable Business Process übereinstimmt, müssen die Verzeichnisse nachgebildet werden. Hierfür wurde die Methode `checkAndCreateDirectory` (siehe Listing A.1) entwickelt. Die Methode löst zunächst den Daten-Verzeichnispfad auf. Anschließend wird über die Funktion `directoryEntry.getDirectory()` versucht, den übergebenen `path` aufzulösen. Gelingt das, existiert der Pfad bereits und muss nicht erstellt werden. Kann der Verzeichnis-Pfad nicht aufgelöst werden, wird er, durch die Angabe von `{create: true}`, erstellt.

Im nächsten Schritt muss jetzt nur noch die Media-Datei gespeichert werden. Diese Funktion wird durch die `writeFile` Methode (siehe Listing A.2) implementiert. Zunächst wird wieder der Verzeichnispfad des Data-Verzeichnisses aufgelöst. Anschließend wird mit der Funktion `fileEntry.getFile()` die übergebene Datei geöffnet bzw. erstellt. Beim Schreiben in die Datei, werden die Daten in den Datentyp Blob umgewandelt. Damit diese auch durch die Applikation richtig dargestellt werden, wird der passende Multipurpose Internet Mail Extensions (MIME)-Typ zur Datei hinzugefügt. Anschließend kann die Datei wie eine normale Ressource über einen direkten Pfad verwendet werden.

6.2.2 Prozessinstanz ausführen

In diesem Szenario hat der Benutzer bereits ein Prozessmodell importiert (siehe Abschnitt 6.2.1). Im Anschluss wurde eine Prozessinstanz gestartet. Um die Ausführung der Prozessinstanz fortzusetzen, löst der Benutzer das Szenario aus Abbildung 6.7 aus.

Implementierung

Eine Benutzerinteraktion hat zur Folge, dass von der Prozess-Engine die Funktion `getNextNode()` aufgerufen wird. Hierbei wird der Funktionsaufruf über den Execution Manager (ohne Abbildung) bis zum Instance Manager durchpropagiert. Hier beginnt das Szenario aus Abbildung 6.7. Die `getNextNode()` Funktion des Instance Manager ruft die eigentliche `handleNode()` Funktion des Instance Managers auf (siehe A.3).
In dieser Funktion findet die Knotenunterscheidung statt. Für jeden Typ wird dabei eine individuelle Behandlung aufgerufen (siehe Listing A.3). Im Szenario aus Abbildung 6.7 wird ein Knoten vom Typ `NT_NORMAL` (siehe Tabelle 5.1) erkannt und behandelt. Dabei wird die Behandlungsfunktion `handleNormalNode()` (siehe Listing A.4) aufgerufen. Der Knoten enthält dabei einen Executable Business Process, der mit der Funktion `processNode(node)` an den Runtime Manager weitergeleitet wird. Wie Listing A.4 zeigt, wird der Executable Business Process über den CommunicationService an den Applikations-Komponenten Manager weitergeleitet, sofern der Executable Business Process eine Benutzeroberfläche darstellt.
Hierbei implementiert der Applikations-Komponenten Manager eine Behandlung, die den Executable Business Process erhält. Des Weiteren übernimmt der Applikations-Komponenten Manager die Verwaltung der Anwendungsbausteine und entscheidet, welcher der Anwendungsbausteine den Executable Business Process implementiert. Dazu wird der entsprechende Anwendungsbaustein erstellt, der den Executable Business Process interpretiert und darstellt. In Listing A.5 wird veranschaulicht, wie die Empfängerseite des Communication Service verwendet wird. Die Besonderheit, die in den Listings A.4 und A.5 implementiert wurde, ist die Verwendung von der JavaScript-Erweiterung `RxJS`. Dadurch ist es möglich (sog. `Observables`) asynchrone, event-basierte Funktionsaufrufe zu verwenden. Dabei ähnelt ein Observable einem Java-Stream.

Das nächste Szenario ähnelt dem gerade vorgestellten aus Abbildung 6.7. Der Benutzer möchte die Prozessausführung fortsetzen und löst das Szenario aus Abbildung 6.8 aus. Dadurch wird erneut die `getNextNode()` Methode der Prozess-Engine über den Execution Manager (ohne Abbildung) bis zum Instance Manager durchpropagiert. Dieser führt ebenfalls, wie in Abbildung 6.7 die Funktion `handleNode()` (siehe Listing A.3)

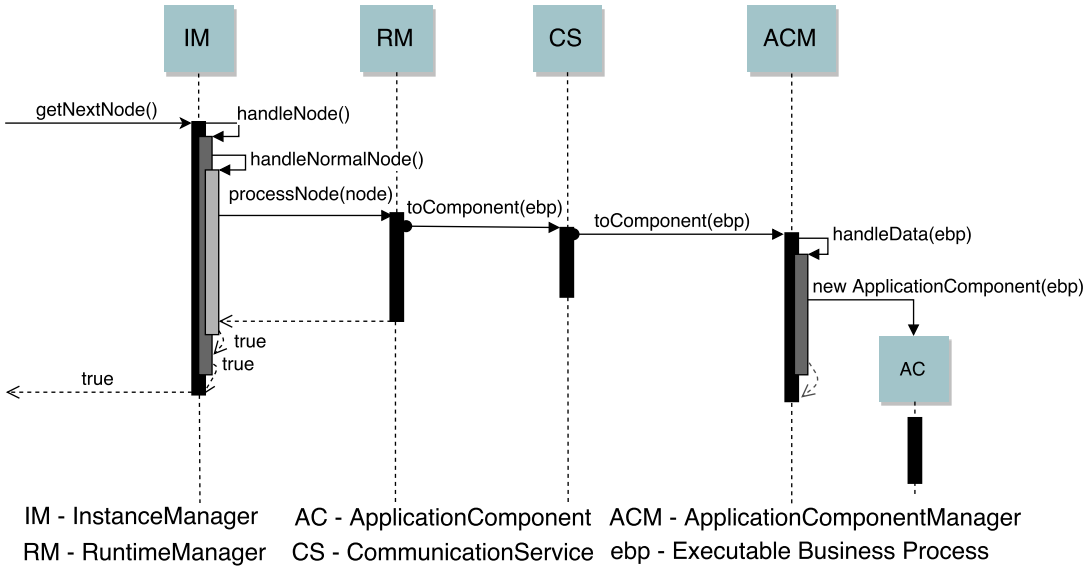

Abbildung 6.7: Anwendungsfall: Prozessausführung. Der ausgeführte Knoten enthält einen Executable Business Process, der eine Benutzeroberfläche generiert.

aus. Diesmal wird jedoch ein `XOR_SPLIT` (siehe Tabelle 5.1) erkannt und behandelt (siehe Listing A.6). Zuerst wird geprüft, ob dieser Knoten einen Executable Business Process besitzt. Ist dies nicht der Fall, wird die Ausführung dieses Knotens wegen eines Fehlers abgebrochen. Dies ist nötig, da ohne Executable Business Process keine Entscheidung gefällt werden kann, mit welchem Zweig fortgefahren werden soll. Damit nicht die Ausführung der gesamten Prozessinstanz abgebrochen wird, muss der Pfad mit dem Index 0 als Nächstes ausgeführt werden.

