Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Entwicklung eines Modellierungswerkzeuges für Quanten-Workflows

Felix Belz

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dr. h. c. Frank Leymann
Betreuer/in:	Martin Beisel, M.Sc. Benjamin Weder, M.Sc.
Beginn am:	4. November 2022
Beendet am:	4. Mai 2023

Kurzfassung

Mit zunehmender Leistungsfähigkeit finden Quanten-Computer und Quantenalgorithmen in immer mehr Bereichen Anwendung, um dort Probleme effizienter als klassische Computer zu lösen. Bei Quantenalgorithmen handelt es sich häufig um hybride Algorithmen, die sowohl auf Quanten-Computern als auch auf klassischen Computern ausgeführt werden. Quanten-Algorithmen und klassische Algorithmen werden in Quantenanwendungen orchestriert. Dafür können Quanten-Workflows verwendet werden, um unter anderem die Unterbrechbarkeit und die Skalierbarkeit der Quantenalgorithmen zu erhöhen. Durch die unterschiedlichen Anwendungsbereiche und Benutzergruppen der Quantendomäne ist bei der Modellierung von Quanten-Workflows die graphische Modellierung besonders wichtig, um die fachlichen und technischen Voraussetzung möglichst gering zu halten. Existierenden Modellierungswerkzeugen fehlt es jedoch häufig an Unterstützung der graphischen Modellierung von Quanten-Workflows und des Datenflusses von Quanten-Workflows. Der in dieser Arbeit entwickelte Quantum Workflow Modeler ermöglicht eine explizite, graphische Modellierung des Datenflusses und von Datentransformationen, durch Erweiterung des BPMN Standards [OMG11]. Zudem kann er Plugin-basiert erweitert werden, um quantenspezifische Modellierungselemente zu integrieren. In dieser Arbeit wurden bereits das Datenfluss-, QuantME, PlanQK und QHAna Plugin in den Modeler integriert, um die Modellierung von Quanten-Workflows zu unterstützen. Um die Portabilität der mit diesen Erweiterungen modellierten Workflows zu gewährleisten, bietet der Modeler Schnittstellen, um erweiterte Workflows in BPMN konforme Workflows zu transformieren. Der Quantum Workflow Modeler wurde als HTML Web-Komponente implementiert, um in andere Web-Anwendungen, unabhängig des verwendeten Frameworks, integriert werden zu können.

Abstract

With increasing performance, quantum computers and quantum algorithms are being used in more and more areas to solve problems more efficiently than classical computers. Quantum algorithms are often hybrid algorithms that run on both quantum and classical computers. Quantum and classical algorithms are orchestrated in quantum applications. For this purpose, quantum workflows can be used to increase the interruptibility and scalability of quantum algorithms, among others. Due to the different application domains and user groups of the quantum domain, graphical modeling is particularly important for modeling quantum workflows in order to minimize the domain-specific and technical prerequisite. However, existing modeling tools often lack support for graphical modeling of quantum workflows and data flow of quantum workflows. The Quantum Workflow Modeler developed in this thesis enables explicit, graphical modeling of data flow and data transformations, by extending the BPMN standard. Moreover, it can be extended in a plugin-based way to integrate quantum-specific modeling elements. In this work, the DataFlow, QuantME, PlanQK, and QHAna plugin have already been integrated into the Modeler to support quantum workflow modeling. To ensure portability of workflows modeled with these extensions, the Modeler provides interfaces to transform extended workflows into BPMN compliant workflows. The Quantum Workflow Modeler has been implemented as an HTML web component to integrate with other web applications, regardless of the framework used.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

1. Einleitung

Mit der zunehmenden Verbesserung der Leistungsfähigkeit von Quantencomputern finden Quantenalgorithmen in immer mehr Bereichen Anwendung, um dort Probleme zu lösen, die mit klassischen Computern und Algorithmen nicht oder nicht so schnell gelöst werden können, wie beispielsweise in der Materialwissenschaft [BBMC20] oder in der Domäne des Machine Learnings [RSM+20]. Dadurch steigt die Bedeutung von Quantenalgorithmen und deren Entwicklung. Bei Quantenalgorithmen handelt es sich häufig um hybride Algorithmen, die teilweise auf einem Quanten-Computer und teilweise auf einem klassischen Computer ausgeführt werden [WBLZ21]. Die klassischen und quantenspezifischen Teile eines Quantenalgorithmus müssen für die Ausführung des Alogrithmus kombiniert bzw. orchestriert werden. Dafür eignen sich Workflows [LR99], die die Modellierung und Ausführung von Quantenanwendungen ermöglichen. Die Verwendung von Workflows erhöht die Skalierbarkeit, Portabilität, Unterbrechbarkeit, Reproduzierbarkeit und Wiederverwendbarkeit von Quantenanwendungen. Quantenanwendungen und deren Bestandteile können in sogenannten Quanten-Workflows orchestriert werden [WBLW20]. Um hier die Modellierung zu vereinfachen, sind spezifische Modellierungselemente der Quantendomänen nötig, die spezielle Aufgaben in einem Quanten-Workflow modellieren, wie beispielsweise Pre- und Post-Processing Tasks [WBLW20]. Die Modellierung von Quantenanwendungen umfasst neben der Modellierung der Quantenalgorithmen auch die Modellierung verschiedener Datenformate und Datentransformationen. Dies ist in der Quanten-Computing-Domäne, aufgrund der großen Heterogenität der verwendeten Programmiersprachen, SDKs und Plattformen und der rapiden Weiterentwicklung, besonders herausfordernd [LBF+20]. Darüber hinaus gibt es unterschiedliche Benutzergruppen, die jeweils mit eigenen Datenformat arbeiten, was zu sehr heterogenen Datensätzen führt. Zur Modellierung von Quanten-Workflows ist demnach die Modellierung des Datenflusses besonders wichtig. Um hier die fachlichen und technischen Kenntnisse möglichst gering zu halten, ist es darüber hinaus wichtig, dass der Datenfluss graphisch modelliert werden kann, um keine Kenntnisse eines spezieller Syntax, den eine textuelle Modellierungssprache erfordert, für die Modellierung vorauszusetzen.

Existierenden Modellierungswerkzeugen für Quanten-Workflows fehlt es häufig an Unterstützung der graphischen Modellierung des Datenflusses. Viele Modellierungswerkzeuge verwenden ein eigenes Konzept zur Modellierung des Datenflusses, das mit anderen Modellierungswerkzeugen nicht kompatibel ist und die Portabilität der modellierten Workflows einschränkt [CAMP723; SPI23]. Existierende Modellierungswerkzeuge bieten zudem meist keine Unterstützung zur Modellierung von spezifischen Elementen aus der Quantendomäne und müssen entsprechend erweitert werden. Vorhandene Erweiterungen zur Modellierung von Quanten-Workflows konzentrieren sich meist nicht auf alle relevanten Aspekt der Modellierung von Quanten-Workflows. Beispielsweise führt das QuantME Transformation Framework [WBLW20] Pre- und Postprocessing Tasks zur Modellierung von Quantenalgorithmen ein, bietet aber keine Unterstützung für alle Aspekte von Quanten-Workflows. Um Quanten-Workflows zu modellieren, müssen somit verschiedene Erweiterungen kombiniert werden. Um diese Herausforderungen zu bewältigen, wurde der in dieser Arbeit vorgestellte Quantum Workflow Modeler entwickelt. Dieses Modellierungswerkzeug ermöglicht

eine explizite, graphische Modellierung des Datenflusses und die dynamische Integration und Erweiterung von spezifischen Modellierungselementen aus der Quanten-Computing-Domäne, um zukünftige Entwicklungen im Quanten-Computing vereinfacht abbilden zu können. Des Weiteren ermöglicht der Modeler die Transformation der modellierten erweiterten Workflows, die Quanten-Computing spezifische Elemente enthalten, in native Workflows. Dadurch bleibt die Portabilität der Workflow-Modelle erhalten und die Ausführung kann in existierenden Workflow Engines erfolgen. Um den Modeler in möglichst vielen Anwendungen und Plattformen verwenden zu können, wurde er als Web-Komponente entwickelt, sodass er in andere Websites mit minimalen Anpassungen integriert werden kann, wie beispielsweise in die PlanQK Plattform [PQK23].

Im Folgenden werden zunächst die technischen und fachlichen Grundlagen in Kapitel 2 ausgeführt, die für das Verständnis der nachfolgenden Konzepte wichtig sind. Anschließend wird in Kapitel 3 der aktuelle Stand der Forschung diskutiert, bevor in Kapitel 4 die Anforderungen an den Quantum Workflow Modeler erhoben werden. Nach den Anforderungen werden die Konzepte, die dem Modeler zugrunde liegen, in Kapitel 5 erläutert. In Kapitel 6 folgt die Beschreibung der Implementierung des Modellierungswerkzeuges. Abschließend werden in Kapitel 8 die vorgestellten Konzepte zusammengefasst und Erweiterungsmöglichkeiten umrissen.

2. Grundlagen

In diesem Kapitel werden die fachlichen und technischen Grundlagen erläutert, die für das Verständnis dieser Arbeit relevant sind. Zunächst werden die Grundlagen der Modellierung von Workflows betrachtet. Anschließend werden die grundlegenden Modellierungselemente des Business Process Model and Notation (BPMN) Standards, der de-facto Standard zur Modellierung von Workflows [HW11], eingeführt.

2.1. Workflows

Ein Workflow ist die Instanz eines Workflow-Modells. Dieses Modell ist der computergestützte Teil eines Prozessmodells, das einen Geschäftsprozess abbildet [LR99]. Das Workflow-Modell definiert eine Menge von Arbeitsschritten und deren partielle Ordnung, den Kontrollfluss. Ein Arbeitsschritt wird durch eine *Activity* repräsentiert. Eine Activity kann entweder elementar oder zusammengesetzt sein. Eine elementare Activity besteht aus einer Menge von primitiven Aktionen, die nicht in weitere Operationen unterteilt werden. Eine zusammengesetzte Activity beschreibt dagegen ein separates Workflow-Modell, das wiederum aus eigenen Activities besteht. Activities können wiederverwendet werden, um denselben Arbeitsschritt in verschiedenen Workflow-Modellen zu modellieren [Ell99].Neben dem Kontrollfluss kann in einem Workflow-Modell der Datenfluss, d.h. die Daten des Geschäftsprozesses modelliert werden, um den Input bzw. Output von Activities zu spezifizieren. Daten werden als *DataElements* modelliert, bzw. falls es sich um eine Sammlung von DataElements handelt, durch *Container* [LR99].

Ein Workflow Management System verwaltet Workflows. Dies umfasst auch eine Workflow Engine, über die ein Workflow-Modell, bzw. ein Workflow ausgeführt werden kann. Für die Ausführung unterscheidet man zwischen zwei Arten von Activities: automatischen und manuellen Activities. Für automatische Activities wird im Workflow-Modell ein ausführbares Computerprogramm spezifiziert. Dieses Programm wird von der Workflow Engine gestartet, sobald die Ausführung des Workflows an der entsprechenden Activity angelangt ist. Manuelle Activities dagegen werden von Benutzern gestartet, die dann die eigentliche Aktion ausführen. [LR99].

2.2. BPMN

In diesem Abschnitt werden die grundlegenden Konzepte des Business Process Model and Notation (BPMN) Standards beschrieben, die für diese Arbeit relevant sind. Alle in diesem Abschnitt beschriebenen Konzepte sind, sofern nicht anders gekennzeichnet, dem BPMN Standard Version 2.0 entnommen. Um näher am BPMN Standard zu bleiben, werden die Namen der Konzepte nicht ins Deutsche übersetzt, sondern auf Englisch verwendet. Insbesondere die BPMN Konzepte zur Modellierung des Datenflusses werden im Folgenden näher betrachtet.

Der BPMN Standard spezifiziert eine einheitliche Notation für die Modellierung von Geschäftsprozessen, die mit dem Ziel entwickelt wurde, sowohl vom Analysten, der den Geschäftsprozess modelliert, als auch vom Entwickler, der für die Implementierung der Activities zuständig ist, verstanden zu werden. Der Standard definiert eine graphische Notation, der ein XML Schema zugrunde liegt und ist der de-facto Standard zur Modellierung von Geschäftsprozessen [HW11]. Ein Prozess wird als BPMN Workflow modelliert, der aus den folgenden Elementen besteht:

Der Kontrollfluss hat einen Beginn, modelliert durch sogenannte *StartEvents*, und ein durch sogenannte *EndEvents* modelliertes Ende. Dazwischen wird der Ablauf des Prozesses dargestellt. Einzelne Schritte im Prozess werden durch *Activities* modelliert. Ist dieser Schritt nicht weiter unterteilbar, handelt es sich um eine atomare Activity, einen sogenannten *Task*. Über den Typen des Tasks kann spezifiziert werden, um welche Art von Arbeitsschritt es sich handelt. Falls in diesem Schritt beispielsweise eine Anfrage an einen Webserver geschickt wird, kann dies durch einen sogenannten *ServiceTask* repräsentiert werden. Tasks, die ein Benutzer mit Hilfe von Software bearbeitet, können durch *UserTasks* modelliert werden. Activities können auch weiter unterteilbar sein und einen separaten Prozess repräsentieren, sogenannte *SubProcesses*. Im Geschäftsprozess auftretende Ereignisse, wie beispielsweise der Erhalt einer Nachricht, können durch *IntermediateEvents* modelliert werden. Ein IntermediateEvent besitzt, ähnlich eines Tasks, einen Typ, über den die Details des Ereignisses spezifiziert werden. Die Verzweigungen des Kontrollflusses in einzelne Kontrollflusspfade, wie beispielsweise Schleifen oder Bedingungen, werden durch sogenannte *Gateways* modelliert. Um beispielsweise den Kontrollfluss aufzuteilen, um eine parallele Ausführung von Activities zu ermöglichen, kann ein *ParallelGateway* verwendet werden. Es teilt den eingehenden Kontrollfluss in parallel ausgeführte ausgehende Pfade auf. Falls es mehrere eingehende Kontrollflusspfade gibt, wird die Ausführung der eingehenden Pfade am ParallelGateway synchronisiert. Das *ExclusiveGateway* dagegen synchronisiert seine eingehenden Kontrollflusspfade nicht, sodass der aus dem ExclusiveGateway ausgehende Kontrollfluss weiterhin parallel sein kann, abhängig vom eingehenden Kontrollfluss. Activities, Events und Gateways werden durch *SequenceFlows* miteinander verbunden, um die Ausführungsreihenfolge bzw. den Kontrollfluss zu definieren. Detaillierte Informationen zu den einzelnen Typen von Activities, Events und Gateways werden an dieser Stelle nicht näher erläutert, da sie für das Verständnis dieser Arbeit nicht notwendig sind und können dem BPMN Standard [OMG11] entnommen werden.

Der Datenfluss eines Geschäftsprozesses wird über *DataObjects* und *DataStores* definiert. Dabei handelt es sich um Elemente, die graphisch im Workflow modelliert werden. Über *DataAssociations* werden diese Elemente mit Elementen des Kontrollflusses verbunden. Mit DataObjects werden Daten modelliert, die nur für einen Prozess benötigt werden. Wird der Prozess beendet, sind die Daten nicht mehr verfügbar. In DataObjects gespeicherte Daten sind im Workflow verfügbar, solange es mindestens eine aktive Activity gibt, die mit ihnen über eine DataAssociation verbunden ist. Sind alle mit dem DataObject verbundenen Activities beendet, ist das DataObject im Prozess nicht mehr verfügbar. Daher können lediglich der Elternprozess bzw. SubProcess, die Elemente im selben Prozess und deren Kindelemente auf ein DataObject zugreifen. Mit einer Activity verbundene DataObjects repräsentieren den Input bzw. Output dieser Activity. Die Struktur der durch ein DataObject modellierten Daten kann durch ein separates *ItemDefinition*-Element angegeben werden, über das beispielsweise ein XML Schema für den Aufbau der Daten definiert werden kann. Die Verwendung von ItemDefinitions ist im BPMN Standard optional. Die genauen Details, wie Daten bzw. Informationen in DataObjects gespeichert werden können, ist nicht näher spezifiziert und abhängig von der jeweiligen Realisierung des BPMN Standards.

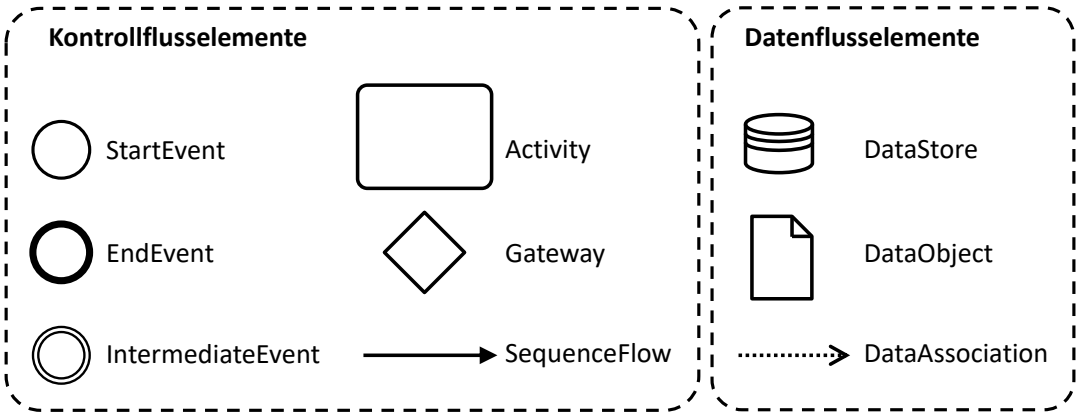

Abbildung 2.1.: Graphische Notation der BPMN Kontrollflusselemente.

Eine Activity kann sogenannte *InputSets* und *OutputSets* definieren. Diese Mengen bestehen aus ItemDefinitions, die angeben, welche Daten in welchem Format als Input bzw. Output des Prozesses zu erwarten sind. Die ItemDefinition einer Activity und die eines, mit ihr über eine DataAssociation verbundenen DataObjects, müssen übereinstimmen oder es muss eine sogenannte Expression für die DataAssociation definiert werden, um die Transformation der ItemDefinitions anzugeben. Eine Expression beschreibt als natürlichsprachiger Text einen Ausdruck, der nicht ausgeführt werden kann. Die Beschreibung einer Transformation durch eine Expression ist daher eine Dokumentationsmaßnahme, die für die Ausführung des Workflows unerheblich ist. Über eine Expression in einer DataAssociation können somit nicht ausführbare Datentransformationen zu Dokumentationszwecken modelliert werden.

Falls Informationen auch nach der Ausführung eines Prozesses abrufbar sein sollen, wird dies über einen DataStore modelliert. Er repräsentiert einen persistenten Datenspeicher, wie beispielsweise eine Datenbank. Über DataAssociations zwischen einem DataStore und einer Activity kann der Datenfluss in bzw. aus diesem Datenspeicher modelliert werden. Der Aufbau des DataStores kann, wie bei DataObjects, über ItemDefinitions spezifiziert werden.

Die graphische Repräsentation der eben beschriebenen Kontroll- bzw. Datenflusselemente ist in Abbildung 2.1 dargestellt.

Mit BPMN modellierte Workflows können in Workflow Engines, die mit dem BPMN Standard konform sind, ausgeführt, werden. Die Workflow Engine arbeitet den Kontrollfluss ab und ruft für die modellierten Activities entsprechende Programme auf, die je nach Task Typ unterschiedlich spezifiziert werden. Wie genau die Programme definiert werden, ist abhängig von der konkreten Workflow Engine. Die genaue Funktionsweise des Datenflusses während der Ausführung ist im BPMN Standard nicht näher spezifiziert. Daher verbleibt es in der Verantwortung der Workflow Engine, entsprechende Erweiterungen zu definieren und den Datenfluss zur Laufzeit zu verwalten.

In Abbildung 2.2 ist ein Beispiel für einen BPMN Workflow, mit den in diesem Abschnitt beschriebenen Elementen, abgebildet. *Task1* lädt Daten aus einem DataStore und verwendet diese als Input. Als Output produziert er ein DataObject, das als Input für *Task2* dient. Nach der Ausführung von *Task1* wird der Kontrollfluss am ParallelGateway auf zwei parallele Pfade aufgeteilt, sodass *Task2* und *Task3* parallel ausgeführt werden. Nach der Ausführung von *Task3* pausiert

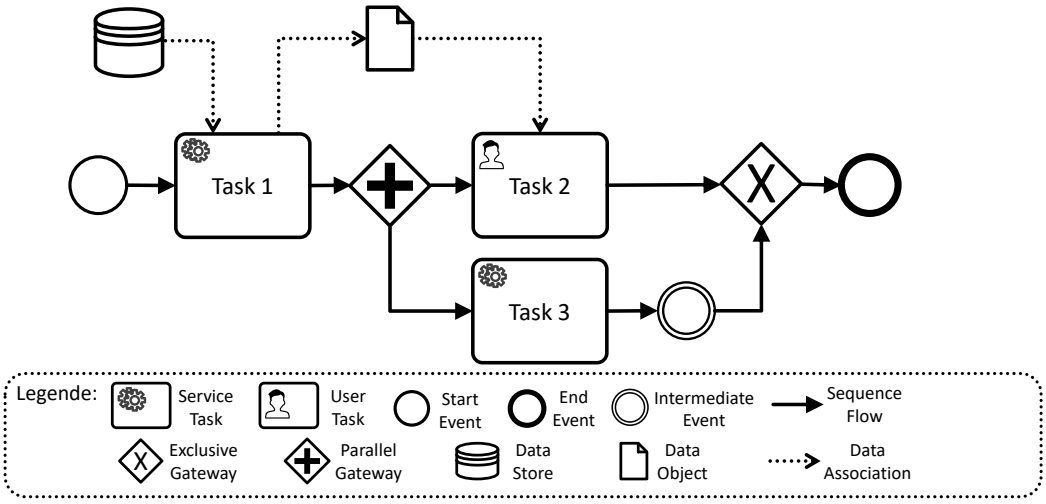

Abbildung 2.2.: Beispiel eines BPMN Workflows.

die Ausführung dieses Kontrollflusspfades bis das Event des IntermediateEvents eintritt. In dem ExclusiveGateway werden die beiden parallelen Kontrollflüsse wieder zu einem zusammengeführt, bevor der Workflow im EndEvent sein Ende erreicht.