Besitzt der Knoten jedoch einen Executable Business Process (EBP), wird dieser im ersten Schritt mit der Funktion `processXORNode(xor)` an den Runtime Manager weitergeleitet, damit er dort weiter behandelt werden kann. Wie Listing A.7 zeigt, wird der Executable Business Process, wie auch schon in Szenario aus Abbildung 6.7 an den CommunicationService (CS) weitergeleitet. Allerdings wird hier im Gegensatz zu Abbildung 6.7 ein Service angesprochen, der im CommunicationService anhand der ImplementationClass identifiziert wird. Zusätzlich werden dem behandelnden Service, die Datenelemente mitgegeben, auf die der Knoten lesend zugreift. Als Resultat liefert die Funktion `handleData` des ausgewählten Service des Zweiges zurück, die im nächsten

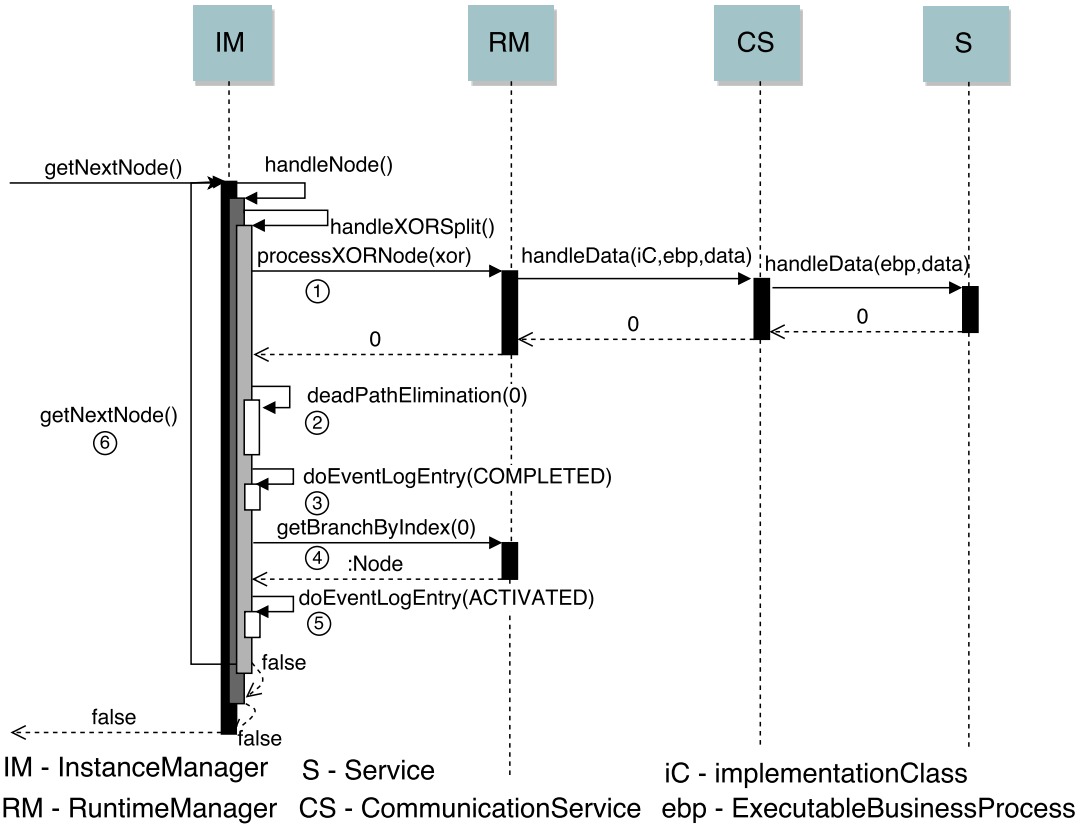

Abbildung 6.8: Anwendungsfall: Prozessausführung. Der ausgeführte Knoten enthält einen Executable Business Process, der eine automatische Funktion darstellt.

Schritt traversiert werden soll. Durch das Behandeln des Executable Business Process wird die Auswahl des Zweiges zur Laufzeit berechnet (siehe **FA5**).
Im Schritt 2, der XOR-Split Knotenbehandlung, werden alle nicht zutreffenden Zweige über die sogenannten Dead-Path-Elimination im Ausführungsprotokoll markiert (siehe Listing A.8). Bei der Dead-Path-Elimination werden alle Knoten, die nicht mehr erreicht werden können als `SKIPPED` (siehe Abbildung 5.4) markiert. Hierfür bekommt die Funktion den Identifikator des Zweiges übergeben, der nicht markiert werden soll. Das ist für den Zweig notwendig, der durch die Bedingungsevaluierung ausgewählt wurde. An dieser Stelle wird anschließend mit der Ausführung der Prozessinstanz fortgefahren.

Gestartet wird die Dead-Path-Elimination am ersten Knoten der Zweige, die für die weitere Ausführung nicht mehr in Frage kommen. Im Folgenden wird der gesamte Zweig

traversiert, solange die Funktion nicht am korrespondierenden Join-Knoten des XOR-Split-Knotens angelangt ist. Anschließend wird der eben beschriebene Verlauf mit den anderen, nicht ausgewählten Zweigen ausgeführt.

Da alle Zweige, die nicht mehr erreicht werden können, als `SKIPPED` markiert sind, kann mit der Ausführung fortgefahren werden. Durch den Abschluss der Dead-Path-Elimination ist die Behandlung des XOR-Split-Knotens abgeschlossen. Dieser Zustand wird in Schritt 3 (siehe Listing A.6) im Ausführungsprotokoll mit `COMPLETED` vermerkt (siehe Abbildung 5.4). In Schritt 4 wird der erste Knoten des durch die Bedingungsevaluierung ausgewählten Zweiges als nächsten Knoten zur Fortsetzung der Prozessausführung ausgewählt. Der Zustand des Knotens wird in Schritt 5 im Ausführungsprotokoll als `ACTIVATED` (siehe Abbildung 5.4) vermerkt. Im Anschluss wird dieser in Schritt 6 direkt über den Funktionsaufruf `getNextNode()` behandelt. In diesem Szenario handelt es sich jedoch um einen Knoten, der keinen Executable Business Process besitzt, deshalb wird `false` zurückgegeben.
Im Unterschied zu dem Szenario aus Abbildung 6.7 wird in Abbildung 6.8 ein Executable Business Process ausgeführt, der keine Benutzerinteraktion benötigt. Im Folgenden werden solche Executable Business Processes (EBPs) als automatisch bezeichnet. Da diese automatischen EBPs keine Benutzerinteraktion benötigen, auf die die Prozessausführung warten muss, erfolgt die Behandlung über synchrone Funktionsaufrufe.

Dafür wird zunächst ein Service, der einen automatischen Executable Business Process implementiert über den Application-Component Manager im CommunicationService registriert. Soll ein solcher Service ausgeführt werden, muss im CommunicationService lediglich die `ImplementationClass` des Executable Business Process mit der des Service verglichen werden. Stimmen diese überein, kann die Funktion `handleData` des jeweiligen Service ausgeführt werden (siehe Listing A.9).
Anhand der beiden Szenarien aus den Abbildungen 6.7 und 6.8 kann eine Konvention für die Implementierung der Executable Business Processes abgeleitet werden. Benötigt ein EBP eine Benutzerinteraktion, ist dieser als Angular Komponente zu implementieren und über einen asynchronen, event-basierten Funktionsaufruf zu verwenden. Wird keine Benutzerinteraktion benötigt, so wird dieser automatische Executable Business Process als Service implementiert und über normale, synchrone Funktionsaufrufe verwendet.

6.3 Diskussion

In diesem Abschnitt wird diskutiert, ob sich die in dieser Implementierung verwendeten Technologien für die Entwicklung einer mobilen Applikation eignen. Hierzu werden Probleme diskutiert, die während der Entwicklung aufgetreten sind. Den Anfang macht dabei die Unterstützung nativer Plugins über Ionic Native. Im Anschluss daran wird mit der Entwicklung der Benutzeroberfläche fortgefahren.

6.3.1 Native Unterstützung

Ionic Native stellt TypeScript Wrapper für native Plugins, wie zum Beispiel für das Cordova File Plugin, zur Verfügung. Im Laufe der Implementierung gab es hier einige Probleme. Zum einen mit der Datenbankanbindung, zum anderen mit dem File Plugin.

Datenbankanbindung: Zunächst gab es an dieser Stelle keine Probleme, bis zum Ionic Update auf die Version 2.0.0-beta.11. Im Zuge des Updates mussten hier auch die Ionic Native Komponenten auf die Version 1.3.10 aktualisiert werden, damit eine stabile Anwendung durch Ionic gewährleistet werden kann. In Folge des Ionic Native Updates, wurde der Wrapper der Datenbankanbindung verändert. Die Dokumentation hierzu stellte sich sehr schnell als unzureichend heraus, da wichtige Details, die zur Implementierung nötig waren, zu diesem Zeitpunkt noch nicht dokumentiert waren. Aufgrund der veränderten Datenbankanbindung musste der Datenbank-Service komplett neu implementiert werden, da der alte Datenbank-Service mit der neuen Datenbankanbindung nicht mehr kompatibel war.

Cordova File Plugin: Im Gegensatz zur Datenbankanbindung, gab es hier von Anfang an Probleme. Der Wrapper des File Plugins besaß zum Zeitpunkt nur Methoden zum Erstellen, Verschieben und Lesen neuer Dateien. Problematisch hieran war, dass sich eine Datei nur erstellen ließ. Sie konnte nicht mit Daten beschrieben werden. Dazu kam noch, dass bei den implementierten Lese-Methoden über eine optionale Angabe der Rückgabe-Typ, zum Beispiel als Data-URL, Text, Binär-String oder als Binär-Vektor, ausgewählt werden konnte. Das Problem hierbei war, dass manche Rückgabe-Typen nicht implementiert waren. Hinzu kam noch, dass in der

Dokumentation nicht darauf hingewiesen wurde. Besonders das Fehlen der Schreiboperationen zum Schreiben in Dateien, welche für die Entwicklung der Applikation besonders wichtig sind, waren problematisch. Um dieses Problem zu umgehen, wurde ein eigener Wrapper als Workaround-Lösung implementiert. Mittlerweile wurden diese Schreiboperationen im Ionic Native Wrapper implementiert.

Zusammenfassend lässt sich feststellen, dass bei den Ionic Native Komponenten große Probleme aufgetreten sind. Diese Probleme wurden jedoch durch erneute Implementierungen und Workaround-Lösungen behoben.

6.3.2 Entwicklung der Benutzeroberflächen

Wie auch bei der nativen Unterstützung gab es auch bei den Ionic Komponenten bzw. mit Angular einige Probleme zu lösen.

Unzureichende Dokumentation: Die meisten Probleme, die mit der Benutzeroberfläche aufgetreten sind, sind auf die Dokumentation zurückzuführen.
Da sich während der Implementierung sowohl Angular 2 als auch Ionic 2 in der Beta-Testphase befunden haben, waren die Dokumentationen noch nicht vollständig und teilweise nicht aktuell. Somit musste des Öfteren der Quellcode beider Frameworks konsultiert werden, um die Verwendung zum Beispiel von Optionen nachvollziehen zu können.
So gab es beispielsweise Probleme bei der Verwendung der Ionic Grid-Umgebung mit dem Inhalt von Listenelementen. Mit einer Ionic Grid-Umgebung lässt sich Inhalt wie mit einem Raster strukturieren. Möchte man den Inhalt von Listenelementen mit so einem Raster strukturieren, darf der Inhalt kein Ionic Input-Feld beinhalten, da sonst die Applikation abstürzt. Dieses Problem wurde gelöst, indem eine andere Strukturierung der Komponenten verwendet wurde.
Ebenfalls wurde in der Dokumentation nichts über eine Inkompatibilität von Input-Elementen und der Grid-Umgebung innerhalb eines Listenelementes erwähnt.

Fehler in der Implementierung: Wie in wahrscheinlich jeder Beta-Testphase, sind auch im Laufe dieser Implementierung Fehler aufgetreten, die sich nicht ohne

weiteres lösen ließen. Dabei handelt es sich um Fehler in der Angular bzw. Ionic Implementierung. Dies war zum Beispiel im Zusammenhang mit dem Ionic-Select Auswahl-Menü der Fall. Wurde vor dem Öffnen des Auswahl-Menüs, welches einer HTML-Auswahlliste, ein Ionic-Alert aufgerufen und dieser mit `dismiss()` geschlossen, lies sich das Auswahl-Menü nicht mehr schließen. Der Grund hierfür war ein Fehler in der `dismiss()`-Funktion. Dieser Fehler wurde in der Ionic Version 2.0.0-beta.11 behoben, was ein Update auf diese Version nötig machte.

Es lässt sich feststellen, dass jeder der, eine mobile Applikation entwickeln möchte, sich die Frage stellen muss, ob die Verwendung solcher neuen Technologien sinnvoll ist, beziehungsweise, ob sie sich für die gewünschte Applikation eignen. Jede neue Technologie besitzt Vorteile aber auch Nachteile. In diesem konkreten Fall wurde die Entwicklung mit Ionic bzw. Angular in Version 2 deutlich einfacher gestaltet als in der komplexeren Version 1. Allerdings besaß die Version 2, während der Umsetzung dieser Arbeit, noch keine stabile, finale Veröffentlichung. Somit muss jeder Entwickler für sich entscheiden, ob er das Risiko eingehen möchte, mit einer Beta-Version zu entwickeln. Hierbei kann es möglich sein, dass bereits implementierte Teile nach einem Versions-Upgrade, komplett neu entwickelt werden müssen.

In Summe muss also jeder Entwickler für sich selbst entscheiden, ob er lieber mit ausgereifteren Technologien arbeitet oder neue Technologien verwendet, die unter Umständen zu Mehraufwand führen. Auf jeden Fall sollte bei der Entscheidung auch die Qualität der Dokumentation herangezogen werden. Denn eine gute Dokumentation kann auch Schwächen von Beta-Implementierungen deutlich reduzieren.

In diesem Kapitel wurde die Implementierung vorgestellt. Zunächst wurde in Abschnitt 6.1 mit den verwendeten Frameworks begonnen. Anschließend wurden im Abschnitt 6.2 interessante Aspekte der Implementierung vorgestellt. Im darauffolgenden Abschnitt 6.3 werden Probleme bei der Implementierung genannt und diskutiert. Im nächsten Kapitel wird nun ein konkretes Anwendungsszenario für die Prozess-Engine vorgestellt.

7

Anwendungsszenario

In diesem Kapitel wird ein Anwendungsszenario für die umgesetzte mobile Prozess-Engine vorgestellt. Dabei wird die Prozess-Engine zur mobilen Datenerhebung eingesetzt. Dies geschieht im Kontext des *QuestionSys*-Frameworks, das zunächst in Abschnitt 7.1 vorgestellt wird. Im Abschnitt 7.2 wird anschließend präsentiert, wie sich die entwickelte mobile Prozess-Engine in das QuestionSys-Framework einfügt und wie mit ihr eine mobile Datenerhebung durchgeführt werden kann.

7.1 Das QuestionSys-Framework

Das *QuestionSys* Projekt wurde im Jahr 2013 ins Leben gerufen. Ziel ist es einen generischen Ansatz zu entwickeln, der den Lebenszyklus für mobile und digitale Fragebögen unterstützt. Dieser Lebenszyklus besteht dabei aus fünf aufeinander aufbauenden Stufen, die in Abbildung 7.1 veranschaulicht werden. Am Anfang muss ein Fragebogen zunächst erstellt werden (P1). Im nächsten Schritt soll dieser Fragebogen auf die mobilen Endgeräte verteilt werden (P2). Anschließend wird der Fragebogen dort instanziiert und ausgeführt (P3). Die durch den Fragebogen gesammelten Daten können dann evaluiert und analysiert werden (P4). Zum Schluss werden die Fragebögen archiviert (P5) [36]. QuestionSys verwendet zum Abbilden der Fragebögen einen prozessorientierten Ansatz [37]. Dabei werden Fragebögen auf Prozessmodelle (wie im Kapitel 2.2 beschrieben) abgebildet. Dadurch ist es möglich, dass Fragebögen durch eine Prozess-Engine ausgeführt werden. Verwendet man eine mobile Prozess-Engine, wie sie in dieser Arbeit entwickelt wurde, lassen sich die Fragebögen auch auf mobilen Geräten ausführen.