3. Verwandte Arbeiten

Für die Modellierung von Quantenanwendungen und Quanten-Workflows gibt es mehrere Modellierungsansätze. Ein grafischer Modellierungsansatz für Quanten-Circuits ist die Quantum Modelling Extension (QuantME) [WBLS21; WBLW20]. Sie ermöglicht die Modellierung von Quantenalgorithmen einschließlich Pre- und Post-Processing Tasks durch die Erweiterung imperativer Workflow-Sprachen. Der Fokus von QuantME liegt auf der Modellierung aller erforderlichen Verarbeitungsschritte eines Quantenalgorithmus. Um die Portabilität der mit der QuantME Erweiterung modellierten Workflows nicht einzuschränken, wird eine Transformationsmethode eingeführt, die den erweiterten Workflow in einen nativen Workflow überführt. MODULO (Modeling, Transformation, and Deployment of Quantum Workflows) [WBL21] erweitert diesen Ansatz, indem es die Modellierung von Quanten-Workflows, ihre Umwandlung in eigenständige Pakete, die Bereitstellung der Pakete und die Ausführung dieser Pakete unterstützt. Eine prototypische eigenständige Implementierung der QuantME-Erweiterungen ist das QuantME Transformation Framework [QME23]. Es basiert auf einer BPMN 2.0 Erweiterung, die die QuantME Erweiterung umsetzt. Während das QuantME Transformation Framework Unterstützung für die Modellierung von Quantenalgorithmen bietet, beinhaltet es keine Modellierungsunterstützung für Datentransformationen. Es gibt auch Ansätze, Quantum-Workflows zu generieren, statt sie zu modellieren: In [VBLW22] wird ein Ansatz vorgestellt, Quanten-Workflows basierend auf existierenden, text-basierten Implementierungen von Quantenalgorithmen zu erzeugen.

Modelliert werden können Quantenanwendungen mittels Orquestra [ORQ23], einer Software-Plattform, die eine Python-basierte Workflow-Sprache ohne graphische Notation bereitstellt. Sie kann zur Modellierung und Ausführung sogenannter quantum-enabled Workflows verwendet werden. Die Workflows werden dabei in Python programmiert, wobei eigene Tasks in Python definiert werden können. Es können auch Tasks aus externen Bibliotheken, wie die von Zapata [ZAP23] entwickelten Algorithmen, verwendet werden, um den Workflow zu definieren.

Um BPMN Workflows zu modellieren, existieren einige Modellierungswerkzeuge. Diese Werkzeuge unterscheiden sich unter andrem im Umfang der Unterstützung des BPMN Standards und in der Art der Modellierung des Datenflusses.

Camunda bietet State-of-the-Art Modellierungswerkzeuge, mit denen BPMN Workflows grafisch modelliert und ausgeführt werden können [CAMP723]. Camunda verwendet in seinen Modellierungswerkzeugen eine Erweiterung des BPMN Standards, die die graphische Notation des Standards realisiert und zusätzliche Attribute hinzufügt. Die Verwendung von DataObjects und DataStores zur Modellierung des Datenflusses wird zwar von dem Camunda Workflow Modeler unterstützt, allerdings sind diese Elemente für die Ausführung des Workflows unerheblich. Stattdessen lässt sich der Datenfluss in Camunda über die in der Erweiterung eingeführten Prozessvariablen modellieren. Diese können jedoch nicht grafisch modelliert werden, sondern müssen textuell als Attribute definiert werden [CAMP723].

SpiffWorkflow [SPI23] ist ein Python basiertes Modellierungswerkzeug, über das BPMN Workflows grafisch erstellt und ausgeführt werden können. SpiffWorkflow verwendet ein Datenkonzept, bei dem Variablen automatisch an den nachfolgenden Task weitergeleitet werden. Diese sogenannte TaskData ist einem Task zugeordnet und kann über Skripte verwaltet werden, um zu steuern, welche Daten der Task als Input bekommt und welche er als Output erzeugt [FUN22]. Somit folgt der Datenfluss automatisch dem Kontrollfluss des Workflows. Daten, die nur von einigen Tasks benötigt werden, können über DataObjects modelliert werden, wobei über DataAssociations definiert wird, in welchen Tasks die Daten verfügbar sein sollen. SpiffWorkflow erlaubt somit eine implizite, graphisch nicht explizit erkennbare Modellierung des Datenflusses, die sich optisch nicht vom Kontrollfluss unterscheidet.

Flowable [FAG23] ist eine leichtgewichtige Workflow Engine, über die sowohl BPMN Workflows grafisch modelliert, als auch deployt und ausgeführt werden können. Flowable verwendet eine eigene BPMN Erweiterung, die Flowable BPMN Extension, die den BPMN Standard realisiert und dahingehend erweitert, dass die Verwendung einiger BPMN Elemente vereinfacht wurde, durch die Definition von neuen Elementen. Der Datenfluss wird bei Flowable, ähnlich dem bei Camunda, durch Prozessvariablen definiert. Diese Variablen können als Attribute dem Workflow bzw. seinen Elementen hinzugefügt werden. Flowable unterstützt zusätzlich die Modellierung des Datenflusses über DataObjects. Der Inhalt der DataObjects kann lediglich durch einige wenige XSD Typen und nicht beliebig vom Modellierer definiert werden. DataObjects werden vor der Ausführung von Flowable auf Prozessvariablen abgebildet [FAG23].

Um die Modellierung des Datenflusses in Service-Choreographien [DKB08] zu verbessern, werden von Hahn et al. [HBKL18] data-aware Service-Choreographien eingeführt. Data-aware Service-Choreographien sollen die data-awareness, d.h. das Bewusstsein für den Datenfluss, in Service-Choreographien während der Modellierung und Laufzeit verbessern. Die Transparent Data Exchange Middleware (TraDE) [HBLW17] ist eine Middleware für Service-Choreographien zur Modellierung des Datenaustauschs von Cross-Partner Datenobjekten und Cross-Partner Datenflüssen, um die Möglichkeiten der Datenmodellierung zu verbessern. Um Daten in Service-Choreographien zu transformieren, führen Hahn et al. [HBL+18] ein neues Modellierungselement ein: das Data Transformation (DT) Element. Dieses Element erlaubt es, Transformationen direkt im Datenfluss zu modellieren, unabhängig vom Kontrollfluss. Dieses DT Element wird dann von der TraDE Middleware verarbeitet, die die eigentliche Transformation übernimmt. Treka et al. [TVS09] führen Anti-Patterns für die Modellierung des Datenflusses in Workflows ein. Diese Anti-Patterns werden formal definiert und ermöglichen die Analyse des Datenflusses eines Workflows. Dadurch sollen häufige Fehler in der Datenflussmodellierung systematisch erkannt und behoben werden.

4. Anforderungen an den Quantum Workflow Modeler

In diesem Kapitel werden die Anforderungen an den Quantum Workflow Modeler beschrieben. Ziel des Modelers ist es, seinen Benutzern zu ermöglichen, BPMN Workflows in der Quantendomäne zu modellieren. Insbesondere soll die explizite Modellierung des Datenflusses erleichtert werden, damit der Nutzer die im Workflow benötigten Daten einfach spezifizieren kann. Um weiterhin die modellierten Workflows in existierenden BPMN Workflow Engines ausführen zu können, sollen die quantendomänenspezifischen Workflows in BPMN konforme Workflows transformiert werden können, wie im QuantME Transformation Framework [WBLW20]. Da es sich bei den Endanwendern des Modelers um Benutzer unterschiedlicher, nicht zwangsläufig Informatik verwandter, Fachrichtungen handelt, wie der Chemie, Physik etc., soll der Modeler leicht zu benutzen sein, ohne dass spezifische Informatikkenntnisse erforderlich sind. Im Folgenden werden die Anforderungen detaillierter ausgeführt.

1. **Graphische Modellierung von ausführbaren Workflows in der Quantendomäne**
   Der Modeler soll es dem Modellierer ermöglichen, Workflows mit quantendomänenspezifischen Elementen zu modellieren. Um dabei die notwendigen fachlichen Modellierungskentnisse möglichst gering zu halten, soll der Workflow graphisch durch die im BPMN Standard definierte Notation modelliert werden können, um das Lernen einer textuellen Syntax zu vermeiden.

2. **Graphische Modellierung des Datenflusses**
   Der Datenfluss des Workflows soll explizit und graphisch modelliert werden können. Die Modellierung soll den Benutzern dadurch erleichtert werden. Der modellierte Datenfluss soll während der Ausführung des Workflows in einer Workflow Engine verfügbar sein und nicht nur zu Dokumentationszwecken dienen. Die Modellierung des Datenflusses beinhaltet, neben der Modellierung von Datenelementen und deren Beziehung zu Activities, ebenfalls die Modellierung von Datentransformationen.

3. **Portabilität der modellierten Workflows**
   Um die Portabilität der modellierten Workflows nicht einzuschränken, soll der modellierte Workflow von einer BPMN konformen Workflow Engine ausgeführt und in anderen BPMN Editoren geöffnet werden können. Aus diesem Grund soll der Modeler die Möglichkeit bieten, den mit Erweiterungen zum BPMN Standard modellierten Workflow in ein mit dem BPMN Standard konformes Modell zu transformieren, wobei die Funktionalität, sowohl des Kontroll- als auch des Datenflusses, erhalten bleibt.

4. **Integrierbarkeit des Modelers in Web-Anwendungen**
   Der Modeler soll als Komponente in verschiedenen Plattformen und Web-Anwendungen eingebaut werden können. Dabei soll es möglich sein, die Komponente in beliebige Web-

Frameworks zu integrieren, wie beispielsweise Vue oder Angular. Die Komponente soll alle benötigten Bibliotheken enthalten und keine Abhängigkeiten zu externen, nicht in der Komponente enthaltenen, Bibliotheken besitzen.

5. **Erweiterbarkeit**
   Um weitere Modellerweiterungen des BPMN Standards in den Quantum Workflow Modeler integrieren zu können, sollen zukünftige Erweiterungen Plugin-basiert in den Modeler integriert werden können. Die Architektur und Implementierung des Workflow Modelers soll dies ermöglichen. Eine Erweiterung kann neben einem Metamodell, das die Elemente der Erweiterung definiert, ebenfalls Erweiterungen der Funktionalität des Editors und der graphischen Notation umfassen. Zum Beispiel soll eine Erweiterung eine zusätzlich Funktion beinhalten können, mit der die Verwendung der Modellierungselemente dieser Erweiterung auf ihre korrekte Verwendung hin überprüft werden kann.

6. **Intuitive Bedienbarkeit**
   Da die Zielgruppe des Modelers nicht zwangsläufig über Informatik- oder Programmierkenntnisse verfügt, soll der Modeler intuitiv, graphisch und ohne Programmierkenntnisse benutzbar sein. Die Benutzeroberfläche ist entsprechend zu gestalten.

5. Quantum Workflow Modeler Konzepte

In diesem Kapitel wird der Quantum Workflow Modeler vorgestellt. Dieser ist ein Modellierungswerkzeug, mit dem BPMN Workflows mit spezifischen Modellierungskonstrukten der Quanten-Computing-Domäne zu modellieren. Er kann als Web-Komponente in andere Web-Anwendungen integriert werden, um in verschiedenen Umgebungen und Plattformen die Modellierung von Quanten Workflows zu ermöglichen. Um Quanten-Computing spezifische BPMN Elemente in der Modellierung verwenden zu können, kann der Modeler Plugin-basiert erweitert werden. So können BPMN Erweiterungen, wie beispielsweise die in [WBLW20] beschriebenen Konzepte, im Modeler übernommen werden, damit zukünftige Entwicklungen im Quanten-Computing einfach zu integrieren sind (siehe Abschnitt 5.1). Um die Wiederverwendbarkeit von modellierten Elementen zu ermöglichen und dadurch die notwendigen Fachkenntnisse des Modellierers zu reduzieren, erlaubt der Modeler den Import von Konfigurationen für BPMN Elemente, über die Attribute eines BPMN Elementes mit spezifischen Werten belegt werden können, um beispielsweise die Implementierungsdetails eines ServiceTasks zu konfigurieren. Dieses Konzept wird in Abschnitt 5.2 näher beschrieben. Es erlaubt die Wiederverwendbarkeit von konkreten, bereits konfigurierten Elementen, um beispielsweise denselben Service in unterschiedlichen Workflows zu modellieren, ohne dass der Modellierer die genauen Details in jedem Workflow erneut konfigurieren muss. Zusätzlich zu den Quanten-Computing-Erweiterungen enthält der Quantum Workflow Modeler eine BPMN Erweiterung, über die der Datenfluss explizit graphisch modelliert werden kann, sodass auch während der Ausführung des Workflows ein Zugriff auf die Daten möglich ist, beschrieben in Abschnitt 5.3.

Die technischen Details und die Implementierung der in diesem Kapitel beschriebenen Konzepte, sowie die Architektur des Quantum Workflow Modelers werden in Kapitel 6 erläutert.

5.1. Plugin-basierte Integration von BPMN Modellerweiterungen

Um die dynamischen Entwicklungen in der Quanten-Computing-Domäne im Modeler abbilden zu können, kann er Plugin-basiert erweitert werden, um neue BPMN Erweiterungen zu integrieren. Die Erweiterung ist dabei in ihrem Umfang nur minimal eingeschränkt und kann neben der eigentlichen Definition von neuen Modellierungselementen und deren graphischer Notation zusätzliche Funktionen, wie die Analyse des Workflows oder die Kommunikation mit externen Endpoints, umfassen. Dazu mehr in Abschnitt 5.1.1. Um die Portabilität des modellierten Workflows nicht einzuschränken, verfügt der Modeler über ein Transformationsfunktionalität, über die die BPMN Erweiterungen in ein BPMN konformes Modell überführt werden können. Dies wird in Abschnitt 5.1.2 näher betrachtet.

5.1.1. Aufbau und Integration der Erweiterungen

Jede Erweiterung wird im Modeler registriert um verwendet werden zu können. Die einzelnen Erweiterungen sind dabei voneinander unabhängig. Der Modeler bietet zusätzliche einige Funktionen, um die Integration neuer Erweiterungen zu erleichtern und Code-Duplikate in den Plugins zu reduzieren. Dies wird in Abschnitt 6.6 genauer erläutert.

Jedes Plugin kann den BPMN Standard um beliebige Elemente erweitern, unabhängig davon, ob es sich um Elemente mit oder ohne graphischer Repräsentation handelt. Namenskollisionen bei den Elementen werden vermieden, indem jede Modellerweiterung ein eindeutiges Präfix besitzt, dass sie identifiziert. Dieses Präfix wird vor jede Typenbezeichnung gesetzt, sodass Namen neuer Elementtypen nur im Kontext der Erweiterung eindeutig sein müssen und nicht im globalen Kontext aller Erweiterungen. Somit können verschiedene Erweiterungen ein Element einführen, das denselben Namen trägt, ohne dass es zu Problemen kommt. Erweiterungen, die sich auf, im BPMN Standard definierte, Elemente beziehen, können zu Problemen und Kollisionen mit anderen Erweiterungen führen. Falls beispielsweise Erweiterung A neue Regeln zur Verwendung von BPMN ServiceTasks definiert, ohne dafür einen eignen Typ einzuführen, kann es zu Problemen mit Erweiterung B kommen, da diese die im BPMN Standard spezifizierten Regeln voraussetzt. Erweiterungen an BPMN Elementen ohne die Einführung eines neuen Typs sollten daher möglichst vermieden werden. Stattdessen kann ein neuer Elementtyp eingeführt werden, auf den sich die Änderungen dann beziehen.

Für die Modellerweiterung können dann Editorfunktionen zum Erstellen der neuen Elemente hinzugefügt werden. So kann der Modeler um neue Modellierungselemente erweitert werden. Ein Plugin kann auch Funktionalität hinzufügen, die nicht direkt mit der Modellierung zu tun hat, wie beispielsweise eine Funktion zur Analyse des Workflows. Um solche Funktionen bei Bedarf in die Benutzeroberfläche des Modelers integrieren zu können, kann die Benutzeroberfläche teilweise über vordefinierte Schnittstellen erweitert werden. Erweiterungen könne beispielsweise Buttons in die Benutzeroberfläche des Modelers integrieren. Die Erweiterung der Benutzeroberfläche ist dabei auf einige vordefinierte Möglichkeiten beschränkt und kann nicht beliebig von den Plugins verändert werden. Nähere Details zu den Möglichkeiten der Erweiterung der Benutzeroberfläche werden in Abschnitt 6.3 erläutert. Ein Plugin kann auch auf externe Endpoints und Server zugreifen, um Daten und Funktionen von externen Quellen abzurufen. Die verwendeten Endpoints können bei Bedarf auch vom Modellierer zur Modellierungszeit konfiguriert werden.

Bei der Einbindung des Modelers in eine Web-Anwendung kann der Programmierer konfigurieren, welche Plugins er in dieser Instanz des Modelers verwenden möchte und welche nicht. So kann jede Instanz des Modelers auf den Kontext der Anwendung, in die der Modeler eingebunden wird, zugeschnitten werden, ohne dass Anpassungen im Code vorgenommen werden müssen. Der Programmierer kann ebenfalls initiale Konfigurationsparameter für Plugins angeben. Diese Parameter werden an das jeweilige Plugin weitergeben und ermöglichen so laufzeitspezifische Parameter festzulegen, wie die konkrete URL eines Endpoints, API Keys oder User Credentials. Der Aufbau dieser Konfigurationsparameter wird vom Plugin definiert. Beispielsweise kann für ein Plugin ein JSON Objekt mit Parametern übergeben werden.

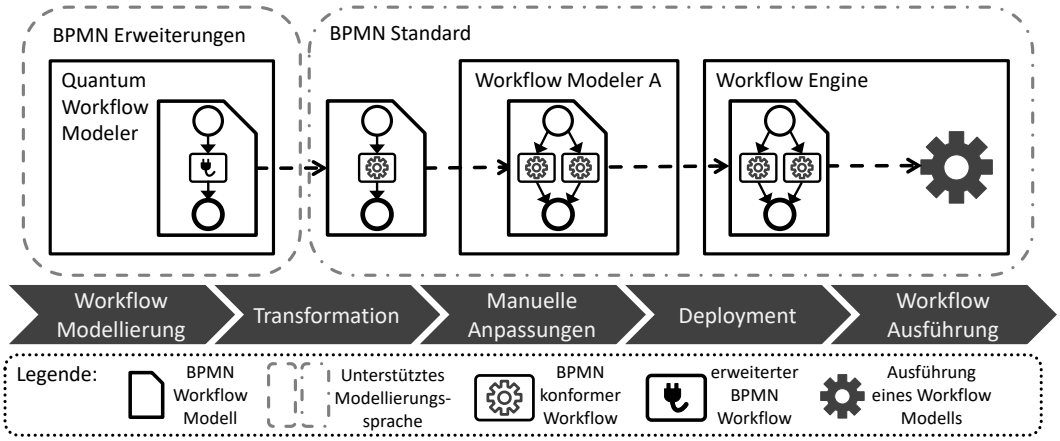

Abbildung 5.1.: Übersicht über die Schritte von der Modellierung über die Transformation bis zur Ausführung eines Workflows.

5.1.2. Kompatibilität mit dem BPMN-Standard

Um BPMN Workflows, die mit durch Plugins spezifizierten BPMN Erweiterungen modelliert wurden, in bereits vorhandenen Workflow Engines, die den BPMN Standard unterstützen, ausführen zu können, ist es erforderlich die erweiterten Workflows in BPMN konforme Workflows zu transformieren. Dadurch können die modellierten Workflows weiterhin in beliebigen BPMN Workflow Engines ausgeführt werden. Die Portabilität der Modelle wird nicht auf bestimmte Workflow Engines eingeschränkt. Ohne diese Transformation könnte ein im Modeler modelliertes Workflow-Modell nur im Quantum Workflow Modeler oder einem anderen Modeler, der dieselben BPMN Erweiterungen unterstützt, geöffnet werden. Die Ausführung wäre ebenfalls auf Workflow Engines beschränkt, die diese Erweiterungen unterstützen. Die Transformation in einen BPMN konformen Workflow dagegen ermöglicht die Verwendung des Workflows in BPMN konformen Editoren und Workflow Engines. Dieses Transformationskonzept wurde bereits bei anderen BPMN Erweiterungen verwendet, wie beispielsweise der QuantME Erweiterung [WBLW20] oder der BlockME Erweiterung [FHBL19], um die mit BPMN Erweiterungen modellierten Workflows in BPMN konformen Workflow Engines ausführen zu können. Dieses Konzept ist nicht auf BPMN beschränkt und wird auch für andere Workflow-Sprachen verwendet werden. Die SitME Erweiterung [BHK+15] verwendet das Transformationskonzept beispielsweise, um eingeführte BPEL Erweiterungen in native BPEL Konstrukte zu transformieren.