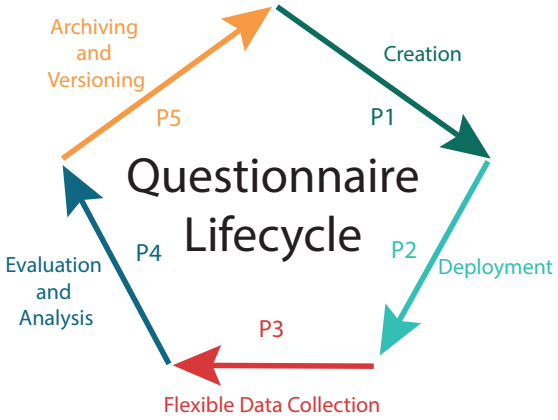

Abbildung 7.1: Lebenszyklus für mobile und digitale Fragebögen (nach [36]).

In den folgenden Abschnitten wird nun die Architektur des QuestionSys Frameworks näher erläutert. Diese wird in Abbildung 7.2 illustriert.

7.1.1 Konfigurator

Der Konfigurator ist für das Erstellen der Fragebögen verantwortlich. Ein mit dem Konfigurator erstellter Fragebogen besteht aus einzelnen Seiten. Diese Seiten sind in einer logischen Beziehung miteinander verbunden. Die Art wie diese Seiten verbunden sind, bestimmt die Ausführungslogik des Fragebogens. Diese Ausführungslogik kann hierbei über Drag & Drop konfiguriert werden.
Eine Seite des Fragebogens kann dabei aus mehreren Elementen bestehen. Im Moment werden Fragen, Überschriften, Texte und Bilder unterstützt. Dabei können diese Elemente in mehreren Sprachen angelegt werden.
Die mit dem Konfigurator erstellten Fragebögen lassen sich auch in andere Formate transformieren und exportieren. Ein Fragebogen kann als Prozessmodell exportiert werden um beispielsweise auf einem Server für Clients zum Herunterladen angeboten werden zu können. Außerdem gibt es die Möglichkeit die Fragebögen als PDF-Dokument oder als einzelne HTML-Datei zu exportieren.

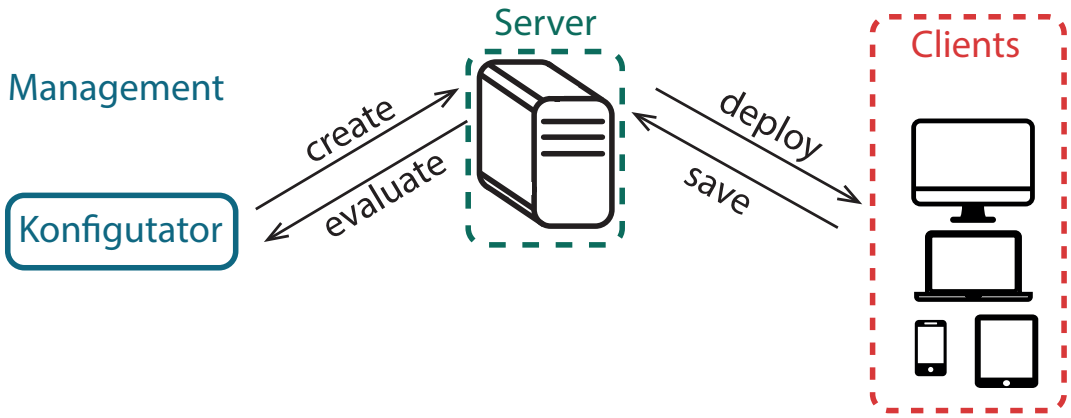

Abbildung 7.2: Zusammenspiel der QuestionSys Komponenten (nach [36]).

7.1.2 Server

Der Server ist für das Verteilen der Fragebogen-Modelle an die Clients zuständig. Außerdem können die erhobenen Daten auf den Server hochgeladen werden, damit diese später ausgewertet werden können.

7.1.3 Client

Um die Fragebögen anzeigen und beantworten zu können, sind unterschiedliche Clients denkbar. Unter anderem wurde in [38] ein moderner Web-Client entwickelt. Zudem bietet die HTML-Exportfunktion ebenfalls einen einfachen Client an.

7.2 Prozess-unterstützte mobile Datenerhebung

Im Abschnitt 7.1 wurden die Grundlagen zum QuestionSys-Framework erläutert. Dabei wurde im Abschnitt 7.1.3 der Client zur Beantwortung der Fragebögen vorgestellt. Da die Fragebögen auf Prozessmodellen basieren, liegt der Schluss nahe, die Ausführung der Fragebögen mithilfe der entwickelten mobilen Prozess-Engine auszuführen.
Da die Prozess-Engine generisch gehalten wurde, lässt sich dieser Ansatz leicht implementieren. Es müssen lediglich alle verwendeten Executable Business Processes

implementiert werden. Dazu gehört vor allem die Benutzeroberfläche, die aus dem Executable Business Process generiert wird. Sie muss als Komponente implementiert werden. Außerdem müssen alle automatisch ablaufenden Executable Business Process als Service implementiert werden. Im Folgenden werden zunächst die architektonischen Aspekte betrachtet, damit mit der Prozess-Engine Fragebögen ausgeführt werden können. Im Anschluss werden Beispiele gegeben, wie die Verwaltung und Ausführung eines solchen Fragebogens aussehen kann.

7.2.1 Erweiterung der bisherigen Architektur

Die Abbildung 7.3 zeigt die Erweiterungen, die nötig sind, um einen Fragebogen auf der mobilen Prozess-Engine ausführen zu können. Dabei fallen die `QuestionnairePage` Komponente und der `XOR_Questionnaire` Service auf. Zunächst wird auf die Benutzeroberfläche `QuestionnairePage` eingegangen. Der `XOR_Questionnaire` Service wird im Anschluss an die Benutzeroberfläche behandelt.

Benutzeroberfläche

Die `QuestionnairePage` implementiert einen Executable Business Process, der bei der Prozessausführung auftritt. Sie generiert die anzuzeigende Benutzeroberfläche. Außerdem werden an dieser Stelle die Daten erhoben, die durch die Ausführung des Fragebogens gesammelt werden. Sie werden über den Kommunikations-Service an den Runtime Manager zur sicheren Verwahrung übersendet.

Entscheidungen

Zusätzlich zur Benutzeroberfläche benötigt die Ausführung von Fragebögen auf der mobilen Prozess-Engine einen zusätzlichen Service. Dieser wird benötigt, da zur Ausführung des Fragebogens auch Entscheidungen getroffen werden müssen. Die dafür benötigten Daten, wie zum Beispiel das Evaluations-Statement und die benötigten Datenelemente werden von einem Executable Business Process abgebildet. Da die

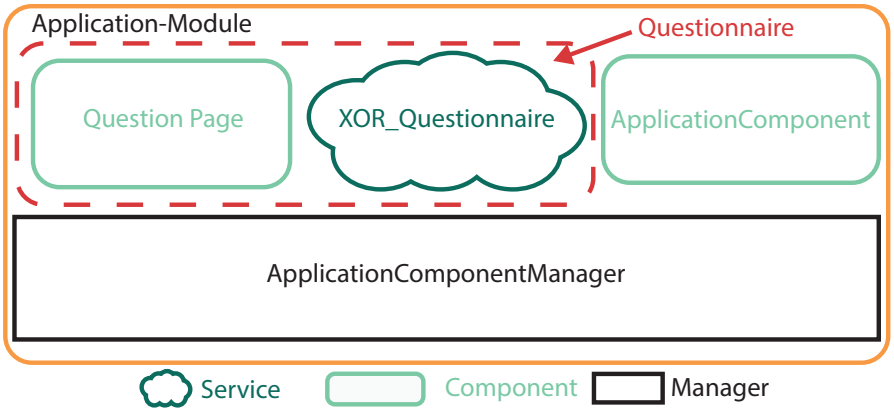

Abbildung 7.3: Erweiterung des Applikations-Moduls um die *QuestionPage* Komponente und den *XOR_Questionnaire* Service.