Die Transformation des Workflows in den BPMN Standard wird in einem separaten Transformationsschritt vor dem Deployment und der Ausführung des Modells durchgeführt. Wie in Abbildung 5.1 dargestellt, wird zunächst im Modellierungsschritt das BPMN-Diagramm mit BPMN Erweiterungen modelliert. Dieses Diagramm kann ausschließlich in Modelern geöffnet und bearbeitet werden, die alle BPMN Erweiterungen unterstützen, wie dem Quantum Workflow Modeler. Eine Ausführung dieses Modells wäre nur in Workflow Engines, die die Erweiterungen unterstützen, möglich. Um weiterhin existierende BPMN Workflow Engines verwenden zu können und die Portabilität des Workflow-Modells nicht einzuschränken, wird daher im Transformationsschritt das erweiterte BPMN-Diagramm in den BPMN-Standard transformiert. Dabei werden die Elemente der Erweiterung durch ein oder mehrere mit dem BPMN Standard konforme Elemente ersetzt. Die als Ersatz dienenden BPMN Elemente werden so in den Workflow eingebunden, dass die

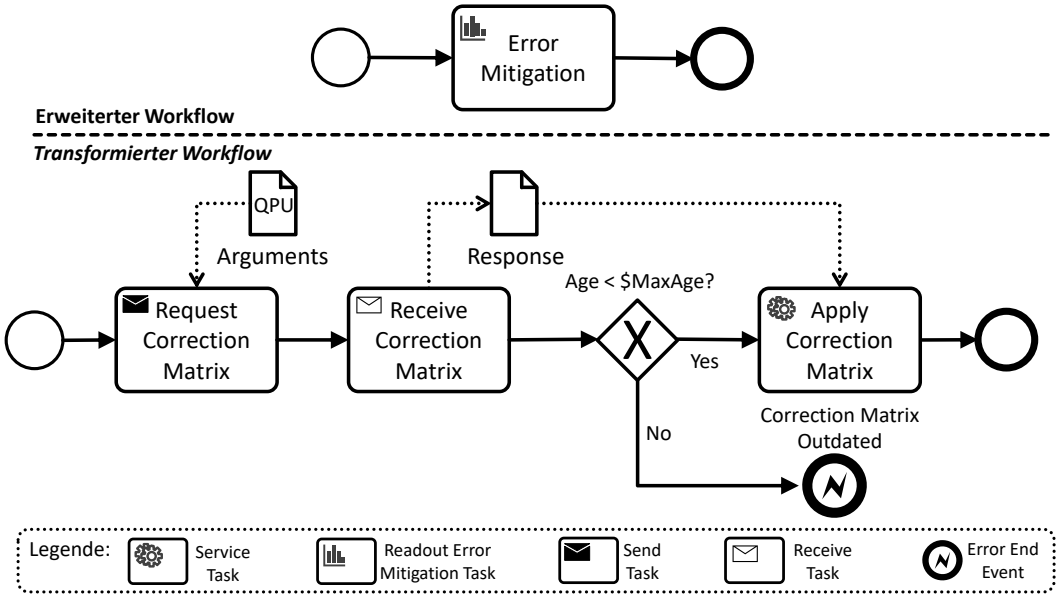

Abbildung 5.2.: Beispiel der Transformation eines erweiterten Workflows in einen BPMN Workflow, basierend auf [WBLW20].

Funktionalität des Workflows dieselbe bleibt, d.h. eine Ausführung des erweiterten Workflows liefert dasselbe Ergebnis wie eine Ausführung des transformierten Workflows. Die genauen Details der Transformation werden von jedem Plugin des Quantum Workflow Modelers selbst definiert. Im Transformationsschritt werden für alle im Modeler registrierten Plugins die entsprechenden Transformationen aufgerufen. Das Ergebnis des Transformationsschrittes ist ein BPMN konformes Workflow-Modell, das in einer entsprechenden Workflow Engine ausgeführt werden kann. Dieses Modell kann nach der Transformation auch in anderen BPMN konformen Workflow Modelern geöffnet und bearbeitet werden, um manuelle Anpassungen vornehmen zu können.

Zur Veranschaulichung des Transformationsschrittes ist in Abbildung 5.2 die Transformation eines Workflows abgebildet, der den `ReadoutErrorMitigationTask` der QuantME Modellierungserweiterung [WBLW20] verwendet. Dieser Task wird im Transformationsschritt durch den unten abgebildeten Workflow ersetzt, wobei die neuen BPMN Elemente dieselbe Funktionalität bieten. Sie modellieren die Schritte, die notwendig sind, um den Readout Error eines Quanten-Circuits durch die Anwendung einer Correction Matrix zu reduzieren. Das transformierte Diagramm enthält keine BPMN Erweiterungen mehr und kann in einer BPMN konformen Workflow Engine ausgeführt werden.

5.2. Dynamischer Import von Konfigurationen

Um die Wiederverwendbarkeit von konfigurierten BPMN Elementen zu ermöglichen, bietet der Modeler die Möglichkeit, sogenannte *Configurations* für ein im Workflow modelliertes BPMN Element zu erstellen. Eine Configuration ermöglicht die Definition einer Belegung der Attribute eines BPMN Elementes, sowie die Definition zusätzlicher Attribute. Dadurch können neue Elementtypen definiert werden, ohne Änderungen im Code vorzunehmen. Das Konzept der Configurations basiert

auf den Camunda Element Templates [CAMP823]. Element Templates erlauben die Erweiterung der Modellierung um domänenspezifische Elemente, ohne eine Erweiterung des, der Modellierung zugrundeliegenden, Metamodells. Mit Element Templates können neue Elemente mit eignen Attributen definiert und in der Modellierung von Workflows verwendet werden. Element Templates unterliegen einigen Einschränkungen, die Configurations nicht haben. Beispielsweise sind Configurations, im Gegensatz zu Element Templates, auf beliebige Elemente, inklusive benutzerdefinierter Erweiterungen, anwendbar, statt nur auf einige BPMN Elemente. Eine ausführlichere Beschreibung von Element Templates und ihren Einschränkungen folgt in Kapitel 6.

Configurations werden aus externen Repositories dynamisch geladen. Ein Repository ist in diesem Kontext ein externer Endpoint, über den Configurations geladen werden können, wie beispielsweise ein Server. Der Zugriff auf diese Repositories kann im Modeler konfiguriert werden. Der Modeler lädt dann die im Repository gespeichert Configurations zur Laufzeit. Aus dieser Liste kann der Modellierer eine geeignete Configuration auswählen. Neue Configurations müssen somit nicht im Modeler definiert werden, sondern werden dem Repository hinzugefügt.

Eine Configuration besteht aus einem Namen, einer ID, einer Beschreibung und einer Menge von ConfigurationAttributes. Ein ConfigurationAttribute definiert ein Attribut für das BPMN Element und dessen Belegung. Zusätzlich muss ein Binding definiert werden, das angibt, in welchem, in der Typdefinition des BPMN Elementes enthaltenen Attributes der Wert des ConfigurationAttributes gespeichert werden soll. Dies erlaubt es, in einer Configuration ein Attribut zu definieren, das nicht in der Typdefinition enthalten ist. Dadurch können zusätzliche Attribute definiert werden, ohne dass Anpassungen im Code vorgenommen werden müssen. Die zusätzlichen Attribute werden vom Modeler geladen und als separate Attribute dem Modellierer angezeigt. Die Werte dieser Attribute werden dann vom Modeler in den entsprechend spezifizierten Attributen gespeichert. Über diesen Mechanismus kann der Name eines definierten Attributes für eine konkrete Instanz eines BPMN Elementes überschrieben werden, um eine spezielle Bezeichnung zu wählen. Zum Beispiel wird eine Configuration *GitHubAccount* für ein benutzerdefiniertes BPMN DataObject *BenutzerAccount* definiert, das ein neues Attribut *KontaktInformation* in seiner Typdefinition spezifiziert. In der Configuration kann nun ein ConfigurationAttribut *MailAdresse* definiert werden, das als Binding das *KontaktInformation*-Attribut hat. Im Workflow-Modell kann der Modellierer nun für ein *BenutzerAccount*-DataObject die Configuration *GitHubAccount* auswählen und dann eine E-Mail Adresse angeben. Die Bedeutung des DataObjects im Modell kann dadurch detailliert spezifiziert werden, ohne dass neue Typen dem Metamodell hinzugefügt werden müssen.

Formal wird eine Configuration wie folgt definiert:

Eine **Configuration** definiert Attribute und deren Belegung für ein spezifisches BPMN Element. Eine Configuration besteht aus den folgenden Einträgen:

- **Name**: Nicht notwendigerweise eindeutiger Name der Configuration.
- **ID**: String, der die Configuration eindeutig beschreibt.
- **Beschreibung**: Textuelle Beschreibung der Configuration. (optional)
- **BpmnElement**: Name des Elementes, auf das die Configuration anwendbar ist. Es kann sich dabei um ein BPMN Element oder ein Elemente einer BPMN Erweiterung handeln.
- **ConfigurationAttributes**: Liste der Attribute, die durch diese Configuration konfiguriert werden.

Ein **ConfigurationAttribut** definiert den Namen, die Belegung und das Binding eines, durch eine Configuration spezifizierten Attributes. Es besteht aus den folgenden Einträgen:

- **Name**: Name des Attributes, der innerhalb einer Configuration eindeutig sein muss.
- **Wert**: Wert des Attributes als String. (optional)
- **Binding**: Name eines Attributes aus der Typdefinition des BPMN Elementes, in dem der Wert des Attributes gespeichert werden soll. Dabei kann es sich sowohl um Attribute des BPMN Standards als auch um Attribute einer Erweiterung handeln.

In Listing 5.1 ist ein Beispiel für eine Configuration, definiert durch einen JSON String, zu sehen. Die *GitHubLogin*-Configuration spezifiziert Attribute für einen benutzerdefinierten Task, den *LoginTask*. Ein *LoginTask* hat zwei Attribute, *Url*, einen String und *Options*, eine Key-Value-Map. Die Configuration definiert das *Url*-Attribut des LoginTask und spezifiziert ein neues Attribut, *RedirectUrl*, welches keinen vordefinierten Wert besitzt und in der Key-Value-Map *Options* gespeichert wird. Dieses Attribut ist nicht im LoginTask definiert und ist spezifisch für einen GitHub Login. Durch diese Configuration kann der Modellierer spezielle Details für einen Login in einen GitHub Account definieren, ohne einen neuen Typ im Metamodell definieren zu müssen. Nach Anwendung der Configuration bleibt das modellierte Element weiterhin ein *LoginTask*. Andere Login-Typen, wie beispielsweise ein Login in einen GitLab Account, können über weitere Configurations spezifiziert werden, ohne dass ein neuer Task-Typ im Code der Erweiterung definiert werden müsste.

Listing 5.1 Beispiel einer Configuration, definiert als JSON String.

```
{
    "Name": "GitHub Login",
    "ID": "GitHubLogin",
    "Beschreibung": "Login in einen GitHub Account",
    "BpmnElement": "LoginTask",
    "ConfigurationAttributes": [
        {
            "Name": "RedirectUrl",
            "Wert": "",
            "Binding": "Options"
        },
        {
            "Name": "Url",
            "Wert": "github.com/login",
            "Binding": "Url"
        }
    ]
}

```

Die Verwendung von Configurations als Ersatz für die Definition neuer Typen ist nur eingeschränkt zu empfehlen. Sobald ein Typ innerhalb einer Erweiterung referenziert werden muss, reicht eine Configuration nicht mehr aus, da eine Configuration keinen neuen Typ erzeugt. Somit kann eine Configuration nicht referenziert werden, um beispielsweise Modellierungsregeln zu erweitern. Der Entwickler einer BPMN Erweiterung sollte sorgfältig abwägen, wann eine Configuration

ausreichend ist und wann einen neue Typdefinition notwendig wird. Allgemein sind Configurations dafür gedacht, bestehende Typdefinitionen für konkrete Anwendungsfälle zu erweitern, um konkrete, detaillierte Varianten dieser Typdefinition zu modellieren.

5.3. Modellierungserweiterung des Datenflusses

Wie bereits in Abschnitt 2.2 erläutert, ist die Modellierung konkreter, detaillierter Daten im BPMN Standard nicht näher beschrieben. Insbesondere ist auch der Umgang mit dem Datenfluss während der Ausführung des Prozesses im BPMN Standard offen gehalten. Um den Datenfluss explizit mit konkreten Daten zu modellieren, sodass diese bei der Ausführung in einer Workflow Engine verfügbar sind, wird in diesem Abschnitt eine konkrete Realisierung und Erweiterung des BPMN Standards eingeführt. Diese wird im Folgenden als die Datenflusserweiterung bezeichnet. Mit ihr können konkrete Daten modelliert werden, die während der Ausführung des Workflows verfügbar sind. Diese Erweiterung ermöglicht es dem Modellierer, neben der eben beschriebenen Modellierung des Datenflusses, die Transformation von Datenobjekten zu modellieren. Um Workflows, die diese Erweiterung verwenden, weiterhin in BPMN Workflow Engines ausführen zu können, wird ein Transformationsschritt vor dem Deployment auf die Workflow Engine eingeführt. Dies ermöglicht dem Modellierer, den Datenfluss eines Workflows mit der hier beschriebenen Erweiterung zu modellieren, ohne hinsichtlich der Ausführung an eine spezielle BPMN Workflow Engine gebunden zu sein.

Die Modellierungserweiterungen und der Transformationsschritt werden im Folgenden näher beschrieben und durch ein XSD-Schema definiert, um die Struktur der Modellierungselemente der Erweiterung formal zu definieren. Das vollständige Schema ist in Anhang A zu finden. Es wird die in Abbildung 5.3 abgebildete graphische Notation für die Modellierungselemente verwendet.

5.3.1. Erweiterung der Datenobjekte

Das Ziel der in diesem Abschnitt beschriebenen Datenobjekte ist es, dem Modellierer die Möglichkeit zu geben, den Datenfluss graphisch zu modellieren, sodass die Daten während der Ausführung des Workflows verfügbar sind. Dabei sollen sowohl konkrete Daten detailliert modelliert, als auch Datentransformationen spezifiziert werden können. Um dies zu erreichen, werden in den folgenden Abschnitten neue Datenklassen eingeführt, die bestehende BPMN Klassen erweitern.

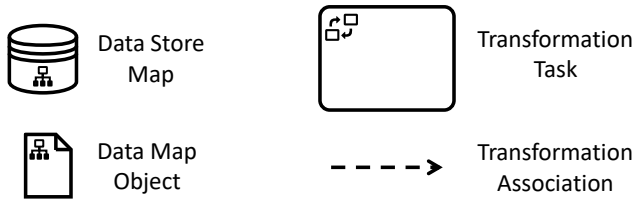

Abbildung 5.3.: Graphische Notation der in Abschnitt 5.3 beschriebenen Modellierungselemente.

Listing 5.2 XSD-Schema der `DataMapObject`-Klasse.

```
1  <xsd:element name="dataMapObject" type="tDataMapObject"
2                      substitutionGroup="bpmn:flowElement"/>
3  <xsd:complexType name="tDataMapObject">
4      <xsd:annotation>
5          <xsd:documentation>
6              Data object type to define a set of data.
7          </xsd:documentation>
8      </xsd:annotation>
9      <xsd:complexContent>
10         <xsd:extension base="bpmn:tDataObject">
11             <xsd:attribute name="content" type="tKeyValueMap" use="required"/>
12         </xsd:extension>
13     </xsd:complexContent>
14 </xsd:complexType>
```

Einführung der DataMapObject-Klasse

In Listing 5.2 ist die `DataMapObject`-Klasse als XSD-Schema beschrieben. Sie erweitert die DataObject-Klasse um das `Content`-Attribut, Zeile 11 in Listing 5.2, in dem der Inhalt des `DataMapObjects` sowie eventuelle Metadaten gespeichert werden können. Um nah an der in Abschnitt 6.4 beschriebenen Implementierung des durch das XSD Schema spezifizierten Metamodells zu bleiben, wurde hier ein Attribut und kein Content-Typ zur Definition des `Content`-Attributes verwendet. Der Typ des Attributs ist eine Key-Value-Map, sodass die Eigenschaften des `DataMapObjects`, d.h. Inhalt, Metadaten wie Dateityp, etc., als Key-Value-Paare flexibel vom Modellierer ergänzt werden können, ohne dass das Datenschema erweitert werden muss, um für jede neue Eigenschaft ein entsprechendes Attribut im Datenschema zu definieren. Dies ermöglicht es dem Modellierer, beliebige Daten in einem `DataMapObject` zu speichern und die Verwendung dieser Daten dennoch graphisch über DataAssociations modellieren zu können. Die Verwendung und die Verfügbarkeit des `DataMapObjects` ist analog zu der eines DataObjects.

Die `DataMapObject`-Klasse kann erweitert werden, um spezifische Datenobjekte von domänenspezifischen BPMN Erweiterungen modellieren zu können und eine Art Typisierung im Datenfluss zu verwenden. Die neuen Datenklassen können domänenspezifische Attribute definieren und dennoch mit anderen Elementen und Konzepten der Datenerweiterung verwendet werden. So können weiterhin flexibel zusätzliche Attribute im `Content`-Attribut spezifiziert werden. Beispielsweise kann ein *UserDataObject* definiert werden, das von der `DataMapObject`-Klasse erbt und über die Attribute *Name* und *Alter* verfügt. Wird ein *UserDataObject* als Input eines Tasks modelliert, kann der Programmierer dieses Tasks davon ausgehen, dass er als Input die Attribute *Name* und *Alter* hat. Dadurch können Daten typisiert werden und der Datenfluss übersichtlicher gestaltet werden. Zusätzlich wird es für den Programmierer leichter nachzuvollziehen, welche Daten eine Activity als Input erhält.

Der Name eines `DataMapObjects` identifiziert im Prozesskontext genau ein Datenobjekt. Es können mehrere `DataMapObjects` mit demselben Namen modelliert werden, allerdings repräsentieren sie alle dieselben Daten bzw. dasselbe Datenobjekt. Änderungen an einem `DataMapObject` überschreiben dadurch den Stand der Daten für alle `DataMapObjects` mit demselben Namen. Dies kann bei der Modellierung verwendet werden, um dieselben Daten mit verschiedenen Tasks zu assoziieren,

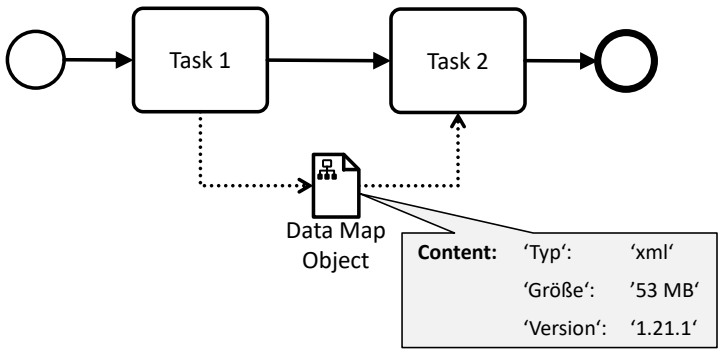

Abbildung 5.4.: Beispiel eines Workflows mit zwei Tasks und einem `DataMapObject` zur Modellierung des Datenflusses.

ohne dass es notwendig ist, für jede Assoziation eine DataAssociation von bzw. zu demselben `DataMapObject` zu ziehen. Falls beispielsweise am Anfang eines langen und komplexen Workflows eine JSON Datei benötigt wird, die dann erst am Ende wieder verwendet wird, kann der Modellierer statt nur eines `DataMapObjects` zwei mit demselben Namen zur Modellierung der JSON Datei verwenden: Das erste `DataMapObject` wird am Anfang des Workflows modelliert, das zweite am Ende. Dadurch, dass beide `DataMapObjects` denselben Namen haben, verweisen sie auf dasselbe Datenobjekt und Tasks, die sie verwenden greifen auf dieselben Daten zu.

Werden `DataMapObjects` mit Activities in Schleifen oder parallelen Kontrollflüssen verbunden, verhält sich das DataObject wie eine Variable. Das bedeutet, dass der Wert des `DataMapObjects` überschrieben wird und somit die Activity, die als letztes ausgeführt wird, den Wert des `DataMapObjects` bestimmt. Ist dieses Verhalten nicht erwünscht, sollte eine `DataStoreMap` verwendet werden, um die Ergebnisse des Kontrollflusses zu speichern. Der Modellierer muss somit bei der Modellierung des Datenflusses auf Race Conditions achten, insbesondere wenn der Datenfluss für parallele Ausführungen modelliert wird.

Abbildung 5.4 zeigt ein `DataMapObject` in einem Workflow. Die Tabelle zeigt die Struktur des `Content`-Attributes des `DataMapObjects`. Es besteht in diesem Beispiel aus drei Key-Value-Paaren, die Metadaten einer durch das `DataMapObject` repräsentierten Datei beschreiben. Das `DataMapObject` wird dabei von *Task1* als Output erzeugt und von *Task2* als Input aufgegriffen.

Einführung der DataStoreMap-Klasse

Analog zu der `DataMapObject`-Klasse wird eine `DataStoreMap`-Klasse eingeführt, deren XSD-Schema in Listing 5.3 abgebildet ist. Sie erweitert die DataStore-Klasse um das *Details*-Attribut. Dieses Attribut, Zeile 10 in Listing 5.3, enthält die Eigenschaften der `DataStoreMap`, wie beispielsweise die Verbindungsdaten des DataStores, als Key-Value-Paare. Eine `DataStoreMap` kann auf dieselbe Art und Weise wie ein DataStore in BPMN verwendet werden, um einen persistenten Datenspeicher zu modellieren. `DataMapObjects` werden dagegen zur Modellierung von temporären Daten verwendet, die nur solange verfügbar sind, wie es eine aktive Activity gibt, die sie referenziert.

Listing 5.3 XSD-Schema der `DataStoreMap`-Klasse.

```
<xsd:element name="dataStoreMap" type="tDataStoreMap" substitutionGroup="bpmn:flowElement"/>
    <xsd:complexType name="tDataStoreMap">
        <xsd:annotation>
            <xsd:documentation>
                Data store type to define a persistent data storage.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="bpmn:tDataStore">
                <xsd:attribute name="details" type="tKeyValueMap" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

In Abbildung 5.5 ist eine `DataStoreMap` in einem Workflow dargestellt. Die `DataStoreMap` modelliert in diesem Beispiel ein GitHub Repository. Die Tabelle zeigt die Struktur des *Details*-Attributes für diese `DataStoreMap`, das in diesem Beispiel aus Key-Value-Paaren besteht, die eine Verbindung mit dem Repository ermöglichen. Durch den flexibel erweiterbaren Aufbau des *Details*-Attributes können weitere solcher Key-Value-Paare vom Modellierer erzeugt werden, um andere Eigenschaften der `DataStoreMap` zu definieren, wie beispielsweise einen Authentifizierungs-Token, um nicht die Login Credentials angeben zu müssen.

5.3.2. Modellierung von Datentransformation

Zusätzlich zur Erweiterung der Datenobjekte soll der BPMN Standard um die Möglichkeit zur Modellierung von ausführbaren Transformation von Datenobjekten erweitert werden. Dafür werden in diesem Abschnitt neue Modellierungselemente eingeführt, mit denen die Transformation von `DataMapObjects` modelliert werden kann.

Im Kontext dieser Arbeit wird zwischen atomaren und komplexen Datentransformation unterschieden. Eine atomare Transformation bezeichnet in diesem Kontext Transformationen, die durch atomare Operationen beschrieben werden, die genau ein Ergebnis erzeugen und keine internen Variablen

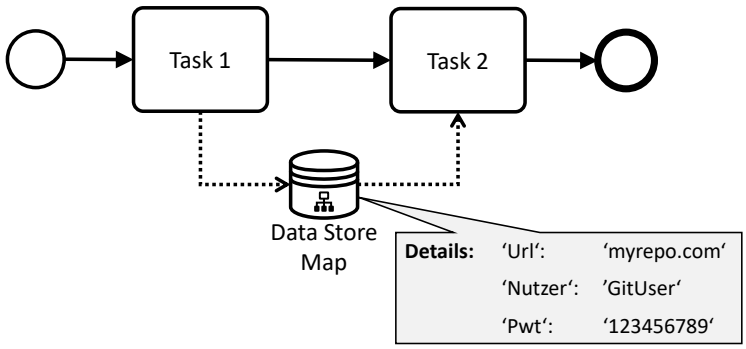

Abbildung 5.5.: Beispiel eines Workflows mit zwei Tasks und einer `DataStoreMap` zur Modellierung des Datenflusses.

besitzen. Das Ergebnis einer atomaren Transformation ist ein primitiver Datentyp, wie eine Zahl oder ein String. Dies umfasst beispielsweise Transformationen, die die Anzahl der Attribute eines DataObjects reduzieren oder Transformationen, die einzelne Attribute verändern, beispielsweise Strings konkatenieren. Komplexe Transformationen dagegen bezeichnen Transformationen, die durch Algorithmen beliebiger Komplexität definiert werden. Diese Algorithmen können interne Variablen besitzen und mehrere komplexe Objekte als Ergebnis erzeugen. Komplexe Transformationen können auf andere Datenobjekte zugreifen oder durch zusätzliche Parameter und Konstanten konfiguriert werden. Ein Beispiel einer solchen Transformation ist die Umwandlung eines `DataMapObjects`, das eine JSON Datei repräsentiert, in ein DataObject, das eine XML Datei repräsentiert.

Modellierung Atomarer Transformationen

Listing 5.4 XSD-Schema der `TransformationAssociation`-Klasse.

```
<xsd:element name="transformationAssociation" type="tTransformationAssociation"
                                                substitutionGroup="bpmn:baseElement"/>
    <xsd:complexType name="tTransformationAssociation">
        <xsd:annotation>
            <xsd:documentation>
                DataAssociation type to define a transformation of DataMapObjects.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="bpmn:tDataAssociation">
                <xsd:attribute name="expressions" type="tKeyValueMap" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

Um atomare Transformationen graphisch modellieren zu können, wird die sogenannte `TransformationAssociation` eingeführt, beschrieben durch das XSD-Schema in Listing 5.4. Eine `TransformationAssociation` verbindet ein `DataMapObject` mit einer Activity oder einem weiteren `DataMapObject`. Die Verbindung ist gerichtet, wobei das Startobjekt transformiert wird. Das Ergebnis der Transformation ist eine Menge von Variablen, die an das Zielobjekt übertragen wird. Handelt es sich beim Zielobjekt um eine Activity, wird das Ergebnis der Transformation der Menge an Input-Variablen der Activity hinzugefügt. Handelt es sich um ein `DataMapObject`, werden die Variablen als Key-Value-Paare dem `Content`-Attribut hinzugefügt, wobei Key-Value-Paare mit demselben Key überschrieben werden.

Graphisch wird eine `TransformationAssociation` durch einen gestrichelten Pfeil, der vom Startobjekt zum Zielobjekt zeigt, dargestellt. Die graphische Notation ist in Abbildung 5.6 dargestellt.

Abbildung 5.6.: Graphische Notation der `TransformationAssociation`.

Die Transformation wird über das Expressions-Attribut, siehe Zeile 11 in Listing 5.4, spezifiziert, das es ermöglicht, atomare Transformationen über eine formale Expression Language, wie beispielsweise die unified Expression Language [UEL10], zu definieren. Das Expressions-Attribut besteht dabei aus einer Liste von Key-Value-Einträgen, wobei jeder Eintrag eine atomare Transformation spezifiziert, deren Ergebnis ein neues Attribut im Zielobjekt ist. Der Key beschreibt den Namen der Variable, die durch die in der TransformationAssociation modellierten Transformationen erzeugt wird. Der Value beschreibt den Wert der Variable, der hart codiert sein kann oder durch eine formale Expression angegeben werden kann. Dadurch können simple Transformationen, wie beispielsweise String-Konkatenationen, definiert werden oder der Wert eines, bereits im Kontext existierenden, DataMapObjects kann referenziert werden. Somit können andere, im Kontext verfügbare DataMapObjects ebenfalls referenziert werden.

Die im Expression-Attribut definierten Attribute werden dem Zielobjekt hinzugefügt. Es werden keine Attribute des Startobjektes automatisch ins Zielobjekt übernommen. Es können im Expressions-Attribut auch neue Variablen erzeugt werden, die nicht im Startobjekt vorhanden sind. Über die Expression Language kann diesen neuen Variablen ein Wert zugewiesen werden, der aus einem oder mehreren Attributen des Startobjektes besteht. Somit können TransformationAssociations verwendet werden, um DataMapObjects bzw. deren Attribute zu filtern und zu transformieren. Dem Modellierer ist es damit möglich, genau zu bestimmen, welche Variablen in der Ergebnismenge enthalten sein sollen und welchen Wert sie haben.

Eine TransformationAssociation zwischen einem DataMapObject und einer Activity kann eingesetzt werden, um die Anzahl der Input-Variablen der Activity, die durch das DataMapObject modelliert werden, für einen spezifischen Task anzupassen. Dem Modellierer wird so ermöglicht, graphisch zu steuern, welche Eigenschaften eines DataMapObjects einer Activity als Input-Variablen zur Verfügung stehen sollen und mit welchem Inhalt. Die modellierte Transformation ist graphisch im Modell erkennbar und verdeutlicht, welches DataObject transformiert wurde. Bei einer textuellen Spezifikation der Transformation direkt in den Input-Variablen der Activity wäre dies nicht der Fall. Durch eine TransformationAssociation kann vermieden werden, dass eine Activity alle Eigenschaften eines DataMapObjects erhält, obwohl beispielsweise nur zwei bestimmte Attribute benötigt werden. Auf diese Weise können ebenfalls Konstanten oder Konfigurationsparameter an einen speziellen Task übermittelt werden. Dabei kann der Input einer Activity genau auf ihren Kontext zugeschnitten werden, ohne dabei den Inhalt des DataMapObjects zu verändern. Der Vorteil der Modellierung durch eine TransformationAssociation gegenüber der Modellierung durch die Spezifikation direkt in den Input-Variablen der Activity ist, dass bei der Modellierung über TransformationAssociations selbst bei mehreren eingehenden Associations nachvollziehbar bleibt, auf welches DataObject sich die Transformationen beziehen. Dies unterstützt den Modellierer bei der Modellierung von Activities, die mehrere transformierte DataMapObjects als Input-Variablen besitzen. Zusätzlich ist durch eine TransformationAssociation direkt im Modell erkennbar, dass diese Activity eine veränderte Version des DataMapObjects als Input erhält. Bei einer Modellierung dieses Verhaltens direkt in den Input-Variablen der Activity ist dieser Umstand nicht graphisch sichtbar.

In Abbildung 5.7 ist ein Beispiel für den Einsatz von TransformationAssociations zur Modellierung eines spezifischen Inputs für *Task2* abgebildet. Das DataMapObject *UserData* enthält Benutzerdaten, gespeichert als Key-Value-Paare im Content-Attribut. *Task2* benötigt nicht alle diese Informationen als Input, lediglich den *Name* und *Adresse*. Über das Expressions-Attribut der TransformationAssociation wird daher der Input von *Task2* auf eben diese Informationen

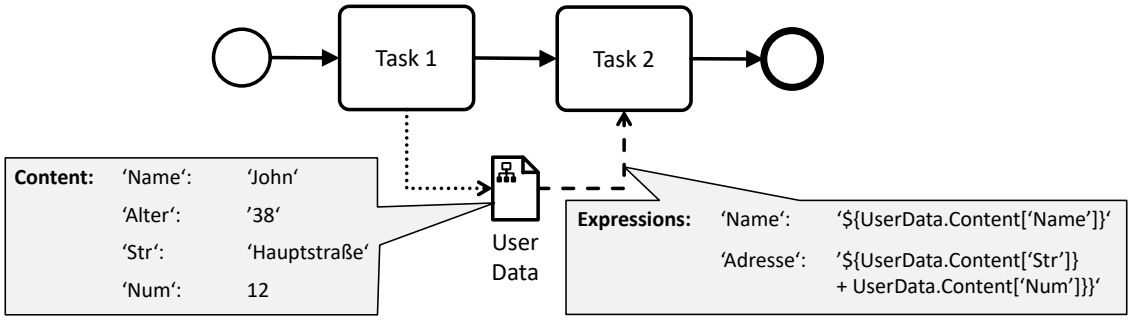

Abbildung 5.7.: Beispiel eines Workflows, indem eine `TransformationAssociation` eingesetzt wird, um die Input-Variablen von Task2 auf Name und Adresse zu beschränken.

reduziert, wobei die Adresse direkt aus den einzelnen Informationen des `DataMapObjects` zusammengesetzt wird. *Task2* erhält somit als Input nicht alle Benutzerdaten, sondern lediglich die für seinen Ausführungskontext relevanten, ohne dass der Inhalt des `DataMapObjects` *UserData* verändert wird.

Eine `TransformationAssociation` zwischen zwei `DataMapObjects` bietet die Möglichkeit, ein `DataMapObject` für andere Activities sichtbar zu transformieren. Dies hat den Vorteil, dass das Ergebnis der Transformation als separates `DataMapObject` definiert und von nachfolgenden Activities wiederverwendet werden kann. Das Ziel-`DataMapObject` wird dabei, falls nicht bereits vorhanden, neu erzeugt und besteht aus der Ergebnismenge der Transformation. Die so entstandenen Variablen werden als Key-Value-Paare dem `Content`-Attribut hinzugefügt. Falls das Ziel-`DataMapObject` bereits existiert, da es beispielsweise der Output eines Tasks ist, so wird die Ergebnismenge an Variablen der Transformation mit den bereits im `Content`-Attribut vorhandenen Variablen gemischt, wobei existierende Variablen überschrieben und neue hinzugefügt werden.

Abbildung 5.8 ist ein Beispiel für den Einsatz von `TransformationAssociations` zur Erzeugung eines neuen `DataMapObjects` durch Transformation aus einem bestehenden `DataMapObject`. Das `DataMapObject` *UserData* enthält Benutzerdaten, wie Name, Straße, Hausnummer und Alter, gespeichert als Key-Value-Paare im `Content`-Attribut. *Task2* und *Task3* benötigen nur die Adressinforma-

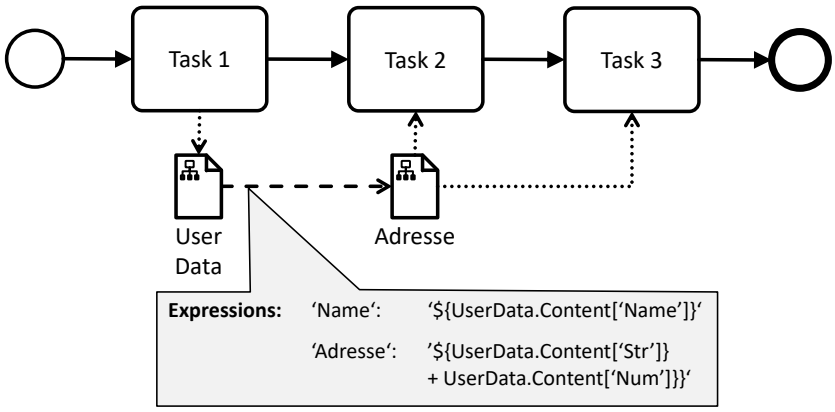

Abbildung 5.8.: Beispiel eines Workflows, in dem eine `TransformationAssociation` eingesetzt wird, um das `DataMapObject` *UserData* in ein `DataMapObject` *Adresse* zu transformieren.

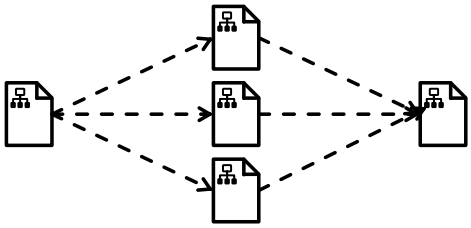

Abbildung 5.9.: Beispiel für die Verwendung von `TransformationAssociations` zur Vereinfachung des Datenflusses.

tionen als Input. Daher wird ein neues `DataMapObject` *Adresse* erzeugt, indem das *UserData*-Objekt transformiert wird. Dabei werden im `Expressions`-Attribut der `TransformationAssociation` ausschließlich die Adressinformationen aufgeführt, sodass lediglich diese Informationen in *Adresse* enthalten sind. Das so neu erzeugte `DataMapObject` dient nun *Task2* und *Task3* als Input. Somit muss nicht eine entsprechende `TransformationAssociation` zwischen *UserData* und *Task2* bzw. *Task3* gezogen werden, wie im Beispiel in Abbildung 5.7, was die Modellierung einfacher und übersichtlicher gestaltet.

Ein `DataMapObject` kann auch mehrere ausgehende bzw. eingehende `TransformationAssociations` besitzen. Dadurch können aus einem `DataMapObject` mehrere neue `DataMapObjects` erzeugt bzw. mehrere `DataMapObjects` zu einem zusammengefasst werden. Somit kann der Datenfluss vereinfacht und übersichtlicher gestaltet werden, da beispielsweise `DataMapObjects`, die von verschiedenen Tasks erzeugt, jedoch zusammen als Input verwendet werden, zu einem `DataMapObject` zusammengefasst werden können. Alle weiteren Tasks erhalten lediglich das zusammengefasste `DataMapObject` als Input, statt aller drei `DataMapObjects`. Dies wird in Abbildung 5.9 beispielhaft dargestellt.

Modellierung Komplexer Transformationen

Eine komplexe Transformation erfordert ein separates Element, um dem Modellierer die Möglichkeit zu geben, die Transformationsalgorithmen, sowie zusätzliche Konfigurationsparameter, wie andere DataObjects, DataStores oder benutzerdefinierte Attribute, zu spezifizieren. Dafür kann ein entsprechendes Element in den Kontrollfluss eingebaut werden oder separat im Datenfluss spezifiziert werden, wie das DT Element von Hahn et al. [HBKL18]. Der Vorteil des DT Elementes, abgebildet in Abbildung 5.10, besteht darin, dass insbesondere in Situationen, in denen mehrere Teilnehmer modelliert werden müssen, die Nachrichten in unterschiedlichen Formaten austauschen, die Transformation der Daten durch das DT Element übersichtlich und explizit modelliert werden kann. Allerdings widerspricht eine wie von Hahn et al. eingeführte separate Middleware zur Ausführung und Interpretation des Datenflusses der Anforderung der Konformität mit dem BPMN Standard. Daher müsste eine solche, direkt im Datenfluss spezifizierte Transformation, um BPMN konform zu sein, im einem Transformationsschritt in den Kontrollfluss integriert werden, um von einer BPMN konformen Workflow Engine interpretiert werden zu können. Die Position innerhalb des Kontrollflusses wäre abhängig von der Verfügbarkeit der `DataMapObjects` des Inputs des `TransformationTasks` und kann nicht explizit vom Modellierer definiert werden. Dies erschwert die Nachvollziehbarkeit des transformierten Workflows. Eine direkt im Kontrollfluss definierte Transformation dagegen, bietet dem Modellierer in diesem Kontext einen nachvollziehbaren Kontrollfluss, da sich dieser durch die Transformation nicht verändert und vollständig vom Modellierer definiert werden kann.

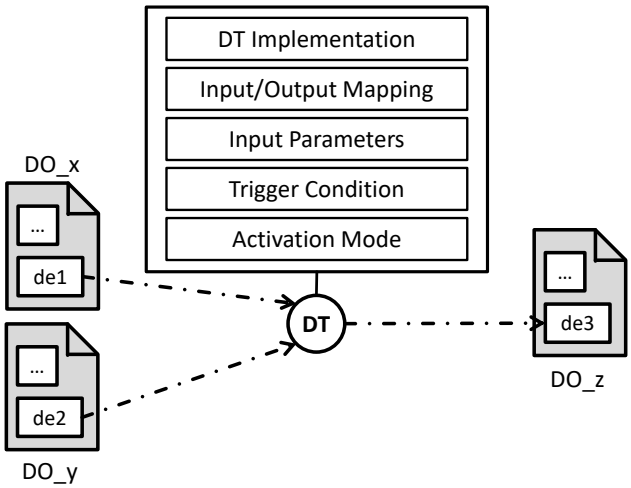

Abbildung 5.10.: Beispiel für die Verwendung eines Data Transformation (DT) Elements zur Modellierung von Transformationen zwischen DataObjects nach Hahn et al. [HBKL18].

Die Transformation würde lediglich durch ein Camunda-BPMN konformes Element ersetzt, bliebe aber an derselben Position im Kontrollfluss. Insbesondere bei Transformationen innerhalb einer Schleife oder einer parallelen Ausführung gestaltet sich eine Modellierung direkt im Kontrollfluss in diesem Kontext übersichtlicher. Daher wird im folgenden eine Modellierungserweiterung zur Modellierung von Datentransformationen direkt im Kontrollfluss eingeführt.

Listing 5.5 XSD-Schema der `TransformationTask`-Klasse.

```
1  <xsd:element name="transformationTask" type="tTransformationTask"
2                                          substitutionGroup="bpmn:flowElement"/>
3      <xsd:complexType name="tTransformationTask">
4          <xsd:annotation>
5              <xsd:documentation>
6                  Task type to define a complex transformation of data.
7              </xsd:documentation>
8          </xsd:annotation>
9          <xsd:complexContent>
10             <xsd:extension base="bpmn:tTask">
11                 <xsd:attribute name="parameters" type="tKeyValueMap" use="required"/>
12             </xsd:extension>
13         </xsd:complexContent>
14     </xsd:complexType>
```

Komplexe Transformationen werden durch den `TransformationTask` modelliert. Ein `TransformationTask` wird durch das XSD-Schema in Listing 5.5 definiert. Der `TransformationTask` erweitert einen Task und ermöglicht es, Transformationen explizit im Kontrollfluss zu spezifizieren. Der Modellierer kann im Task alle benötigten Attribute und Variablen spezifizieren. Dafür verfügt der `TransformationTask` über ein `Parameters`-Attribut, Zeile 11 in Listing 5.5, dem als Key-Value-Paare beliebige Konfigurationsparameter und Konstanten hinzugefügt werden können. Der Key ist dabei der Name des Parameters bzw. der Konstanten und der Value der Wert. Der Wert

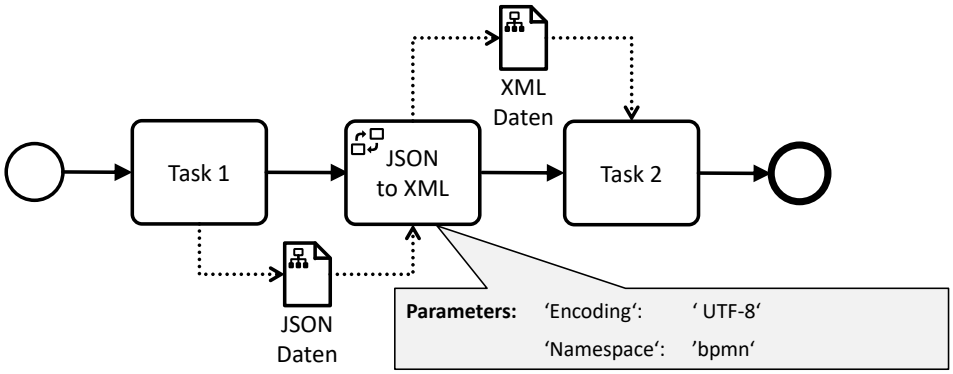

Abbildung 5.11.: Beispiel eines `TransformationTasks` um Daten im JSON-Format in ein XML-Format zu transformieren.

kann entweder hart kodiert sein oder über eine formale Expression auf andere `DataMapObjects` zugreifen. Zusätzlich können notwendige Input-Parameter über die Datenflusserweiterungen auch graphisch durch `DataMapObjects` und `DataStoreMaps` modelliert werden. In der Implementierung des `TransformationTasks` können beliebig komplexe Algorithmen angewandt werden, die die modellierten Input-Parameter verwenden können.