Executable Business Processes jedoch Anwendungsfall abhängig sind, in diesem Fall vom QuestionSys-Framework, kann das Fällen von Entscheidungen nur durch einen QuestionSys individuellen Service erfolgen. In diesem Fall wird das durch den `XOR_Questionnaire` Service erledigt. Zur Evaluation erhält er den Executable Business Process und die Daten, die im Laufe der Prozessausführung gesammelt wurden, sodass sie diese nach dem individuellen Antwort-Schema analysieren kann. Im Endeffekt wird von dem `XOR_Questionnaire` Service nur eine Zahl zurückgeliefert, welche signalisiert, mit welchem Zweig fortgefahren werden soll. Anhand dieser Entscheidung wird dann die Ausführung der Prozessinstanz fortgesetzt.

7.2.2 Ausführung eines Fragebogens durch die mobile Prozess-Engine

In den vorherigen Abschnitten wurden das QuestionSys-Framework und die architektonischen Erweiterungen besprochen. Im Folgenden werden die Schritte beschrieben, die benötigt werden, damit ein Fragebogen durch die Prozess-Engine ausgeführt werden kann. Die Abbildung 7.4 liefert einen Überblick der auszuführenden Schritte. Als Erstes muss ein Fragebogen vom Server heruntergeladen und importiert werden (1). Im Anschluss daran kann der Fragebogen ausgeführt werden (2). Im Schritt 3 werden die erhobenen Daten wieder auf den Server hochgeladen. Alle diese Schritte werden durch die entwickelte hybride Applikation abgedeckt.

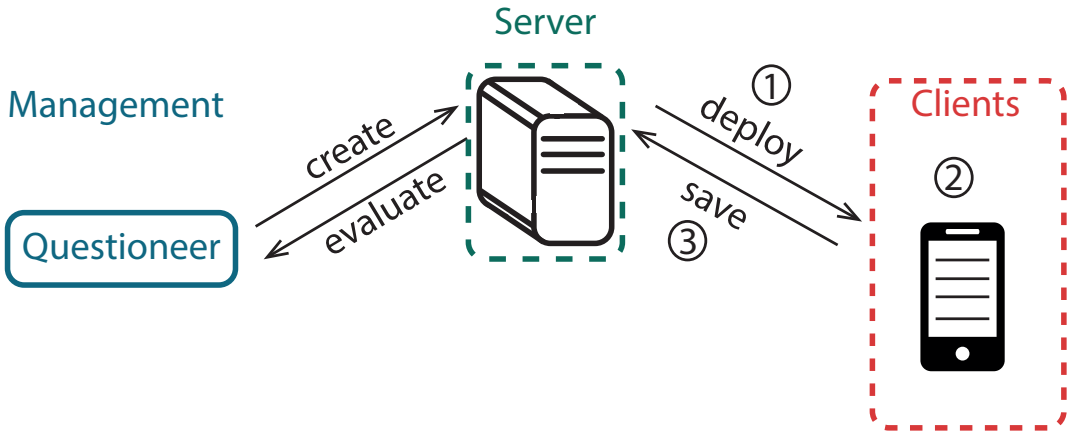

Abbildung 7.4: Verteilung (1) und Ausführung (2) eines Fragebogens mit anschließendem Hochladen der gesammelten Daten (3) (nach [36]).

1. Herunterladen und Importieren eines Fragebogens

Die Abbildung 7.5 zeigt das Herunterladen und Importieren eines Fragebogens, das im Folgenden detailliert beschrieben wird.
Damit ein Fragebogen heruntergeladen werden kann, muss zunächst eine gesicherte Verbindung zum Server aufgebaut werden. Um sich am Server anmelden zu können, muss das dafür zuständige Menü (**A**) geöffnet werden. Im nächsten Schritt (**B**) kann sich der Benutzer anmelden. Ist die Anmeldung erfolgreich, wird der Benutzer auf eine Seite weitergeleitet, die die für ihn sichtbaren Bereiche anzeigt (**C**). Nachdem einer dieser Bereiche ausgewählt wurde, werden die enthaltenen Fragebögen angezeigt. Diese Fragebögen können dann direkt heruntergeladen werden (**D**). Wurde ein Fragebogen ausgewählt und heruntergeladen, wird dieser in die Prozess-Engine importiert. Anschließend kann der Fragebogen direkt ausgeführt werden.

2. Ausführen des Fragebogens

Die Abbildung 7.6 illustriert die Ausführung eines Fragebogens, die im folgenden Abschnitt detailliert beschrieben wird.
Um einen Fragebogen zu starten, muss dieser vom Benutzer ausgewählt werden (**E**).

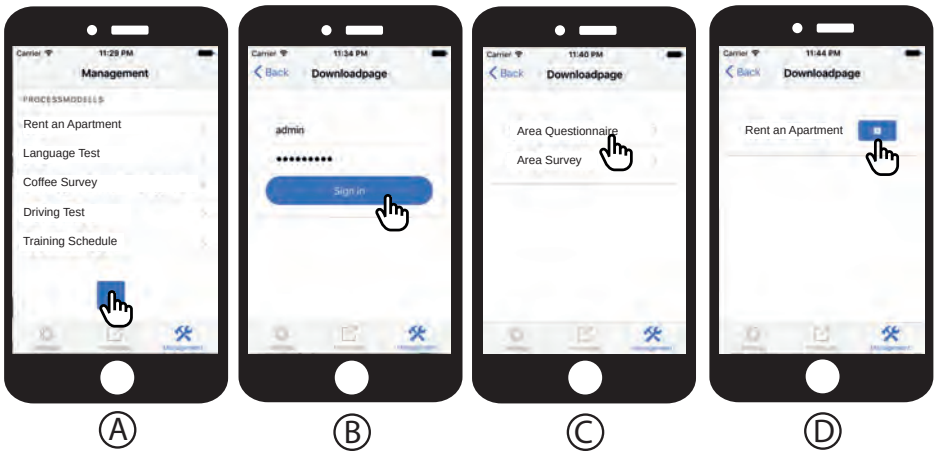

Abbildung 7.5: Schritt 1: Herunterladen und importieren eines Fragebogens.

Im Anschluss erscheint ein Auswahlmenü, zur Sprachauswahl (**F**). Sobald eine Seite vollständig bearbeitet wurde, kann der Benutzer auf der nächsten Seite fortfahren (**G**). Dies muss vom Benutzer solange durchgeführt werden, bis der Fragebogen vollständig bearbeitet wurde. Ist dies der Fall, so kehrt der Benutzer wieder zur Startseite zurück (**E**). Die durch die Ausführung des Fragebogens erhobenen Daten sind zunächst auf dem mobilen Endgerät gespeichert und können nun dem Server übergeben werden.

3. Hochladen der erhobenen Daten

Zum Hochladen der erhobenen Daten (siehe Abbildung 7.7) muss der Benutzer die Fragebogen-Detail-Ansicht öffnen (**H**). Hat ein Benutzer ein Prozessmodell ausgewählt, werden ihm alle verfügbaren Prozessinstanzen angezeigt (**I**). Nun kann er die gesammelten Daten der Prozessinstanzen mit einem Klick auf das Synchronisations-Icon (**J**) hochladen oder die erhobenen Daten samt Prozessinstanz löschen (Mülleimer-Icon). Sollen die gesammelten Daten hochgeladen werden, wird ein Upload-Service verwendet. Bevor die Daten hochgeladen werden, muss sich der Benutzer am Server anmelden (**K**). Ist die Anmeldung erfolgreich, werden die Daten hochgeladen. Zum Hochladen wird eine Transportverschlüsselung verwendet um die Daten vor unbefugtem Zugriff zu schützen.