Um Transformationsalgorithmen wiederverwenden zu können, kann der Modellierer Configurations und ein entsprechendes Repository konfigurieren, wie in Abschnitt 5.2 beschrieben. In diesem Repository können verschiedene Transformationsalgorithmen und deren Implementierung als Configurations gespeichert werden. Aus dem Repository kann im Modeler dann ein Transformationsalgorithmus bzw. die Configuration, die ihn repräsentiert, ausgewählt werden. Der `TransformationTask` bzw. seine Attribute werden dann gemäß dieses Transformationsalgorithmus gesetzt, was neben dem Namen des Algorithmus ebenfalls Konfigurationsparameter und Konstanten umfassen kann. Neue Transformationsalgorithmen können über dieses Repository hinzugefügt und direkt im Modeler verwendet werden, ohne dass der Programmcode erweitert werden muss.

Abbildung 5.11 zeigt einen Workflow, in dem durch einen `TransformationTask` Daten im JSON-Format in das XML-Format transformiert werden. Die JSON-Daten werden als Input über ein `DataMapObject` modelliert. Der `TransformationTask` wird über das `Parameters`-Attribut konfiguriert. In diesem Beispiel kann so das Encoding und der Namespace der erzeugten XML-Daten konfiguriert werden, die als Output durch ein `DataMapObject` repräsentiert werden.

5.3.3. Transformation in den BPMN-Standard

Um BPMN Diagramme, die die in den vorherigen Abschnitten beschriebenen Erweiterungen des Datenflusses enthalten, in bereits vorhandenen, BPMN konformen Workflow Engines ausführen zu können, wird in diesem Abschnitt die Transformation der Datenflusserweiterung in einen BPMN konformen Workflow beschrieben, wie in Abschnitt 5.1.2 erläutert.

Im Zuge der Transformation werden die Elemente der Datenflusserweiterung durch die BPMN Elemente, die sie erweitern, ersetzt. Wie in Abbildung 5.12 dargestellt, werden dabei die speziellen Details der jeweiligen Realisierung des BPMN Standards der Workflow Engine genutzt, um die Funktionalität der Datenflusserweiterung beizubehalten und dennoch die Ausführbarkeit in

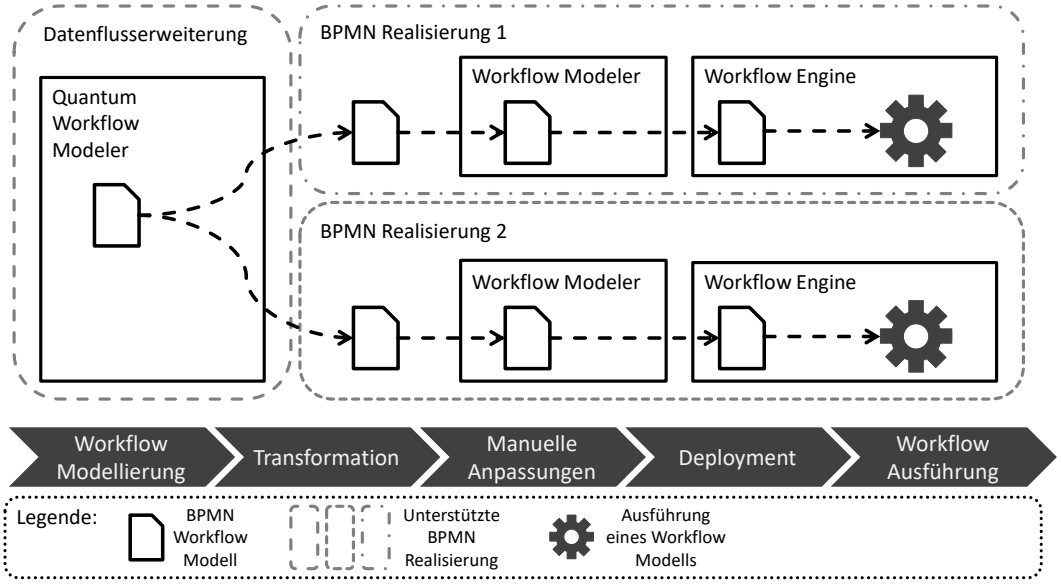

Abbildung 5.12.: Darstellung der Transformation eines Workflows, modelliert mit der Datenflusserweiterung, in Workflows mit spezifischen Realisierungen.

der jeweiligen BPMN Workflow Engine zu gewährleisten. Insbesondere der Umgang mit dem Datenfluss wird von Workflow Engines unterschiedlich gehandhabt. Daher sind die Details des Transformationsschrittes abhängig von der Workflow Engine, in der das Workflow-Modell ausgeführt werden soll. Das Ergebnis des Transformationsschrittes ist ein BPMN konformes Workflow-Modell, das in einer entsprechenden Workflow Engine ausgeführt werden kann.

5.3.4. Validierung der Datenflusserweiterung

Um die Datenflusserweiterung und die durch sie eingeführten Modellierungselemente zu validieren, sind in Abbildung 5.13 zwei Workflows abgebildet. Der obere Workflow wurde mithilfe der Datenflusserweiterung modelliert und der untere Workflow wurde ohne Erweiterung mit BPMN Elementen modelliert. Beide Workflows beschreiben denselben Prozess zur Erstellung einer Versandrechnung für einen Kunden. Wie das Beispiel zeigt, lässt sich der Datenfluss durch die Modellierungserweiterung wesentlich übersichtlicher darstellen.

Durch die Verwendung einer `TransformationAssociation` wird aus dem `DataMapObject` *Kundendaten* das `DataMapObject` *Adresse* erzeugt, welches die vollständige Adresse des Kunden aus den *Kundendaten* ableitet. Mit der Datenflusserweiterung kann dies ohne einen zusätzlichen BPMN Task modelliert werden. Um dieselbe Funktionalität ohne die Erweiterung zu modellieren, muss das DataObject *Adresse* über den zusätzlichen Task *Adresse erstellen* erzeugt werden.

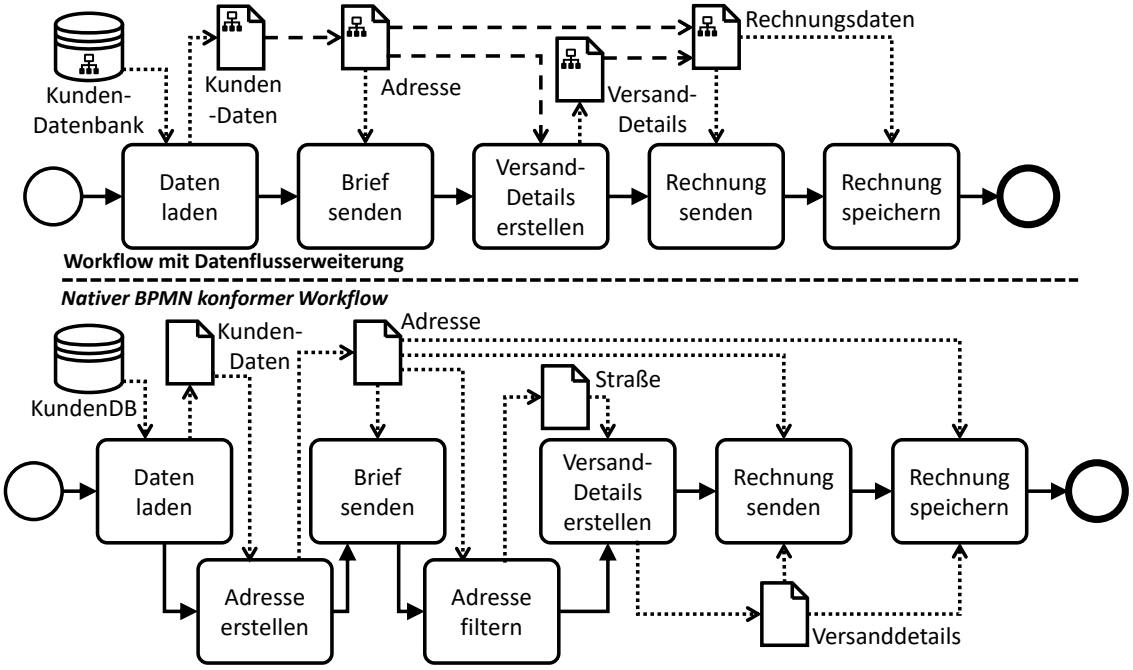

Abbildung 5.13.: Modellierung eines Workflows zur Erstellung einer Versandrechnung mit und ohne die Elemente der Datenflusserweiterung.

Der Task *Versanddetails erstellen* benötigt nur die Straße des *Adresse*-Objektes. Daher wird eine `TransformationAssociation` verwendet, um nur diesen Teil des *Adresse*-Objektes dem Task zur Verfügung zu stellen. Um dasselbe Verhalten in BPMN zu modellieren, wird zusätzlich der Task *Adresse filtern* benötigt, der die Straße der Adresse entnimmt und als Output das DataObject *Straße* erzeugt, welches als Input für den Task *Versanddetails erstellen* dient.

Die Versanddetails und die Adresse werden zum Versenden und Speichern der Rechnung benötigt. Durch die Datenflusserweiterungen können diese Daten in dem `DataMapObject` *Rechnungsdaten* gebündelt werden, wobei dies direkt im Datenfluss durch `TransformationAssociations` modelliert werden kann. Dadurch kann der benötigte Input der Tasks *Rechnung senden* und *Rechnung speichern* über das `DataMapObject` *Rechnungsdaten* modelliert werden. Ohne die Datenflusserweiterung muss der Input dieser Tasks durch zusätzliche DataAssociations modelliert werden, wie im unteren Workflow abgebildet. Dies reduziert die Übersichtlichkeit des Datenflusses.

Durch die Verwendung der Modellierungselemente der Datenflusserweiterung lässt sich der Workflow des Prozesses übersichtlicher und mit weniger Modellierungselementen modellieren. Der erweiterte Workflow benötigt weniger DataAssociations und Tasks zur Modellierung desselben Prozesses. Zusätzlich können die benötigten Daten im Datenfluss erzeugt werden und es müssen keine zusätzlichen Tasks definiert werden, um die Datenmanipulation zu modellieren.

6. Implementierung

In diesem Kapitel wird die Implementierung des Quantum Workflow Modelers vorgestellt und die Realisierung der in Kapitel 5 eingeführten Konzepte. Es werden nicht alle Details der Implementierung erläutert, lediglich die wesentlichen. Kleinere Funktionen, wie der ExtensibleButton oder die Benachrichtigungskomponente, werden an dieser Stelle nicht näher betrachtet. Ausführliche Informationen und Details zur Implementierung, sowie die Dokumentation des Quantum Workflow Modelers befindet sich im GitHub Repository des Modelers [QWM23].

6.1. Verwendete Technologien

Der Quantum Workflow Modeler ist eine Single-Page Web-Anwendung, die als HTML Web-Komponente implementiert ist. Dies ermöglicht die Integration des Modelers in Web-Anwendungen, unabhängig von dem verwendeten Framework. Der Modeler kann als benutzerdefinierte HTML Komponente der Web-Anwendung hinzugefügt und als HTML Tag verwendet werden. Der innere Aufbau der Web-Komponente ist mit dem React Framework [META23] und JavaScript [JS23] umgesetzt.

Zur Validierung der grundlegenden Funktionalität des Modelers wurden Unit-Tests implementiert. Diese Tests wurden in eine GitHub CI Pipeline integriert, um sie automatisch auszuführen und die Funktionalität des Modelers zu gewährleisten.

Ein weit verbreitetes, State-of-the-Art Modellierungswerkzeug zur Modellierung von BPMN Workflows ist der Camunda Modeler [CAM23]. Dabei handelt es sich um einen Open Source Desktop Modeler, in dem BPMN Workflows modelliert werden können. Der ebenfalls von Camunda entwickelte bpmn-js Modeler [BPMNJS23] ist die Web-basierte Open-Source-Variante des Camunda Modelers ohne Funktionen, die über das modellieren von Workflows hinausgehen, wie Speichern oder Öffnen von Workflows. Der bpmn-js Modeler kann über Dependency Injection erweitert werden [BPMNJSDOC23]. Dies erlaubt es, benutzerdefinierte BPMN Erweiterungen in den bpmn-js Modeler zu integrieren. Daher wird der bpmn-js Modeler verwendet und erweitert, um die Modellierung der Quanten-Workflows im Quantum Workflow Modeler zu ermöglichen.

Camunda stellt mit bpmn.io [BPMNIO23] einen Online Modeler zur Verfügung, der zur Modellierung den bpmn-js Modeler verwendet und zusätzliche Funktionen zum Speichern, Öffnen und Exportieren von Workflows bereitstellt. Dieser Modeler ist nicht Open Source und dient zur Demonstration, wie der bpmn-js Modeler zur Erstellung eines Web-basierten Workflow Modelers verwendet werden kann.

Camunda bietet zur Ausführung der Workflows eine eigne Workflow Engine, die Camunda Platform. Es gibt zwei Versionen: Die Camunda Plattform 7 [CAMP723] und die Camunda Platform 8 [CAMP823]. Der wesentliche Unterschied zwischen beiden Versionen ist, dass die Camunda

Platform 8 primär eine Cloud-Anwendung ist. Zusätzlich wurden einige technische Änderungen durchgeführt, die jedoch für den Anwender nicht sichtbar sind [NIA22]. Der Quantum Workflow Modeler verwendete den Stand der Camunda Platform 7, da diese im Gegensatz zur Camunda Platform 8 Open Source ist.

Das von Camunda verwendete Metamodell realisiert den BPMN Standard und wurde um zusätzliche Elemente erweitert. Bei diesen Erweiterungen handelt es sich hauptsächlich um zusätzliche Attribute, die die im BPMN Standard beschriebenen Modellierungselemente erweitern. Die graphische Notation der Modellierungselemente wurde dem BPMN Standard gemäß realisiert.

6.2. Architektur des Quantum Workflow Modelers

In Abbildung 6.1 ist die Architektur des Quantum Workflow Modelers dargestellt. Der Modeler besteht aus zwei Teilen: dem *bpmn-js Modeler* und dem *Workflow Editor*. Der bpmn-js Modeler ermöglicht die graphische Modellierung von BPMN Workflows. Er kann erweitert werden, um die Modellierung zusätzlicher Elemente zu ermöglichen. Dafür können das zugrunde liegende Metamodell, die Modellierungsregeln, die graphische Repräsentation der Elemente und die Benutzeroberfläche des bpmn-js Modelers erweitert werden. Der Zugriff auf den bpmn-js Modeler erfolgt über den Workflow Editor.

Der Workflow Editor, im Folgenden als *Editor* bezeichnet, bietet Schnittstellen zur Erweiterung des bpmn-js Modelers und seiner Benutzeroberfläche durch Plugins. Zusätzlich bietet er einige grundlegende Funktionen zum Verwalten von Workflows, wie das Speichern oder Öffnen von Workflow-Diagrammen und einige Hilfsfunktionen, wie das Ersetzen eines Modellierungselementes im bpmn-js Modeler. In die Benutzeroberfläche des Editors ist der bpmn-js Modeler eingebettet.

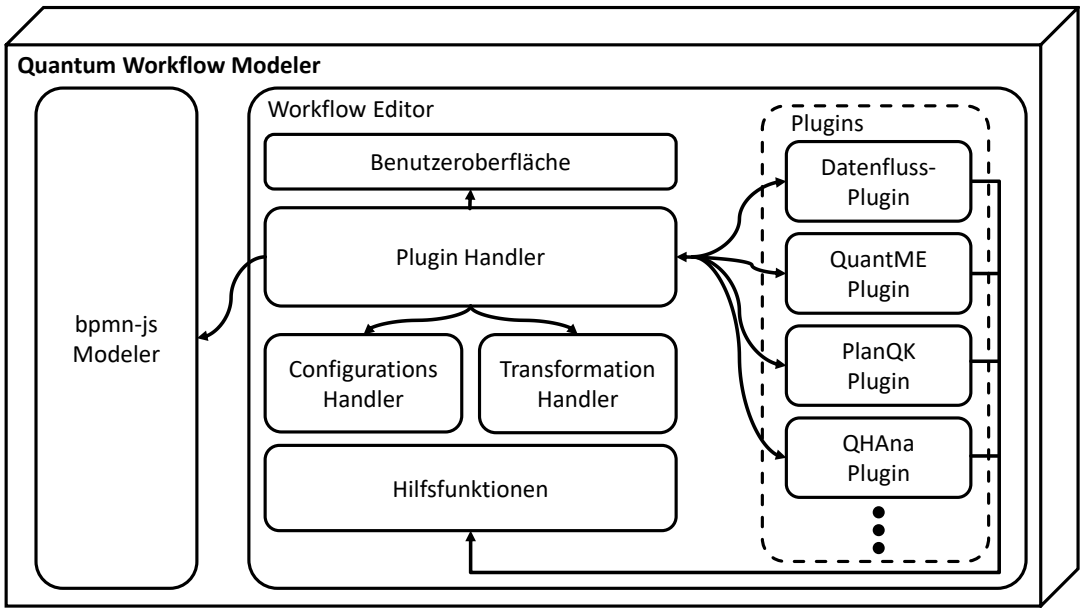

Abbildung 6.1.: Architektur des Quantum Workflow Modelers.

Über den Plugin Handler des Editors kann der Quantum Workflow Modeler Plugin-basiert erweitert werden. Ein Plugin kapselt die Funktionalität und Modellierungselemente einer Erweiterung und registriert sich beim Plugin Handler. Über den Plugin Handler kann ein Plugin auf den bpmn-js Modeler und die Benutzeroberfläche des Editors zugreifen, um diese zu erweitern. Während ein Plugin direkten Zugriff auf den bpmn-js Modeler erhält und ihn ohne Einschränkung durch den Editor erweitern kann, ist die Erweiterung der Benutzeroberfläche des Editors auf spezifische Schnittstellen beschränkt und nur über den Plugin Handler möglich. Der Plugin Handler ermöglicht es einem Plugin, zusätzliche Buttons in die Toolbar des Editors hinzuzufügen und benutzerdefinierte Einträge im Konfigurationsdialog des Editors zu erstellen. Näheres zu den Schnittstellen und dem Aufbau der Benutzeroberfläche findet sich in Abschnitt 6.3. Ein Plugin kann bei Bedarf auf eigene externe Services und Endpoints zugreifen, ohne dass diese im Editor registriert werden müssen. Eine Erweiterung kann somit in beliebigem Umfang Funktionalität in externe Services auslagern. Der Umfang und die Details des Zugriffs auf die externen Endpoints bzw. Services sind ausschließlich dem Plugin bekannt. Für den Editor ist ein Plugin eine Black Box. Beispielsweise lädt das Datenfluss-Plugin Datentransformationsalgorithmen über einen externen Endpoint, die dann als Configurations im bpmn-js Modeler verwendet werden können, ohne dass dieser Endpoint im Editor registriert wird (siehe Abschnitt 6.6.1).

Der Quantum Workflow Modeler wird von der Web-Anwendung, in die er integriert ist, initial konfiguriert. Neben einer Liste an Plugins, die in der konkreten Instanz des Modelers aktiv sein sollen, kann für jedes Plugin eine spezifische Konfiguration übergeben werden, die das Plugin über den Plugin Handler abrufen kann (siehe Abschnitt 6.6).

Der *Configurations Handler* ermöglicht die Verwendung der in Abschnitt 5.2 beschriebenen Configurations. Er bietet die benötigten Erweiterungen und einige Funktionen zur Verwendung von Configurations, über die andere Erweiterungen eigene Configurations für ihre Modellierungserweiterungen definieren und in den bpmn-js Modeler integrieren können.

Der *Transformation Handler* verwaltet die Transformation von Workflow-Modellen, die durch Plugins spezifizierte Modellierungserweiterungen verwenden, in BPMN konforme Workflow-Modelle. Bei der Registrierung eines Plugins im Plugin Handler kann jedes Plugin eine Transformationsfunktion definieren, die die BPMN Erweiterungen des Plugins in eine BPMN konforme Repräsentation überführt. Der Transformation Handler führt dann im Transformationsschritt, siehe Abschnitt 5.1.2, alle spezifizierten Transformationsfunktionen nacheinander aus, wobei das Ergebnis einer Transformationsfunktion der Input der nächsten Funktion ist. Da jede Transformationsfunktion nur ihre eigenen Erweiterungen ersetzt bzw. transformiert, ist die Reihenfolge der Ausführung der Transformationsfunktionen nicht von Bedeutung.

Das *Datenfluss-Plugin* enthält die in Abschnitt 6.4 beschriebenen Konzepte zur Erweiterung der Modellierung des Datenflusses in Workflows. Darauf aufbauend können andere Plugins bzw. Erweiterungen spezifische Datenobjekte basierend auf den Elementen der Datenflusserweiterung einführen, um den Datenfluss im Kontext der Erweiterung modellieren zu können. Der Aufbau des Datenfluss-Plugin wird in Abschnitt 6.6.1 erläutert. Das *QuantME Plugin* integriert die Modellierungserweiterungen des QuantME Transformation Frameworks [WBLW20] in den Quantum Workflow Modeler. Weitere Details zu dem Plugin werden in Abschnitt 6.6.2 ausgeführt. Das *PlanQK Plugin* ermöglicht die Modellierung von Elementen der PlanQK Plattform und wird in Abschnitt 6.6.3 ausführlicher betrachtet. Das *QHAna Plugin*, detaillierter erläutert in

Abschnitt 6.6.4, ermöglicht die Integration und Modellierung von QHAna Services im Workflow. Alle soeben genannten Plugins definieren Transformationsfunktionen, um die durch sie eingeführten Modellierungserweiterungen durch BPMN konforme Elemente zu ersetzen.

6.3. Aufbau der Benutzeroberfläche des Quantum Workflow Modelers

Die Benutzeroberfläche des Quantum Workflow Modelers ist in Abbildung 6.2 abgebildet. Sie besteht im Wesentlichen aus den Elementen der Benutzeroberfläche des Editors und den graphischen Elementen des eingebetteten bpmn-js Modelers.