Abbildung 7.6: Schritt 2: Ausführen eines Fragebogens.

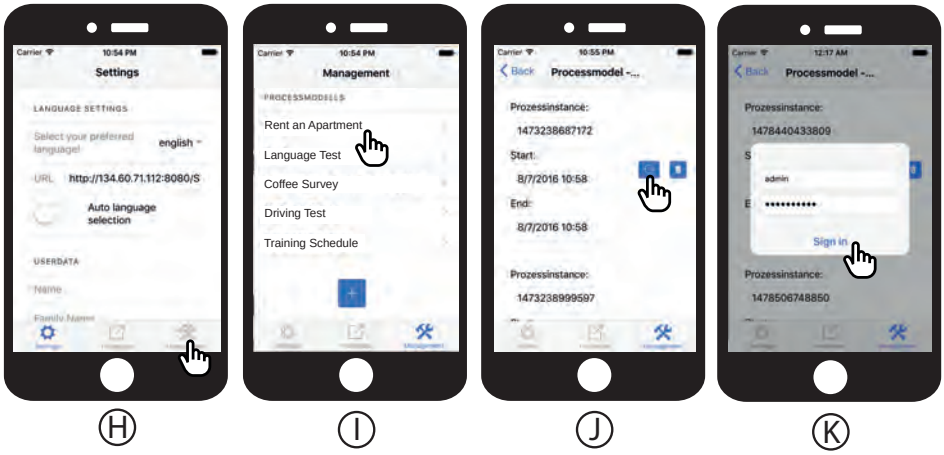

Abbildung 7.7: Schritt 3: Hochladen der gesammelten Daten.

In diesem Kapitel wurde eine Anwendungsmöglichkeit vorgestellt, die entwickelte mobile Prozess-Engine praktisch einzusetzen. Hierfür wurde zunächst QuestionSys vorgestellt. Im Anschluss daran wurde die Integration der Questionnaire-Oberflächenkomponente sowie die des XOR-Service aufgezeigt. Darauf folgte ein kleines Fallbeispiel, das zeigt, wie die Prozess-Engine sich konkret in das QuestionSys-Framework einfügt. Dazu wurden einige Beispiel-Abläufe vorgestellt.
Im nächsten Kapitel wird ein abschließendes Fazit gezogen und mögliche Erweiterungen der entwickelten mobilen Prozess-Engine aufgezeigt.

8

Zusammenfassung und Ausblick

Ziel der Arbeit war es, eine mobile und flexible Prozess-Engine zu entwickeln, die mit neuen Web-Technologien implementiert wird. Bei der Entwicklung wurde viel Wert auf eine leichte Portierbarkeit der Applikation gelegt. Somit lässt sich die Prozess-Engine einfach in anderen Projekten verwenden. Außerdem werden die mobilen Plattformen iOS, Android und Windows unterstützt werden.

Zunächst wurden Anforderungen erörtert, welche durch die später implementierte Applikation umsetzt werden sollen. Hierbei lag der Fokus auf der Prozessausführung. Die Prozess-Engine sollte konditionale und parallele Verzweigungen ausführen können. Dazu wurde auch eine Protokollierung während der Prozessausführung gefordert, damit die Prozessmodelle später durch Process-Mining Technologien analysiert werden können. Bei den nicht-funktionalen Anforderungen wurde der Fokus vor allem auf der Mehrsprachigkeit der Applikation gelegt.

Aufbauend auf der Anforderungsanalyse wurde mit der Konzeption des Gesamtsystems begonnen. Dabei besteht die Applikation aus mehreren Modulen. Das wichtigste Modul ist die Prozess-Engine, die aus drei Managern besteht. Der `Runtime Manager` ist für den Datenfluss einer Prozessinstanz verantwortlich. Der Kontrollfluss wird vom `Instance Manager` überwacht. Der `Execution Manager` verwaltet den Zugriff auf die `Instance Manager`. Außerdem ist er für den Import von Prozessmodellen, sowie das Speichern und Wiederherstellen angefangener Prozessinstanzen verantwortlich. Zusätzlich zu den drei Managern besitzt die Prozess-Engine auch noch eine Reihe von Diensten, die beispielsweise den Zugriff auf das Dateisystem, die Datenbank oder den Kommunikationskanal erlauben.

Aufbauend auf der Architektur wurde die Applikation implementiert. Dabei wurden mit Ionic 2, Angular 2 und Cordova aktuelle Webtechnologien für die Implementierung verwendet. Der größte Vorteil, den diese Technologien mit sich bringen, ist, dass eine Implementierung ausreichend ist, um Applikationen für mehrere Plattformen zu entwickeln. Außerdem stand die offline Prozessausführung bei der Implementierung im Vordergrund, durch die die gesamte Ausführungslogik auf dem mobilen Gerät ausgeführt wird.

Im Anschluss an die Implementierung wurde ein Anwendungsszenario präsentiert, das zeigt, wie die mobile Prozess-Engine eingesetzt werden kann. Hierfür wurde das QuestionSys Projekt vorgestellt. Das Projekt beschäftigt sich mit mobiler Datenerhebung durch digitale Fragebögen. Dabei werden die Fragebögen auf Prozessmodelle abgebildet, die dann die mobile Prozess-Engine ausführen kann. Um die Fragebögen auch grafisch repräsentieren zu können, wurden Anwendungsbausteine für die Client-Komponente des QuestionSys Frameworks entwickelt.

8.1 Ausblick

In diesem Abschnitt werden nun Möglichkeiten vorgestellt, wie die Funktionen der Prozess-Engine in Zukunft erweitert werden könnten.

Die bisherige Implementierung der Prozess-Engine erlaubt nur eine Ausführung von konditionalen und parallelen Verzweigungen. Daher wäre eine Implementierung einer Schleife der nächste logische Schritt, um die Prozess-Engine zu erweitern. Außerdem ist es bisher nur möglich, Prozessinstanzen ohne Berücksichtigung von Synchronisationskanten auszuführen. In Zukunft wäre eine Berücksichtigung solcher Synchronisationskanten aber von Vorteil, um Prozessmodelle, die diese Kanten zwingend benötigen, zu unterstützen. Hierbei könnte auch eine Unterscheidung zwischen strikten und einfachen Synchronisationskanten erfolgen, wie dies in [9] angedacht wurde.

Eine weitere Möglichkeit, die mobile Prozess-Engine zu erweitern, wäre die Integration von Ad-hoc Änderungen am Prozessmodell. Dies würde die Funktionalität deutlich erweitern. Ein Beispiel hierfür wäre die Anwendung im medizinischen Bereich. Während einer Untersuchung könnte die Aktivität „MRT Untersuchung“ für einen Patienten mit

Herzschrittmacher übersprungen beziehungsweise ausgelassen werden. Im Zuge dessen könnte die Prozess-Engine auch noch um eine Modellierungs-Komponente erweitert werden. Das hätte den Vorteil, dass ein Prozess nicht an einer Workstation modelliert werden muss, sondern die Modellierung direkt während eines Arbeitsablaufes erfolgen kann. Dadurch können die Prozessemodelle näher an die Realität angepasst werden. Dadurch würden Modellierungsfehler des Prozessablaufs deutlich reduziert.

Eine weitere Möglichkeit die Prozess-Engine zu erweitern wäre ein Wrapper für die verschiedenen Prozessbeschreibungssprachen. Dadurch müsste das Prozessmodell nicht mehr ausschließlich in der blockstrukturierten ADEPT-Notation vorliegen, sondern könnte beispielsweise auch in BPMN 2.0 vorliegen. Dies würde den Arbeitsaufwand reduzieren, ein BPMN 2.0 Prozessmodell in die ADEPT-Notation zu übersetzen. Die automatisierte Transformation hätte außerdem den Vorteil, dass Modellierungsfehler während der manuellen Transformation komplett vermieden würden.