Die Benutzeroberfläche des Editors besteht aus einer Toolbar und dem bpmn-js Modeler. Die Toolbar gliedert sich in drei Abschnitte: Zunächst enthält sie die grundlegenden, generischen Editorfunktionen in Form von Buttons. Diese Funktionen sind nicht spezifisch für Workflows, sondern die in ihnen enthaltene Funktionalität findet sich allgemein in vielen Editoren, auch wenn sie an den Workflow-Kontext angepasst wurden, wie das Speichern der Workflows oder die Konfiguration des Modelers. Der zweite Abschnitt enthält Funktionen, die spezifisch für Workflows entwickelt wurden, wie das Deployment der Workflows in eine Workflow Engine. Der dritte Abschnitt enthält Buttons, die jedes Plugin bei Bedarf hinzufügen kann, um eigene Features in die Benutzeroberfläche zu integrieren.

Die Benutzeroberfläche des bpmn-js Modelers besteht aus der sogenannten *Palette* (links), der Modellierungsebene im zentralen Teil und dem *Properties Panel* (rechts). Die Palette enthält Modellierungselemente, um diese direkt, ohne ein zusätzliches Menü, mittel Drag-and-Drop in das Workflow-Modell einbauen zu können. In der Modellierungsebene findet die eigentliche Modellierung statt. Hier kann der Modellierer die Elemente eines Workflows modellieren. Neue Elemente können entweder über die Palette hinzugefügt werden oder direkt an ein bereits modelliertes

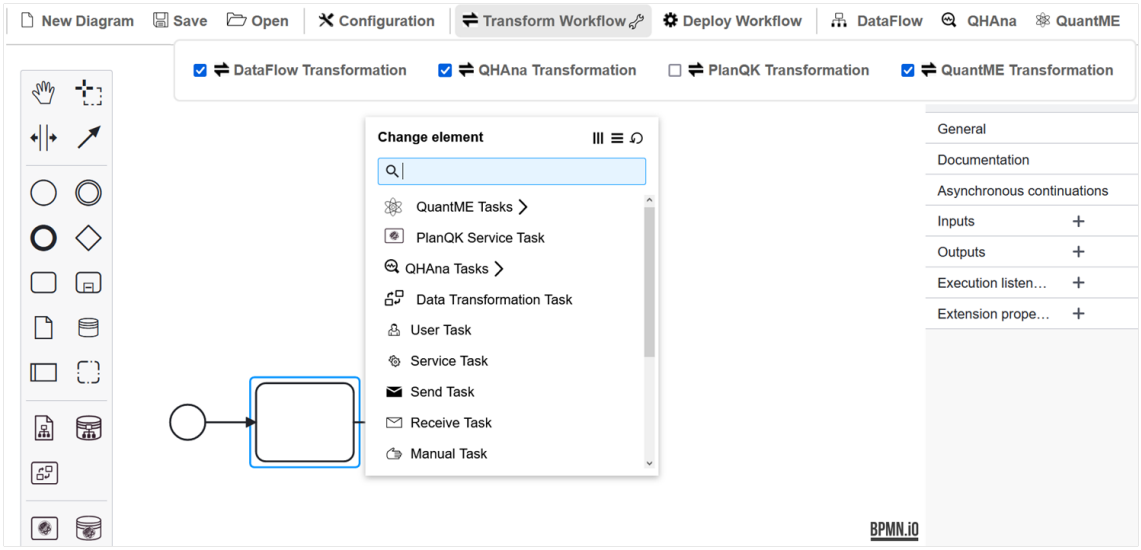

Abbildung 6.2.: Screenshot der Benutzeroberfläche des Quantum Workflow Modelers.

Element über das *Context Pad* angehängt werden. Über das Context Pad kann das sogenannte *Replace-Menü* geöffnet werden. Über dieses Menü werden dem Modellierer eine Reihe von Optionen angezeigt, durch die das aktuell ausgewählte Element ersetzt werden kann, im Folgenden auch als Ersetzungskandidaten bezeichnet. Beispielsweise enthält das Replace-Menü für einen Task Einträge für einen Service Task und alle anderen im BPMN Standard definiert Task-Typen. Wird ein Eintrag ausgewählt, wird das Element durch das, über den Eintrag repräsentierte Element ersetzt. Das Properties Panel zeigt die Attribute des aktuell ausgewählten Elementes an. Über das Properties Panel können einzelne Attribute bearbeitet werden. Alle soeben beschriebenen Elemente der Benutzeroberfläche des bpmn-js Modelers können erweitert und an benutzerdefinierte Erweiterungen angepasst werden.

Um den Anforderungen an die einfache Benutzbarkeit (siehe Kapitel 4) gerecht zu werden, wurden einige Konstrukte definiert und Erweiterungen am bpmn-js Modeler vorgenommen:

Der 'Transform Workflow'-Button führt alle durch Plugins spezifizierten Transformationsfunktionen aus. Über ihn kann das aktuell im bpmn-js Modeler geladene Workflow-Modell transformiert werden. Um selbst bei vielen Plugins die Übersichtlichkeit zu bewahren und zusätzlich dem Modellierer die Möglichkeit zu bieten, gezielt einzelne Transformationsfunktionen zu aktivieren bzw. zu deaktivieren, kann durch Klicken auf das Werkzeug-Icon der 'Transform Workflow'-Button aufgeklappt werden. Dadurch werden alle Transformationsfunktionen der Plugins, die aktuell im Modeler spezifiziert sind, angezeigt. Der Modellierer kann dann durch Auswählen der Checkboxen der jeweiligen Transformationsfunktion einzelne Transformationsfunktionen aktivieren oder deaktivieren, wie in Abbildung 6.2. Wird der 'Transform Workflow'-Button angeklickt, so werden alle aktivierten Transformationsfunktionen der Reihe nach ausgeführt.

Um verwendete Endpoints bzw. externe Services zur Laufzeit des Modelers konfigurieren zu können, gibt es den *Konfigurationsdialog*. Er kann über den 'Configurations'-Button der Toolbar geöffnet werden und enthält eine Reihe von Tabs, wie in Abbildung 6.3 zu sehen ist. Plugins können

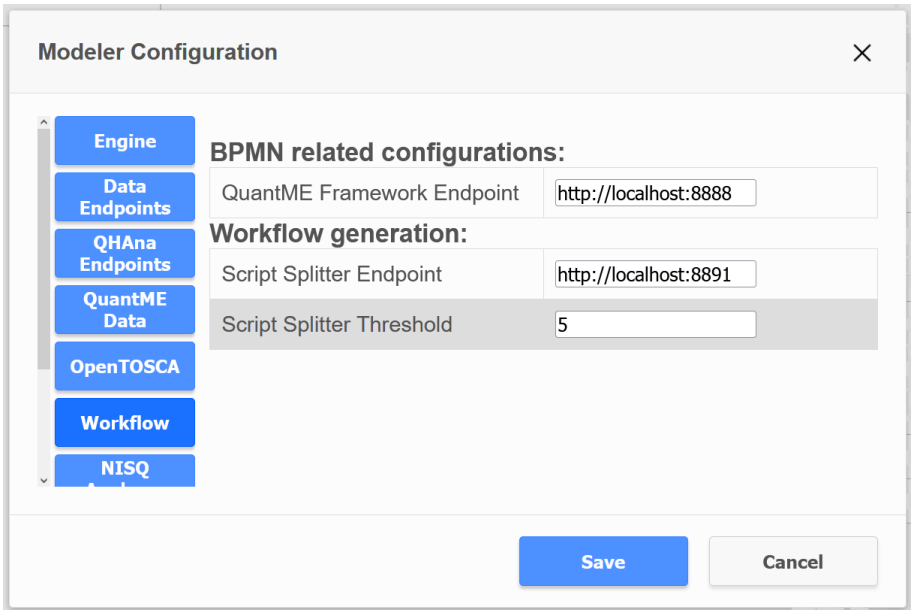

Abbildung 6.3.: Screenshot des Konfigurationsdialoges.

solche Tabs spezifizieren, um ihre Erweiterungen zur Laufzeit konfigurieren zu können. Der Inhalt eines Tabs wird vollständig vom jeweiligen Plugin definiert und als React Komponente bei der Registrierung des Plugins an den Modeler übergeben.

Modellierungselemente, die von Erweiterungen bzw. deren Plugins definiert werden, müssen über die Benutzeroberfläche des bpmn-js Modelers ausgewählt werden können, um in einem Workflow-Modell verwendet zu werden. Eine Möglichkeit hierfür ist das Replace-Menü des bpmn-js Modelers. In ihm können neue Elemente als Ersetzungskandidaten für das gerade ausgewählte Workflow-Element definiert werden. Falls eine Erweiterung jedoch viele solcher Ersetzungskandidaten hinzufügt oder mehrere Plugins Kandidaten für dasselbe Workflow-Element definieren, kann dieses Menü unübersichtlich werden. Um dem entgegen zu wirken, kann der sogenannte *MoreOptionsEntry* verwendet werden. Er ermöglicht die hierarchische Gliederung der Menüeinträge. Ein MoreOptionsEntry ist ein Menüeintrag, der ein Menü mit weiteren Optionen öffnet, falls er ausgewählt wird, ähnlich dem Öffnen eines Ordners in einem File Explorer. In Abbildung 6.4 ist ein MoreOptionsEntry für QuantME Tasks zu sehen. Wird er ausgewählt öffnet er das rechts abgebildete Menü, das die eigentlichen Einträge enthält. Über den ersten Eintrag kann man zurück zum ursprünglichen Menü gelangen. Die Einträge hinter einem MoreOptionsEntry können ebenfalls MoreOptionsEntries enthalten, sodass hierarchische Einträge beliebiger Tiefe erstellt werden können. Die Suchfunktion des Replace-Menüs ist an die hierarchische Struktur angepasst und ermöglicht so die Suche in allen Einträgen, auch den hinter MoreOptionsEntries verborgenen. Dies ermöglicht die Darstellung von vielen Ersetzungskandidaten verschiedener Plugins in einer übersichtlichen Art und Weise, durch die einfach navigiert werden kann.

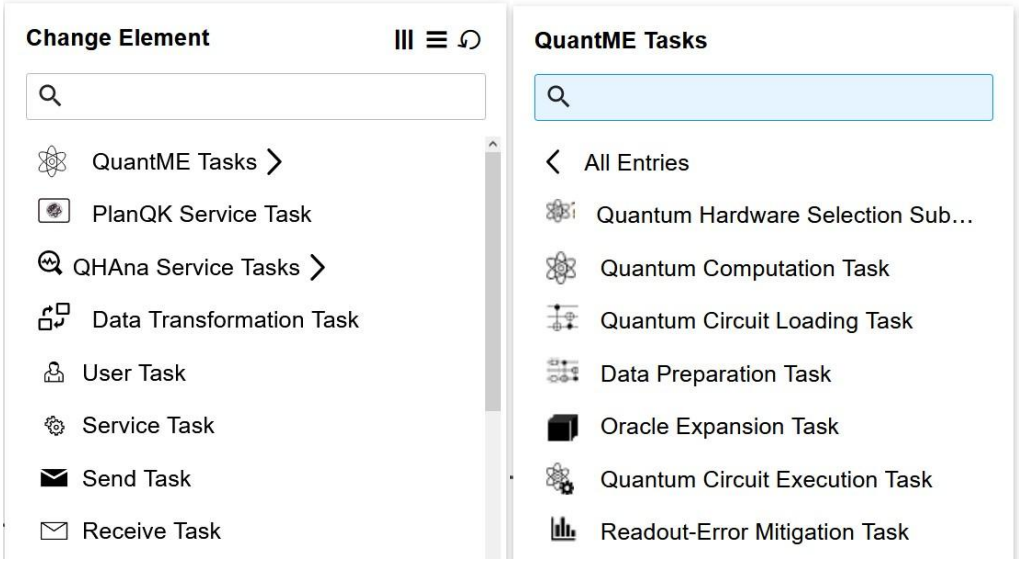

Abbildung 6.4.: Screenshot des QuantME MoreOptionsEntries.

6.4. Implementierung der Datenflusserweiterung

In diesem Kapitel wird die Implementierung der in Abschnitt 5.3 eingeführten Konzepte beschrieben. Die Implementierung der Datenflusserweiterung erfolgt durch Erweiterung des bpmn-js Modelers [BPMNJS23] bzw. des Metamodells von Camunda. Wie bereits in Kapitel 3 diskutiert, erlaubt die Camunda BPMN Erweiterung zwar die Modellierung des Datenflusses, wie es der BPMN 2.0 Standard vorsieht, jedoch dient diese Modellierung lediglich Dokumentationszwecken und wird weder von der Workflow Engine der Camunda Plattform 7 [CAMP723] noch der Camunda Plattform 8 [CAMP823] bei der Ausführung des Workflows berücksichtigt. Eine graphische Modellierung des Datenflusses, die von der Camunda Workflow Engine berücksichtigt wird, ist durch die Datenflusserweiterung möglich.

Neben der Beschreibung der Implementierung der Modellierungselemente der Datenflusserweiterung in Abschnitt 6.4.1, wird in Abschnitt 6.4.2 die Transformationsfunktion beschrieben, um die Modellierungselemente vor der Ausführung in einer Camunda Workflow Engine durch Elemente der Camunda BPMN Erweiterung zu ersetzen, die von der Camunda Workflow Engine berücksichtigt werden. Bei dieser Transformation entsteht ein Workflow-Model, dass der Camunda BPMN Erweiterung entspricht und ein Datenschema enthält, dass von der Camunda Workflow Engine interpretiert werden kann.

6.4.1. Implementierung der Modellierungselemente der Datenflusserweiterung

Die in Abschnitt 5.3 beschriebenen Modellierungselemente sind als Modellierungserweiterung des bpmn-js Modelers implementiert. Die Implementierung folgt dabei dem XSD-Schema der Datenflusserweiterung (siehe Anhang A). Die `DataMapObject`-, `DataStoreMap`- und `TransformationTask`-Klassen erben von denr DataObject-, DataStore- und Task-Klassen des bpmn-js Modelers und erweitert diese jeweils um ihre spezifisches Attribut, das `Content`-, `Details`- und `Parameters`-Attribut.

Die `TransformationAssociation`-Klasse der Datenflusserweiterung erweitert die Association-Klasse des bpmn-js Modelers um das `Expressions`-Attribut. Zur Modellierung von ein- bzw. ausgehenden `TransformationAssociations` werden zwei zusätzliche Klassen eingeführt, die `InputTransformationAssociation`-Klasse und die `OutputTransformationAssociation`-Klasse. Diese erben von der `TransformationAssociation` und sind nötig, um die Richtung der Pfeile der `TransformationAssociation` entsprechend darzustellen. Zur Beschreibung der Expressions einer `TransformationAssociation` wird die sogenannte Camunda Expression Language [CAMP723] verwendet. Die Camunda Expression Language verwendet die Open Source Java Unified Expression Language Implementierung (JUEL) [JUEL14]. Die Ausdrücke der Expression Language werden zur Laufzeit von der Workflow Engine evaluiert und können logische Ausdrücke, Zuweisungen oder einfache Operationen, wie beispielsweise String-Konkatenationen oder Additionen, enthalten. In den Ausdrücken kann dabei über den Namen von Prozessvariablen auf sie Bezug genommen werden. Dies ermöglicht es, den Wert einer Prozessvariable während der Ausführung zu manipulieren. Beispielsweise kann einer Prozessvariable a so der Wert der Summe von b und c zugewiesen werden, wobei a, b und c Prozessvariablen sind.

6.4.2. Transformationsfunktion der Datenflusserweiterung

Damit die mit der Datenflusserweiterung modellierten Daten in der Ausführung des Workflow-Modells verfügbar sind, müssen die Elemente der Datenflusserweiterung im Transformationsschritt vor der Ausführung in ein von der Camunda Workflow Engine berücksichtigtes Format transformiert werden.

Das für die Ausführung relevante Datenschema sind Prozessvariablen [CAMP723]. Sie sind in der Camunda BPMN Erweiterung enthalten und können im Prozesskontext oder lokal im Kontext einer Activity definiert werden. Auf diese Prozessvariablen und die in ihnen gespeicherten Daten kann während der Ausführung des Workflows über den Namen der Variable zugegriffen werden. Die Sichtbarkeit und Verfügbarkeit von Prozessvariablen ist dabei ähnlich der eines DataObjects in BPMN 2.0: Eine Prozessvariable existiert, sobald der sie erzeugende Prozess oder die sie erzeugende Activity aktiviert wurde und ist für alle im Kontrollfluss nachfolgenden Activties sichtbar. Abweichend vom BPMN 2.0 Standard kann allerdings eine Prozessvariable, die in einem SubProcess erstellt wurde auch von nachfolgenden Activities im Elternprozess verwendet werden. Teil der Camunda Erweiterung sind Attribute, über die Input- und Output-Variablen als Prozessvariablen dynamisch und in beliebiger Anzahl für eine Activity definiert werden können. Input-Variablen erzeugen dabei keine neuen Prozessvariablen. Sie sind lediglich im lokalen Kontext verfügbar. Über sie können Prozessvariablen auf lokale Variablen abgebildet werden. Output-Variablen dagegen erzeugen automatisch neue Prozessvariablen im Prozesskontext oder überschreiben den Wert einer bereits existierenden Prozessvariable mit demselben Namen.

Der Wert einer Prozessvariable kann zur Modellierungszeit hart codiert, dynamisch zur Laufzeit gesetzt oder über die Camunda Expression Language definiert werden.

Im Zuge der Transformation der Datenflusserweiterung werden alle `DataMapObjects`, `DataStoreMaps`, `TransformationAssociations` und `TransformationTasks` durch DataObjects, DataStores, DataAssociations und Tasks ersetzt. Aus technischer Sicht könnten die Datenflusserweiterungen aus dem Diagramm entfernt werden, da die Datenflusselemente der Camunda BPMN Erweiterung von der Workflow Engine nicht berücksichtigt werden. Für das bessere Verständnis des Datenflusses im transformierten Diagramm und zu Dokumentationszwecken findet eine Ersetzung statt, sodass relevante Informationen zum Datenfluss dem Modellierer nach dem Transformationsschritt noch graphisch zur Verfügung stehen. Der Inhalt der ersetzten `DataMapObjects` und `DataStoreMaps` wird in den Documentation-Attributen der sie ersetzenden Camunda Datenobjekte gespeichert. Dadurch kann der Wert eines DataObjects bzw. DataStores weiterhin dem jeweiligen Objekt entnommen werden, obwohl die eigentlichen Daten nun nicht mehr in den Datenobjekten zu finden sind.

Die in den Attributen der Datenflusserweiterungen spezifizierten Daten werden im Transformationsschritt in Prozessvariablen umgewandelt, damit die in ihnen gespeicherten Eigenschaften von der Workflow Engine berücksichtigt werden. Das entstehende Diagramm enthält eine Menge an Prozessvariablen, die, abhängig vom Typ der Prozessvariable, zu einem festen Zeitpunkt und im entsprechenden Kontext erstellt werden und einen, während der Ausführung unveränderlichen, Namen besitzen. Diese Eigenschaften, d.h. der Zeitpunkt der Erstellung und der Name der Prozessvariablen sind statisch und ändern sich zur Laufzeit nicht. Somit kann die Modellierung des Datenflusses den Kontrollfluss nicht beeinflussen, da der Datenfluss in ein statisches Datenschema transformiert wurde. Der Inhalt der Prozessvariablen kann sich dagegen zur Laufzeit verändern. Er kann zwar im Transformationsschritt oder manuell vom Modellierer gesetzt werden, jedoch kann er in der Ausführung des Workflows überschrieben werden.

Das aus dem Transformationsschritt resultierende Workflow-Modell entspricht der Camunda BPMN Erweiterung und kann somit in der Camunda Workflow Engine ausgeführt werden. Diese Transformation wird im Folgenden genauer beschrieben.

Transformation von DataMapObjects und DataStoreMaps

`DataMapObjects` werden durch DataObjects ersetzt, wobei die mit ihnen verbunden DataAssociations zu Dokumentationszwecken erhalten bleiben. Das `Content`-Attribut wird in eine Prozessvariable umgewandelt. Die Prozessvariable übernimmt den Namen des `DataMapObjects` und enthält als Wert alle Key-Value-Paare des `Content`-Attributes. Somit steht der Inhalt des `DataMapObjects` nun in der Workflow Engine über die Prozessvariable zur Verfügung. Um welche Art von Prozessvariable es sich dabei handelt, wird von den ein- bzw. ausgehenden DataAssociations bestimmt und durch folgende Regeln definiert:

1. **DataMapObject -> Activity**
   Für eine vom `DataMapObject` ausgehende DataAssociation hin zu einer Activity, wird dieser Activity eine Input-Variable hinzugefügt, die den Namen des `DataMapObjects` trägt und als Wert den Inhalt des `Content`-Attributes, d.h. die Key-Value-Paare, enthält.
2. **Activity -> DataMapObject**
   Einer Activity, die eine, zu einem `DataMapObject` führende, DataAssociation besitzt, wird eine Output-Variable mit dem Namen des `DataMapObjects` und dem Inhalt des `Content`-Attributes hinzugefügt.
3. **DataMapObject ohne Initialisierung**
   Hat ein `DataMapObject` keine eingehenden DataAssociations bzw. wird sein Wert durch eine ausgehende DataAssociation abgefragt, bevor er durch eine eingehende DataAssociation gesetzt wurde, so wird eine entsprechende Prozessvariable im aktuellen Prozesskontext für das `DataMapObject` angelegt. Diese Variable trägt den Namen des `DataMapObjects` und den Wert des `Content`-Attributes.

Ist ein StartEvent mit einem `DataMapObject` verbunden, so wird für jeden Eintrag seines `Content`-Attributes eine entsprechende Prozessvariable als FormData [CAMP723] hinzugefügt. Dies ermöglicht die Modellierung von Prozess-Inputs, die zu Beginn der Ausführung des Workflows durch Camunda Forms [CAMP723] definiert werden können. Der so definiert Wert wird dann unter dem Namen der FormData, in diesem Fall unter dem Namen des Eintrags des `Content`-Attributes, als Prozessvariable zur Verfügung gestellt. Dadurch kann der Modellierer Prozess-Inputs modellieren, die für jede Ausführung individuell spezifiziert werden können.