Eine andere Erweiterungsmöglichkeit wäre auch die Integration einer Benutzerverwaltung, in der jeder Benutzer nur die Tätigkeiten ausführen kann, die er auch ausführen soll. Dies würde auch die Daten-Sicherheit der Applikation in dem Fall erhöhen, dass einer dritten Person der Zugang zu sensiblen Daten verwehrt wird, wenn das Gerät, das die sensiblen Daten erhoben hat, in falsche Hände gerät.

Literaturverzeichnis

[1] Schmalen, K.: IDC-Studie: Deutsche Unternehmen setzen auf mobile Apps zur Verbesserung ihrer Geschäftsprozesse. http://idc.de/de/ueber-idc/press-center/56517-idc-studie-deutsche-unternehmen-setzen-auf-mobile-apps-zur-verbesserung-ihrer-geschaftsprozesse (zuletzt abgerufen am 01.10.2016)

[2] StatCounter: Marktanteile der führenden mobilen Betriebssysteme an der Internetnutzung mit Mobilgeräten in Deutschland von Januar 2009 bis Mai 2016. http://de.statista.com/statistik/daten/studie/184332/umfrage/marktanteil-der-mobilen-betriebssysteme-in-deutschland-seit-2009/ (zuletzt abgerufen am 12.08.2016)

[3] Heitkötter, H., Hanschke, S., Majchrzak, T.A. In: Evaluating Cross-Platform Development Approaches for Mobile Applications. Springer Berlin Heidelberg, Berlin, Heidelberg (2013) 120–138

[4] Schobel, J., Schickler, M., Pryss, R., Nienhaus, H., Reichert, M.: Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In Krempels, K.H., Stocker, A., eds.: 9th International Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps, Berlin, Heidelberg, Springer Berlin Heidelberg (2013) 509–518

[5] Apache Software Foundation: Embedding WebViews. https://cordova.apache.org/docs/en/latest/guide/hybrid/webviews/index.html (zuletzt abgerufen am 12.08.2016)

[6] Apache Software Foundation: Apache Cordova. https://cordova.apache.org (zuletzt abgerufen am 12.08.2016)

[7] Zhang, M., Duan, Z. In: From Business Process Models to Web Services Orchestration: The Case of UML 2.0 Activity Diagram to BPEL. Springer Berlin Heidelberg, Berlin, Heidelberg (2008) 505–510

[8] Dadam, P., Reichert, M.: The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support - Challenges and Achievements. Computer Science - Research and Development **23** (2009) 81–97

[9] Reichert, M.: Dynamische Ablaufänderungen in Workflow-Management-Systemen. PhD thesis, Universität Ulm (2000)

[10] Kreher, U.W.: Konzepte, Architektur und Implementierung adaptiver Prozessmanagementsysteme. PhD thesis, Universität Ulm (2014)

[11] Reichert, M., Dadam, P.: Enabling Adaptive Process-aware Information Systems with ADEPT2. In Cardoso, J., van der Aalst, W., eds.: Handbook of Research on Business Process Modeling. Information Science Reference, Hershey, New York (2009) 173–203

[12] Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)

[13] Leonard Richardson, S.R.: RESTful Web Services. O'Reilly Media, Inc., Heidelberg (2008)

[14] Tilkov, S., Schreier, S., Eigenbrodt, M., Wolf, O.: REST und HTTP - Entwicklung und Integration nach dem Architekturstil des Web. dpunkt.verlag, Heidelberg (2015)

[15] Hackmann, G., Haitjema, M., Gill, C., Roman, G.C. In: Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. Springer Berlin Heidelberg, Berlin, Heidelberg (2006) 503–508

[16] Battista, D., de Leoni, M., De Gaetanis, A., Mecella, M., Pezzullo, A., et al.: ROME4EU: A Web Service-Based Process-Aware System for Smart Devices. In Bouguettaya, A., Krueger, I., Margaria, T., eds.: Service-Oriented Computing – ICSOC 2008: 6th International Conference, Sydney, Australia, December , 2008, Springer Berlin Heidelberg (2008) 726–727

[17] Angelaccio, M., Buttarazzi, B., D'Ambrogio, A.: WETICE2008 Showcase Report. In: Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2008. WETICE '08. IEEE 17th. (2008) 271–273

[18] Pryss, R., Tiedeken, J., Kreher, U., Reichert, M.: Towards Flexible Process Support on Mobile Devices. In Soffer, P., Proper, E., eds.: Information Systems Evolution: CAiSE Forum 2010, Hammamet, Tunisia, June , 2010, Selected Extended Papers. Number 72, Berlin, Heidelberg, Springer (2010) 150–165

[19] Pryss, R., Tiedeken, J., Reichert, M.: Managing Processes on Mobile Devices: The MARPLE Approach. In: CAiSE'10 Demos. (2010)

[20] Pryss, R.: Robuste und kontextbezogene Ausführung mobiler Aktivitäten in Prozessumgebungen. PhD thesis, Ulm University (2015)

[21] Pryss, R., Langer, D., Reichert, M., Hallerbach, A.: Mobile Task Management for Medical Ward Rounds - The MEDo Approach. In La Rosa, M., Soffer, P., eds.: 1st International Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops. Number 132, Berlin, Heidelberg, Springer (2012) 43–54

[22] Kunze, C.P.: DEMAC: A Distributed Environment for Mobility-Aware Computing. In Ferscha, A., Mayrhofer, R., Strang, T., Linnhoff-Popien, C., Dey, A., et al., eds.: Adjunct Proceedings of the 3rd International Conference on Pervasive Computing. Volume 191 of Advances in Pervasive Computing., Wien, Österreichische Computer Gesellschaft (2005) 115–121

[23] Zaplata, S., Straßenburg, D., Wunderlich, B., Bade, D., Hamann, K., et al.: Ad-hoc Management Capabilities for Distributed Business Processes. In Abramowicz, W., Alt, R., Fähnrich, K., Franczyk, B., Maciaszek, L.A., eds.: 3rd International Conference on Business Process and Services Computing (BPSC 2010), GI-Edition, Lecture Notes in Informatics Bonn (2010) 139–152

[24] Kunze, C.P., Zaplata, S., Lamersdorf, W.: Mobile Process Description and Execution. In Eliassen, F., Montresor, A., eds.: Proceedings of the 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS 2006). Volume 4025 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2006) 32 – 47

[25] Zaplata, S., Kunze, C.P., Lamersdorf, W.: Context-based Cooperation in Mobile Business Environments. Business & Information Systems Engineering **1** (2009) 301–314

[26] Trepowski, C.D.P.: '...And suddenly the memory revealed itself'. The role of IT in supporting social reminiscence. PhD thesis, University of Trento, Italy / Catholic University of Asunción, Paraguay (2014)

[27] Peng, T., Chi, C.H., Chiasera, A., Armellin, G., Ronchetti, M., et al.: Business Process Assignment and Execution in Mobile Environments. In: Collaboration Technologies and Systems (CTS), 2014 International Conference on. (2014) 267–274

[28] Wipp, W.: Workflows on Android: A Framework Supporting Business Process Execution and Rule-Based Analysis. Master's thesis, Ulm University (2016)

[29] Schobel, J., Pryss, R., Schickler, M., Reichert, M.: A Lightweight Process Engine for Enabling Advanced Mobile Applications. In: 24th International Conference on Cooperative Information Systems (CoopIS 2016). Number 10033, Berlin Heidelberg, Springer (2016) 552–569

[30] Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer Science & Business Media, Berlin Heidelberg (2013)

[31] Dadam, P., Reichert, M., Rinderle-Ma, S., Goeser, K., Kreher, U., Jurisch, M.: Von ADEPT zur AristaFlow BPM Suite - Eine Vision wird Realität: "Correctness by Construction“ und flexible, robuste Ausführung von Unternehmensprozessen. EMISA Forum **29** (2009) 9–28