`DataStoreMaps` werden im Transformationsschritt durch DataStores ersetzt. Die mit ihnen verknüpften DataAssociations bleiben bestehen. Analog zur Transformation von `DataMapObjects` werden die `Details`-Attribute durch Prozessvariablen ersetzt. Diese Prozessvariable, die den Namen der `DataStoreMap` trägt und den Inhalt des `Details`-Attributes enthält, wird im Prozesskontext erzeugt und ist somit für alle Activities dieses Prozesses, unabhängig von Verbindungen über DataAssociations, verfügbar. DataAssociations zwischen `DataStoreMaps` und Activities dienen demnach lediglich zu Dokumentationszwecken. Der Grund hierfür ist, dass `DataStoreMaps` zur Modellierung von persistenten Datenspeichern dienen. Eine `DataStoreMap` wird, wie ein DataStore, verwendet, um eine Referenz auf einen Datenspeicher zu modellieren. Ein- bzw- ausgehende DataAssociations

modellieren keine Änderungen an der `DataStoreMap`, sondern Änderungen in dem durch sie repräsentierten Datenspeicher. Somit wäre es wenig sinnvoll, für jede zu einer `DataStoreMap` führende DataAssociation eine Output-Variable zu erzeugen, da sich nicht die Attribute der `DataStoreMap` ändern, sondern der Inhalt des Datenspeichers. Deshalb wird für eine `DataStoreMap` eine Prozessvariable im globalen Kontext angelegt.

Camunda bietet keine direkte Lösung, um Prozessvariablen mit beliebigem Inhalt im Prozesskontext zu Beginn eines Prozesses zu erzeugen [CAMP723]. Prozessvariablen mit atomarem Inhalt, wie Zahlen oder Strings können durch FormData-Einträge oder sogenannte ExecutionListener erzeugt werden. Über diese Varianten können aber keine komplexen Variablen mit einer internen Struktur erzeugt werden, wie beispielsweise Listen oder Key-Value-Maps [CAMP723]. Daher sind diese Ansätze nicht geeignet, um den Inhalt eines `DataMapObjects` oder einer `DataStoreMap` als Prozessvariable im Prozesskontext zu erzeugen. Um dennoch solche Prozessvariablen erzeugen zu können, wird ein neuer Task durch die Transformation erzeugt, der jede Prozessvariable, die im aktuellen Prozesskontext erzeugt werden soll, als Output-Variable enthält. Dieser Task, im Folgenden als *ProcessVariablesTask* bezeichnet, wird direkt nach jedem StartEvent des entsprechenden Elternprozesses eingefügt und erzeugt dadurch noch vor der Ausführung des ersten Tasks des ursprünglichen Workflows alle benötigten globalen Prozessvariablen.

In Abbildung 6.5 ist eine Beispieltransformation abgebildet. Der originale Workflow enthält Elemente der Datenflusserweiterung, die im transformierten Workflow ersetzt wurden. Das `Details`-Attribut der `DataStoreMap` *DataRepository* wurde dem ProcessVariablesTask als Output-Variable hinzugefügt, um im Prozesskontext verfügbar zu sein. Das `DataMapObject` *Adresse* wurde durch ein DataObject ersetzt und das `Content`-Attribut wurde entsprechend der DataAssociations zu *Task2* als Output- und zu *Task3* als Input-Variable hinzugefügt. Die erzeugten Prozessvariablen tragen jeweils den Namen der Datenelemente.

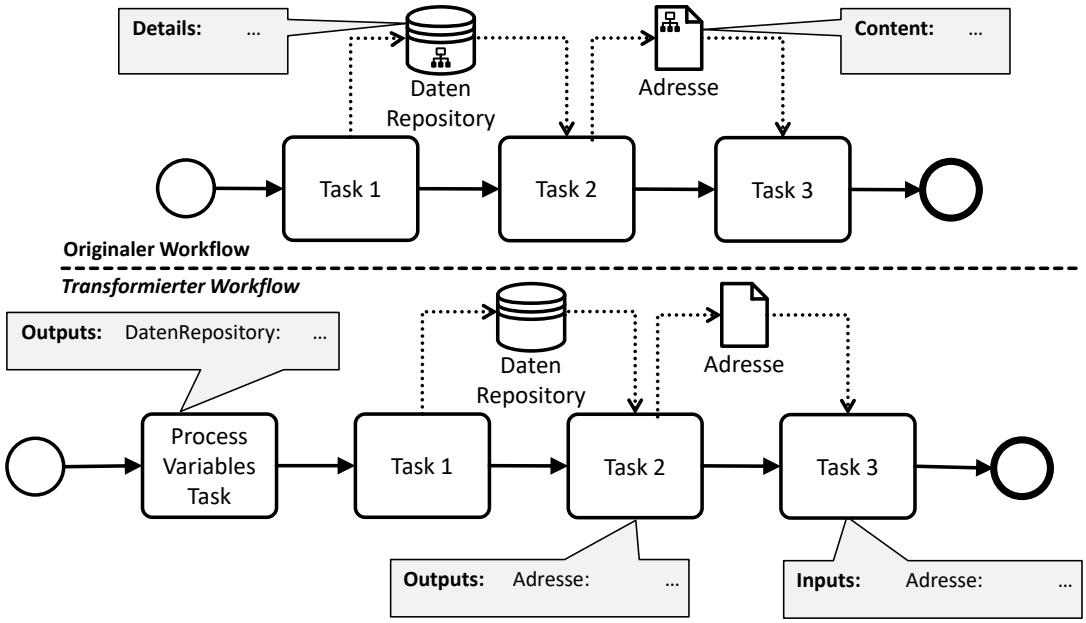

Abbildung 6.5.: Beispiel für eine Transformation von `DataMapObjects` und `DataStoreMaps`.

Transformation von TransformationAssociations

Ist ein DataMapObject über eine TransformationAssociation mit einer Activity verbunden, wird das DataMapObject nicht wie im vorherigen Abschnitt beschrieben, transformiert. Die im Expressions-Attribut durch die Camunda Expression Language definierten Transformationen werden ohne Veränderung in die erstellten Prozessvariablen übernommen. Die Auswertung der formalen Ausdrücke wird zur Laufzeit von der Camunda Workflow Engine übernommen. TransformationAssociations und mit ihnen verbundene DataMapObjects werden abhängig vom Typ des Zielobjektes der TransformationAssociation transformiert:

1. **DataMapObject -> Activity**
   Handelt es sich bei dem Zielobjekt der TransformationAssociation um eine Activity, wird die TransformationAssociation im Transformationsschritt durch eine DataAssociation ersetzt. Für jedes im Expressions-Attribut spezifizierte Key-Value-Paare wird der Activity eine Input-Variablen hinzugefügt, wobei der Key des Paares der Name und der Value der Wert der Input-Variable ist.

2. **DataMapObject -> DataMapObject**
   Verbindet eine TransformationAssociation zwei DataMapObjects, kann keine DataAssociation eingesetzt werden. Die beiden Objekte sind nach dem Transformationsschritt nicht verbunden. Um dennoch die Verbindung nachvollziehbar zu dokumentieren, wird in dem Documentation-Attribut des Zielobjektes der TransformationAssociation beschrieben, durch welche Transformation und aus welchem Startobjekt es entstand. Die in dem Extensions-Attribut beschriebenen Transformationen werden als Key-Value-Paare in das Content-Attribut des Ziel-DataMapObjects übertragen. Das Start- und Zielobjekt werden jeweils auf eine entsprechende Prozessvariable abgebildet, die durch den ProcessVariablesTask im aktuellen Prozess- bzw. SubProcess-Kontext angelegt wird, damit alle nachfolgenden Activities darauf Zugriff erhalten. Wird das Ziel-DataMapObject durch DataAssociations oder TransformationAssociation referenziert, wird es entsprechend der bereits vorgestellten Regeln auf Input- bzw Output-Variablen abgebildet.

In Abbildung 6.6 ist die Transformation von TransformationAssociations beispielhaft dargestellt. Das DataMapObject *Adresse* wird durch eine TransformationAssociation aus dem DataMapObject *Nutzerdaten* erzeugt, wobei es zwei Attribute durch die TransformationAssociation erhält, *a* und *b*. Der genaue Wert dieser Attribute wird von der Workflow Engine durch Auswertung der formalen Expression bestimmt. Eine Expression wird durch '${}' gekennzeichnet. Durch die Transformation des Workflows wird die TransformationAssociation zwischen *Nutzerdaten* und *Adresse* entfernt und es werden zwei Output-Variablen im ProcessVariablesTask erzeugt, um den Inhalt des Content-Attributes von *Adresse* und *Nutzerdaten* im aktuellen Prozesskontext zu veröffentlichen. Der Input von *Task3* wird über eine TransformationAssociation zwischen *Nutzerdaten* und *Task3* modelliert, wobei der Inhalt von *Nutzerdaten* nicht übernommen wird. Der Input für *Task3* besteht aus den Attributen *x* und *y* bzw. aus dem Ergebnis der in ihnen spezifizierten formalen Expressions. Im Zuge der Transformation des Workflows wird diese TransformationAssociation durch eine DataAssociation ersetzt und die Einträge *x* und *y* des Expressions-Attributes werden zu Input-Variablen von *Task3*.

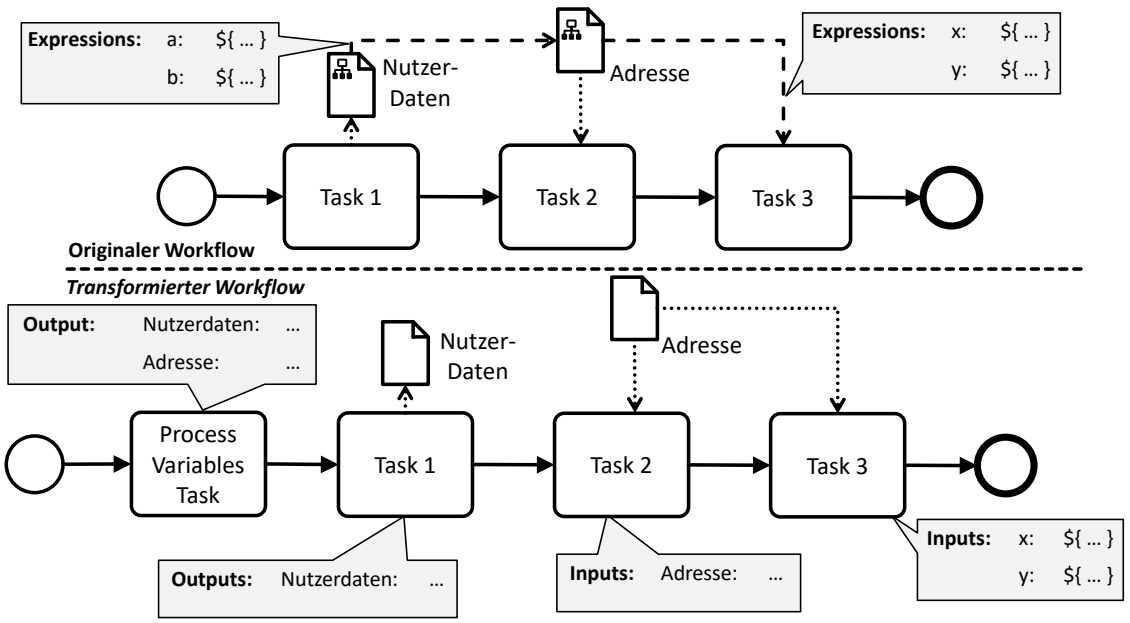

Abbildung 6.6.: Beispiel der Transformation von `TransformationAssociations`.

Transformation von TransformationTasks

`TransformationTasks` werden im Transformationsschritt durch ServiceTasks ersetzt. Das `Parameters`-Attribut wird als Input-Variable dem Task hinzugefügt und enthält alle ursprünglichen Key-Value-Paare. Die durch DataAssociations mit dem `TransformationTask` verbundene `DataMapObjects` werden entsprechend des Typs der Assoziation zu Input- bzw. Output-Variablen des ServiceTasks.

Abbildung 6.7 zeigt ein Beispiel einer solchen Transformation. Der `TransformationTask` *JSONtoXML* enthält Konfigurationsparameter in seinem `Parameters`-Attribut zur Spezifikation des Encodings und des Namespaces der durch den `TransformationTask` erzeugten XML Daten. Der Input bzw. der Output des Tasks wird durch die `DataMapObjects` *JSONDaten* und *XMLDaten* modelliert. Durch die Transformation wird der `TransformationTask` durch einen ServiceTask ersetzt und das `Parameters`-Attribut wird zu einer Input-Variable des neu erstellten ServiceTasks.

6.5. Implementierung des Configurations-Konzeptes

Die in Abschnitt 5.2 eingeführten Configurations basieren auf dem Konzept der Camunda Element Templates [CAMP823], über die domänenspezifische Element modelliert werden können. Ein Element Template kann für eine Activity, einen SequenceFlow, ein Event oder einen Process erstellt werden und ermöglicht die Definition von benutzerdefinierten Namen, Icons und Attributen. Es ist ein leichtgewichtiger Ansatz, um einen domänenspezifischen Workflow zu modellieren, ohne eine Erweiterung des BPMN Standards vorzunehmen. Jedes Attribut im Element Template wird auf ein Attribut des zugrundeliegenden XML Schemas des Workflows abgebildet. Dieses Binding wird im Element Template definiert. Mit Element Templates können neue Typen für BPMN Elemente simuliert werden. Im XML Schema und dem Metamodell des Workflows wird jedoch kein neuer

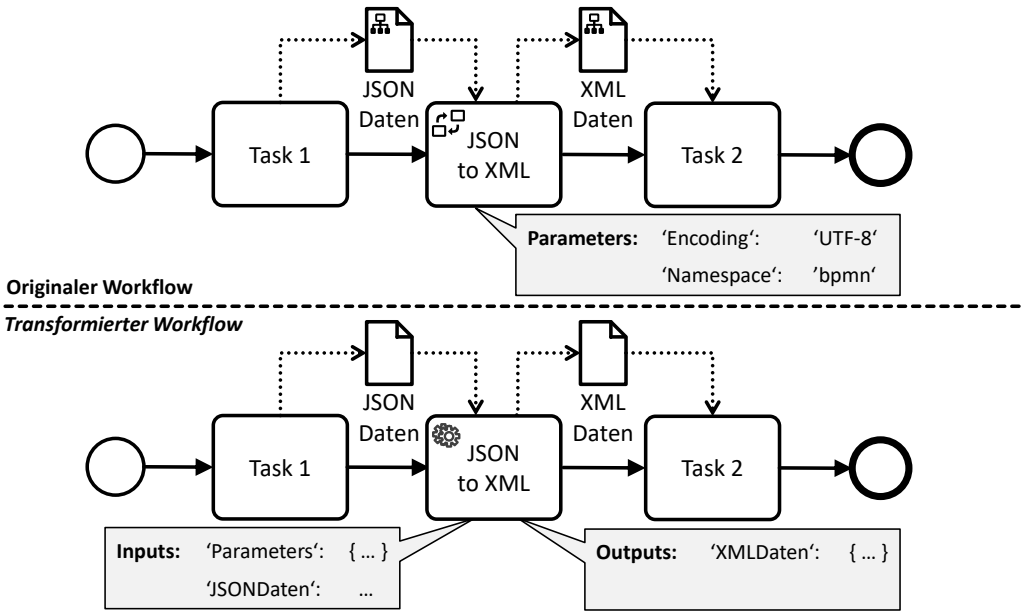

Abbildung 6.7.: Beispiel einer Transformation eines Transformation-Tasks.

Typ hinzugefügt. Element Templates erweitern die graphische Notation des BPMN Elementes. Im zugrundeliegenden XML Schema wird jedoch kein neuer Typ erzeugt. Element Templates können für ein entsprechendes BPMN Element ausgewählt werden, wodurch die im Template definierten Attribute und deren Belegungen auf das BPMN Element angewandt werden. Beispielsweise kann so der Wert eines Attributes über ein Element Template gesetzt werden. Durch die Auswahl eines Element Templates werden alle Attribute des BPMN Elementes ausgeblendet und der Modellierer hat lediglich noch auf die im Element Template definierten Attribute Zugriff. Die Werte der Attribute werden dann, wie im Binding spezifiziert, auf andere Attribute abgebildet, wobei es sich um Camunda spezifische Attribute oder im BPMN Standard definierte Attribute handeln kann. Eine Abbildung auf Attribute, die durch eine benutzerdefinierte Erweiterung definiert werden, ist nicht möglich [CAMP823].

Element Templates sind für den Quantum Workflow Modeler nicht geeignet, da (i) sie bzw. die in ihnen spezifizierten Bindings nicht für BPMN Erweiterungen definiert werden können, im Gegensatz zu Configurations. Außerdem (ii) können nach Auswahl eines Element Templates die anderen Attribute eines Elementes nicht mehr im Properties Panel bearbeitet werden. Dadurch muss ein Element Template die bereits enthaltenen Attribute eines BPMN Elementes erneut definieren, um sie nach der Auswahl des Element Templates bearbeiten zu können. Configurations dagegen beheben diese Einschränkungen und erweitern den Ansatz der Element Templates, weswegen sie im Quantum Workflow Modeler implementiert und verwendet werden.

Zusätzlich zu den im Konzept beschriebenen Funktionen, bietet die Implementierung der Configurations die Möglichkeit, die graphische Repräsentation der Attribute im Properties Panel zu steuern, in dem ein Icon als SVG definiert werden kann. Zur Realisierung des Bindings neuer Attribute wurden die in Abschnitt 5.2 beschriebenen Einträge eines Configurations-Attributes um weitere Einträge erweitert.

Diese zusätzlichen, für die Implementierung des Konzeptes relevanten Einträge sind im Folgenden aufgelistet:

- **Label**: Text, der das Label des Eintrags im Properties Panel beschreibt.
- **Typ**: Der Typ des Attributes, der die Repräsentation im Properties Panel steuert. Verfügbare Typen sind String und Boolean.
- **Deaktiviert**: Ein Boolean, der angibt, ob der Wert des Attributes im Properties Panel bearbeitet werden kann oder nicht. (optional)
- **Versteckt**: Ein Boolean, der angibt, ob das Attribut im Properties Panel nicht angezeigt werden soll. (optional)
- **Binding**: Das Binding ist in der Implementierung ein JavaScript-Objekt und besteht aus einem Namen und einem Typ. Der Name definiert, wie im Konzept, den Namen des Attributes, in dem der Wert des Configurations-Attributes gespeichert werden soll. Der Typ spezifiziert dabei den Typ des Binding-Attributes, um korrekt formatierte Speicherung zu ermöglichen. Der Wert des Configurations-Attributes wird in ein Element des definierten Typs umgewandelt und dann in dem durch den Namen spezifizierten Attribut gespeichert. Verfügbare Typen sind Camunda-Input-Parameter, Camunda-Output-Parameter, Key-Value-Map und String. Bei Camunda-Input- bzw. Output-Parameter wird der Wert als Input- bzw. Output-Parameter hinzugefügt. Bei Key-Value-Map wird der Wert als Key-Value-Eintrag gespeichert.

In Abbildung 6.8 sind Beispiele für die verschiedenen graphischen Repräsentationen der Attribute abgebildet. Links ist ein String-Attribut und rechts ein Boolean-Attribut zu sehen. Darunter befinden sich die jeweiligen deaktivierten Varianten.

Um die graphische Repräsentation einer Configuration zu spezifizieren, kann ein Icon definiert werden, dass die graphische Notation des Elementes, auf das die Configurations angewandt wurde, darstellt. Dadurch kann für eine Configuration eine individuelle graphische Notation definiert werden. Beispielsweise wird dies verwendet, um quantenspezifische `DataMapObjects` für das QuantME Plugin aus Abschnitt 6.6.2 zu definieren. Das Icon wird durch einen zusätzlichen, optionalen **Icon**-Eintrag der Configuration definiert. Ein Icon-Eintrag besteht aus zwei verpflichtenden Einträgen, **Transformation** und **SVG**. Im SVG-Eintrag ist das Icon als SVG in einem String enthalten. Der Transformation-Eintrag beschreibt eine Transformationsmatrix, über die das Icon skaliert und verschoben wird.

Um neue Configurations einfach in neue Erweiterungen bzw. Plugins integrieren zu können, wurden entsprechende Hilfsklassen und -funktionen entwickelt. Diese umfassen unter anderem Funktionen, über die die Attribute einer Configuration automatisch im Propeties Panel, gemäß ihrer spezifizierten

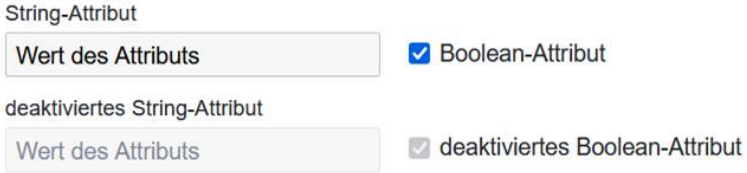

Abbildung 6.8.: Beispiele für die verschiedenen graphischen Repräsentationsmöglichkeiten eines ConfigurationsAttributs

Eigenschaften, angezeigt werden. Um Configurations für ein Element einfach auswählen zu können, besteht die Möglichkeit über die Hilfsfunktionen einen Menüeintrag für das Replace-Menü zu erstellen, der alle für das ausgewählte Workflow-Element verfügbaren Configurations als einen MoreOptionsEntry enthält. Dadurch kann der Modellierer über das Replace-Menü Configurations einfach und direkt auswählen.