[32] Lanz, A., Weber, B., Reichert, M.: Workflow Time Patterns for Process-aware Information Systems. In: Proceedings Enterprise, Business-Process, and Information Systems Modelling: 11th International Workshop BPMDS and 15th International Conference EMMSAD at CAiSE 2010. Number 50 in LNBIP, Springer (2010) 94–107

[33] Google Inc.: Architecture Overview. `https://angular.io/docs/ts/latest/guide/architecture.html` (zuletzt abgerufen am 12.08.2016)

[34] Drifty Co.: Ionic: Advanced HTML5 Hybrid Mobile App Framework. `http://ionicframework.com` (zuletzt abgerufen am 12.08.2016)

[35] Drifty Co.: Cordova File Plugin. `https://github.com/apache/cordova-plugin-file` (zuletzt abgerufen am 18.09.2016)

[36] Schobel, J., Pryss, R., Schickler, M., Ruf-Leuschner, M., Elbert, T., et al.: End-User Programming of Mobile Services: Empowering Domain Experts to Implement Mobile Data Collection Applications. In: 5th IEEE International Conference on Mobile Services (MS 2016), IEEE Computer Society Press (2016) 1–8

[37] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: 10th International Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 371–382

[38] Damaschk, D.: Konzeption und Realisierung einer konfigurierbaren Plattform zur flexiblen Datenerhebung mittels moderner Webtechnologien. Master's thesis, Ulm University (2016)

A

Quelltexte

In diesem Anhang sind einige wichtige Quelltexte aufgeführt.

```
1  public checkAndCreateDirectory(path:string):void {
2    window.resolveLocalFileSystemURL(cordova.file.dataDirectory,
         function (directoryEntry:any) {
3              directoryEntry.getDirectory(path, {create: true},
                   function (fileEntry) {
4              }, (error)=> {
5                  console.log("cordova file " + error);
6              });
7            }, (error)=> {
8                console.log("cordova file " + error);
9            });
10   }
11  }
```

Listing A.1: Überprüfen und Erstellen der Verzeichnisinfrastruktur über das Cordova-Plugin-File.

```
1  public writeFile(path:string, fileName:string, data:string):void {
2    window.resolveLocalFileSystemURL(cordova.file.dataDirectory,
       function (directoryEntry:any) {
3       directoryEntry.getFile(path + fileName, {create: true},
          function (fileEntry) {
4         fileEntry.createWriter(function (fw) {
5             fw.onwriteend = function (e) { };
6
7             fw.onerror = function (e) { console.log('Write failed: '
                + e.toString());};
8
9                       var blob = new Blob([data], {type: this.
                          getMIMEType(fileName)});
10                      fw.write(blob);
11                  }, (error)=> {
12                      console.log("create writer" + JSON.stringify(
                          error));
13                  });
14              }, (error)=> {
15                  console.log("gerFile " + JSON.stringify(error));
16              });
17          }, (error)=> {
18              console.log("resolve file system " + JSON.stringify(
                  error) + " path " + (cordova.file.dataDirectory +
                  path));
19      });
20    }
```

Listing A.2: Die Methode `writeFile` speichert eine Media-Datei in den übergebenen Pfad mit dem übergebenen Dateinamen.

```
1  private handleNode():boolean {
2     switch (this._actualNode.type) {
3        case NodeType.START:
4          return this.handleStartNode();
5        case NodeType.END:
6          return this.handleEndNode();
7        case NodeType.NORMAL:
8          return this.handleNormalNode();
9        case NodeType.XOR_SPLIT:
10         return this.handleXORSplit();
11       case NodeType.XOR_JOIN:
12         return this.handleXORJoin();
13       case NodeType.AND_SPLIT:
14         return this.handleANDSplit();
15       case NodeType.AND_JOIN:
16         return this.handleANDJoin();
17       case NodeType.LOOP_START:
18         return this.handleLoopStart();
19       case NodeType.LOOP_END:
20         return this.handleLoopEnd();
21       default:
22         this.handleError();
23    }
24    return false;
25  }
```

Listing A.3: Knotenunterscheidung während der Prozess-Traversierung.

```
1  public processNode(node:Node):void {
2      if (node.hasGUI()) {
3           //send data to Component
4           this._communicationService.toComponent(JSON.stringify(node.
                executableBusinessProcess));
5        }
6   }
```

Listing A.4: Ein Executable Business Process wird über den CommunicationService an die Benutzeroberfläche gesendet.

```
this.subscription = this.communicationService.
    getComponentTargetStream().subscribe(data =>{
                this.handleData(data);
})
```

Listing A.5: Empfang des durch den Runtime Manager gesendeten Executable Business Process.

```
private handleXORSplit():boolean {
    if (this._actualNode.executableBusinessProcess !== undefined) {
      let xor:Node = this._actualNode;

      //Step 1: process executable business process from xor node
      let result = this._runtimeManager.processXORNode(xor);

      //Step 2: do dead path elemination
      this.initDeadPathElimination(result);

      //Step 3: split node has been handled so it is completed and
          terminated
      this.doEventLogEntry(ActivityState.COMPLETED);

      //Step 4: take first node of decision-branch as next node
      this._actualNode = xor.getBranchByIndex(result)[0];

      //Step 5: perform eventlogEntry for new actualNode
      this.doEventLogEntry(ActivityState.ACTIVATED);

      //Step 6: process this new actualNode
      return this.getNextNode();
    } else {
      this.handleError();
    }
  }
```

Listing A.6: XOR-Knoten Behandlungsfunktion im Instance Manager.

```
1  public processXORNode(xor:Node):any {
2      //collect all input DataElements
3     let inputDataElements:Array<string> = new Array<string>();
4     for (let k in xor.connectorParameterMapping) {
5       if (xor.connectorParameterMapping[k].type === "READ") {
6         inputDataElements.push(xor.connectorParameterMapping[k].
              parameterName);
7      }
8     }
9     //call handleData() function of executable business process
          implementation, identified by implemtationClass property
10    return this._communicationService.handleData(xor.
          implementationClass, xor.executableBusinessProcess, this.
          _answerStore.getObjectsFormIdentifiers(inputDataElements));
11   }
```

Listing A.7: Behandlungsfunktion für einen XOR-Knoten mit automatischem Executable Business Processes.

```
1  private deadPathElimination(livePath:number):void {
2     for (let k = 0; k < this._actualNode.getBranchSize(); k++) {
3       if (k != livePath) { //get branch from actualNode
4         let branch = this._actualNode.getBranchByIndex(k);
5         //if node in branch is a split, we have to traverse the whole
              split-branches and mark them as skipped
6         for (let key in branch) {
7          if (branch[key].type === NodeType.XOR_SPLIT || branch[key].
               type === NodeType.AND_SPLIT ) {
8            this.deadPathElimination(branch[key]);
9            //do eventlog entry for splitNode
10           this.doEventLogEntryWithNodeID(ActivityState.SKIPPED,
                 branch[key].nodeID);
11         } else {   //do eventlog entry for normal node
12           this.doEventLogEntryWithNodeID(ActivityState.SKIPPED,
                 branch[key].nodeID);
13         }
14        }
15     }}}
```

Listing A.8: Dead Path Elimination für die Zweige eines Split-Knotens, die nicht mehr zur Ausführung kommen.

```
1 public handleData(implementationClass:string, ebp:Object, data:Object
      , ...params:any[]):any {
2     let index = undefined;
3     index = this.implementationClassToService[implementationClass];
4     if (index !== undefined) {
5       return this.services[index].handleData(ebp, data, params);
6     }
7     return 0;
8   }
```

Listing A.9: Die Methode `handleData` des CommunicationService selektiert den richtigen Dienst und führt anschließend die gleichnamige Funktion des Dienstes aus, die den automatischen Executable Business Process behandelt.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Name: Julian Sebastian Schregle Matrikelnummer: 751794

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Julian Sebastian Schregle