Der *Configurations Endpoint* verwaltet die Kommunikation mit dem Repository der Configurations, das beispielsweise ein Server Endpoint sein kann und verwaltet die geladenen Configurations. Er kann entweder direkt in einer Erweiterung verwendet oder erweitert werden, um den Zugriff auf entsprechende Configurations zu steuern. Die genaue Art des Repositories der Configurations kann dabei beliebig sein, solange es einen Endpoint definiert, über den die entsprechend formatierten Configurations geladen werden können. Um Configurations aus einem Repository zu beziehen, dass nicht das entsprechende Format unterstützt, kann der Configuration Endpoints erweitert werden, um Configurations aus beliebigen Formaten zu erzeugen. Dadurch können bestehende Repositories genutzt werden, um Configurations zu erstellen.

6.6. Realisierung der Plugin-basierten Erweiterung

Um Erweiterungen des Quantum Workflow Modelers als Plugins zu integrieren, müssen die Erweiterungen durch ein Plugin-Objekt beschrieben und beim Plugin Handler registriert werden. Der Plugin Handler integriert dann die Erweiterungen, basierend auf den Attributen des Plugin-Objektes, in den Modeler.

Ein Plugin-Objekt einer Erweiterung besteht aus folgenden Attributen:

- **Name**: Eindeutiger Name des Plugins, der zur Identifikation des Plugins dient.
- **ExtensionModule**: Zusätzliche Module, die die Modellierungserweiterungen definieren und dem bpmn-js Modeler hinzugefügt werden. (optional)
- **ModdleDescription**: Beschreibung des Metamodells einer Erweiterung des bpmn-js Modelers als JSON Datei. (optional)
- **Buttons**: Liste an React Komponenten, die in die Toolbar des Editors integriert werden. (optional)
- **KonfigTabs**: Liset an Tabs der Erweiterung, die in den Konfigurationsdialog integriert werden. (optional)
- **TransformationsButton**: Button, über den eine Transformationsfunktion ausgeführt wird, die alle Modellierungserweiterungen durch Elemente, die in der Camunda BPMN Erweiterung enthalten sind, ersetzt. Dieser Button wird in den 'Transform Workflow'-Button des Editors integriert. (optional)

Über das *Buttons*- bzw. *KonfigTabs*-Attribut kann ein Plugin die Benutzeroberfläche des Editors erweitern. Die *ExtensionModule*- und *ModdleDescription*-Attribute definieren die Erweiterungen am bpmn-js Modeler, um beispielsweise neue quantenspezifische Elemente modellieren zu können.

Werden über das *ExtensionModule*- oder *ModddleDescription*-Attribut neue Modellierungselemente eingeführt, so muss das *TransformationsButton*-Attribut ebenfalls gesetzt werden, um einen Button mit einer Transformationsfunktion zu spezifizieren, die die eingeführten Modellierungselemente durch Elemente der Camunda BPMN Erweiterung ersetzt.

Jedes Plugin kann eine initiale Konfiguration definieren. Diese wird von der, den Modeler integrierenden, Web-Anwendung an den Plugin Handler übergeben. Der Plugin Handler speichert alle Konfigurationen, die über den Plugin-Namen abgerufen werden können. Die Struktur dieser Konfiguration bleibt dem Plugin überlassen und wird vom Plugin Handler nur gespeichert, nicht interpretiert. Somit kann ein Plugin in beliebigem Umfang konfiguriert werden. Die initiale Konfiguration, die dem Modeler übergeben wird, besteht aus einem Array, dessen Einträge JavaScript-Objekte sind. Diese verfügen über zwei Attribute:

- **Name**: Der obligatorische Name des Plugins, der zur eindeutigen Identifikation des Plugins dient.
- **Konfig**: Die Konfiguration des Plugins. Der Inhalt der Konfiguration kann vollständig vom Plugin definiert werden. (optional)

Alle Plugins, deren Name in der initialen Konfiguration enthalten ist, werden vom Plugin Handler im Modeler aktiviert. Die anderen Plugins bleiben deaktiviert und können nicht im Modeler verwendet werden. Deaktiviert bedeutet hier, dass sowohl die Erweiterungen des bpmn-js Modelers als auch eventuelle Erweiterungen der Benutzeroberfläche des Editors nicht in den Quantum Workflow Modeler integriert werden. Die Erweiterungen sind zur Laufzeit nicht im Modeler vorhanden.

Ein Plugin kann auf eine Reihe von Hilfsfunktionen zugreifen, um den bpmn-js Modeler und seine APIs leichter verwenden zu können. Sie erleichtern die Erweiterung des bpmn-js Modelers und vereinfachen die Entwicklung neuer Plugins. Der Quantum Workflow Modeler verfügt unter anderem über Hilfsfunktionen zur Modellierung von Workflow-Elementen, zur Erstellung und Erweiterung des Replace-Menüs, zum vereinfachten Rendering neuer Elemente zur Entwicklung einer Transformationsfunktion für die definierten Erweiterungen. Zusätzlich sind Hilfsfunktionen zum Verwalten des Workflow-Modells definiert, wie beispielsweise das Speichern oder Öffnen eines Workflows.

Im Folgenden werden die in dieser Arbeit entwickelten und in den Quantum Workflow Modeler integrierten Plugins aufgeführt.

6.6.1. Datenfluss-Plugin

Das Datenfluss-Plugin enthält die in Abschnitt 6.4 vorgestellte Datenflusserweiterung und integriert diese in den Quantum Workflow Modeler. Neben den Modellierungselementen enthält dieses Plugin Erweiterungen der Palette, des Properties Panel und des Replace-Menüs des bpmn-js Modelers, um die Verwendung der Datenflusselemente zu erleichtern. Diese Erweiterungen, sowie ihre Modellierungselemente, sind in Abbildung 6.9 abgebildet.

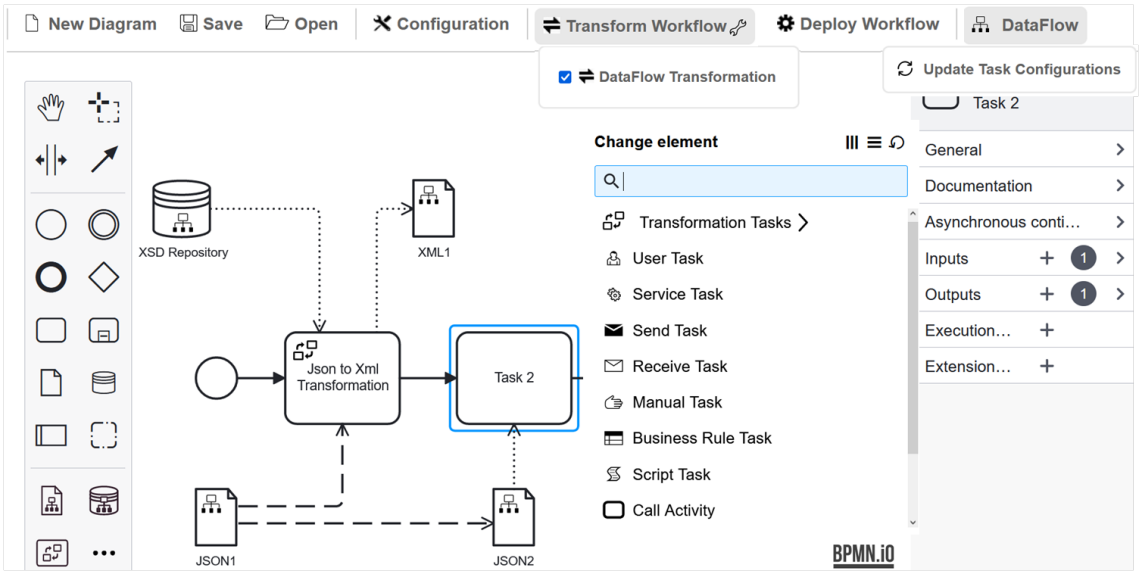

Abbildung 6.9.: Erweiterungen der Benutzeroberfläche und der Modellierungselemente des Datenfluss-Plugins.

Für `TransformationTasks` ist im Plugin ein Configurations Endpoint spezifiziert, über den konkrete Datentransformationsalgorithmen geladen und im Modeler ausgewählt werden können. Diese Configurations ermöglichen die Spezifikation von zusätzlichen Attributen und deren Belegung, ohne im Code einen neuen Typ definieren zu müssen. So können neue Transformationsalgorithmen zur Laufzeit in den Modeler geladen und verwendet werden.

Das Datenfluss-Plugin registriert die in Abschnitt 6.4.2 beschriebene Transformationsfunktion über seine Plugin-Definition beim Transformation Handler des Quantum Workflow Modelers. Dies ermöglicht die Transformation eines, mit Elementen der Datenflusserweiterung modellierten, Workflows in einen, mit der Camunda BPMN Erweiterung konformen, Workflow.

6.6.2. QuantME Plugin

Das QuantME Plugin enthält das QuantME Transformation Framework [WBLW20], das an die Plugin-Struktur und die Technologien des Quantum Workflow Modelers angepasst wurde. Dafür wurde die Implementierung überarbeitet und an den neuesten Stand des bpmn-js Modelers angepasst. Die Features des QuantME Transformation Frameworks sind in die Toolbar des Modelers integriert. Die in QuantME eingeführte Transformationsfunktion wurde entsprechend im Transformation Handler registriert, um mit Elementen des QuantME Plugins modellierte Workflows transformieren zu können. In Abbildung 6.10 ist ein Screenshot der QuantME Erweiterung abgebildet.

Zusätzlich wurden Configurations für `DataMapObjects` angelegt, um die in [WBLW20] beschriebenen Datenobjekte `Result Object` und `Quantum Circuit Object` modellieren zu können. Über die Configurations wurde eine graphische Notation definiert, um jedes Datenobjekt mit einem individuellen Icon graphisch darstellen zu können. Die graphische Notation der Datenobjekte ist in Abbildung 6.10 zu sehen.

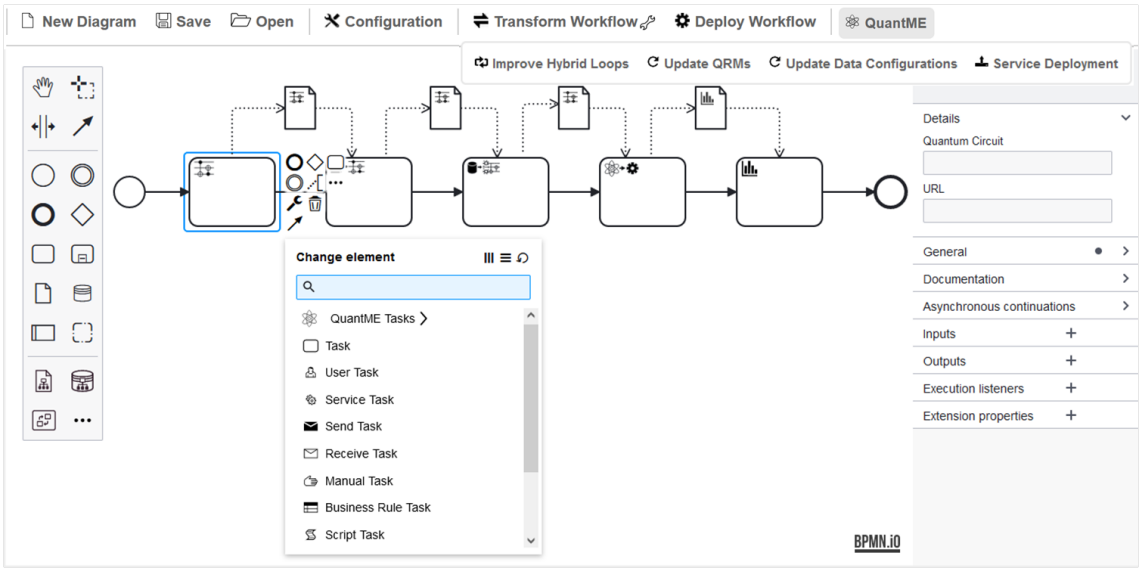

Abbildung 6.10.: Erweiterungen der Benutzeroberfläche und der Modellierungselemente des QuantME Plugins.

6.6.3. PlanQK Plugin

Die PlanQK Plattform [PQK23] steht für *Plattform und Ökosystem für Quantenapplikationen*. Sie bietet eine Plattform, auf der Wissen über Quantenalgorithmen ausgetauscht werden kann und mit der Quantenalgorithmen als Services provisioniert werden können. Die Quantenalgorithmen können als Services über die API der PlanQK Plattform abgerufen werden. Input-Daten und Output-Daten für die PlanQK Services können u.a. als sogenannte Data Pools zur Verfügung gestellt werden. Data Pools repräsentieren ein oder mehrere Dateien, die auf der PlanQK Plattform gespeichert werden. Das PlanQK Plugin erweitert den bpmn-js Modeler um die Möglichkeit, Services der PlanQK Plattform, sowie Data Pools der PlanQK Plattform als Datenquellen bzw. -senken zu modellieren. Dafür wurden neue Modellierungselemente eingeführt: `DataPools` und `PlanQKServiceTasks`.

Die `DataPool`-Klasse erbt von der `DataStoreMap`-Klasse. Dies ermöglicht die Verwendung des `Details`-Attributes um zusätzliche Informationen in `DataPools` zu speichern. Die `DataPool`-Klasse definiert die folgenden Attribute:

- **Name**: Name des Data Pools.
- **Link**: Link zur Definition bzw. ausführlichen Beschreibung des Data Pools und seiner Schnittstellen auf der PlanQK Plattform.
- **Beschreibung**: Kurze Beschreibung des Data Pools. Dient dem Modellierer als Unterstützung und Dokumentation.

`PlanQKServiceTasks` erben von der Task-Klasse und ermöglichen die Modellierung von Services der PlanQK Plattform, die zuvor auf die Plattform deployt bzw. auf der Plattform abonniert wurden. Services sind nach Anwendungen gruppiert, wobei eine Anwendung verschiedene Services

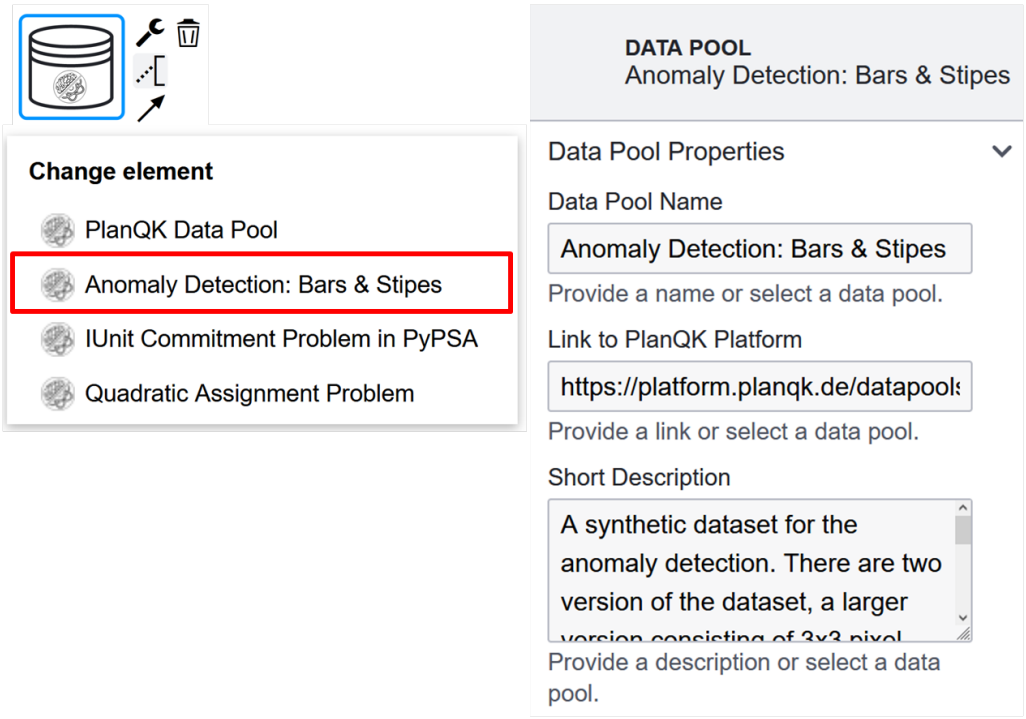

Abbildung 6.11.: Instanzen für einen PlanQK Data Pool.

abonnieren kann. Bevor ein Service modelliert werden kann, muss er zuerst von einer Anwendung abonniert werden. Die `PlanQKServiceTask`-Klasse definiert die folgenden Attribute, über die die Details des Services definiert werden können:

- **Application**: Anwendung, zu der der modellierte Service gehört bzw. die ihn abonniert hat.
- **Service**: Name des Services der PlanQK Plattform.
- **SubscriptionnId**: ID der Subscription, die die Anwendung auf den Service hat.
- **InputData**: String, der den Input des Services definiert. (optional)
- **Parameters**: Parameter des PlanQK Services.
- **Result**: Ergebnis der Ausführung des PlanQK Services. (optional)

Für `DataPools` und `PlanQKServiceTasks` werden dem Plugin über seine Konfiguration eine Menge an verfügbaren Instanzen der `DataPools` bzw. `PlanQKServiceTasks` übergeben, die der Modellierer dann über das Replace-Menü eines `DataPools` bzw. `PlanQKServiceTask` auswählen kann. Wurde ein solche Instanz ausgewählt, werden die Attribute des Data Pools bzw. Service Tasks entsprechend ausgefüllt, wie in Abbildung 6.11 zu sehen ist. Die Web-Anwendung, die den Modeler integriert, konfiguriert somit über die Plugin-Konfiguration welche Data Pools und Services der PlanQK Platform im Modeler verwendet werden können.

Das PlanQK Plugin definiert für seine Modellierungserweiterungen eine Transformationsfunktion im Transformation Handler. Durch diese Funktion werden alle `DataPools` durch DataStores und alle `PlanQKServiceTasks` durch ServiceTasks ersetzt. Die spezifischen Attribute des `PlanQKServiceTasks` werden dem ServiceTask, der ihn ersetzt, als Input-Variablen hinzugefügt. Da die `DataPool`-Klasse

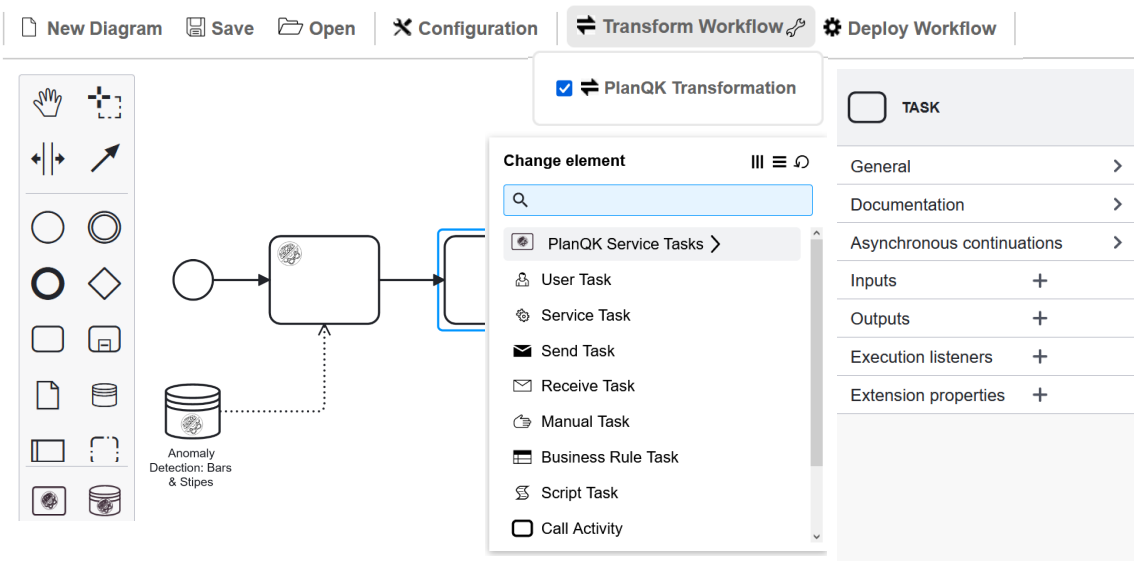

Abbildung 6.12.: Erweiterungen der Benutzeroberfläche und der Modellierungselemente des PlanQK Plugins.

von der `DataMapObject`-Klasse erbt, werden `DataPools` wie `DataMapObjects` transformiert, siehe Abschnitt 6.4.1 für weitere Details. Die spezifischen Attribute des `DataPools` werden vor der Transformation dem geerbten `Details`-Attribut des `DataPools` hinzugefügt.

Das PlanQK Plugin erweitert die Benutzeroberfläche des bpmn-js Modelers, um `DataPools` und `PlanQKServiceTasks` modellieren zu können. Die erweiterte Benutzeroberfläche ist in Abbildung 6.12 abgebildet.

6.6.4. QHAna Plugin

Das Quantum Humanities Data Analysis Tool [QHA23], kurz QHAna, bietet verschiedene Techniken des Machine Learnings, die sowohl auf klassischen als auch auf Quantencomputern ausgeführt werden können. Das QHAna Plugin Registry [QPR23] bietet diese Techniken als Services an und ermöglicht so die Ausführung der unterschiedlichen Techniken. Das QHAna Plugin ermöglicht die Orchestrierung der Services als Workflow. Dafür wird ein neues Modellierungselement, der `QHAnaServiceTask` eingeführt, der von der Task-Klasse erbt und diese um die folgenden Attribute erweitert:

- **Identifier**: Eindeutiger String der den Service identifiziert.
- **Name**: Name des Services als String.
- **Version**: Tag, der die konkrete Version des Services angibt.
- **Description**: Beschreibung des Services.

Die in einem QHAna Service spezifizierten Input-Daten werden dem `QHAnaServiceTask` als Input-Variablen hinzugefügt. Für die definierten Output-Daten wird ein `DataMapObject` angelegt, das über eine vom `QHAnaServiceTask` zum `DataMapObject` führende DataAssociation verbunden ist.

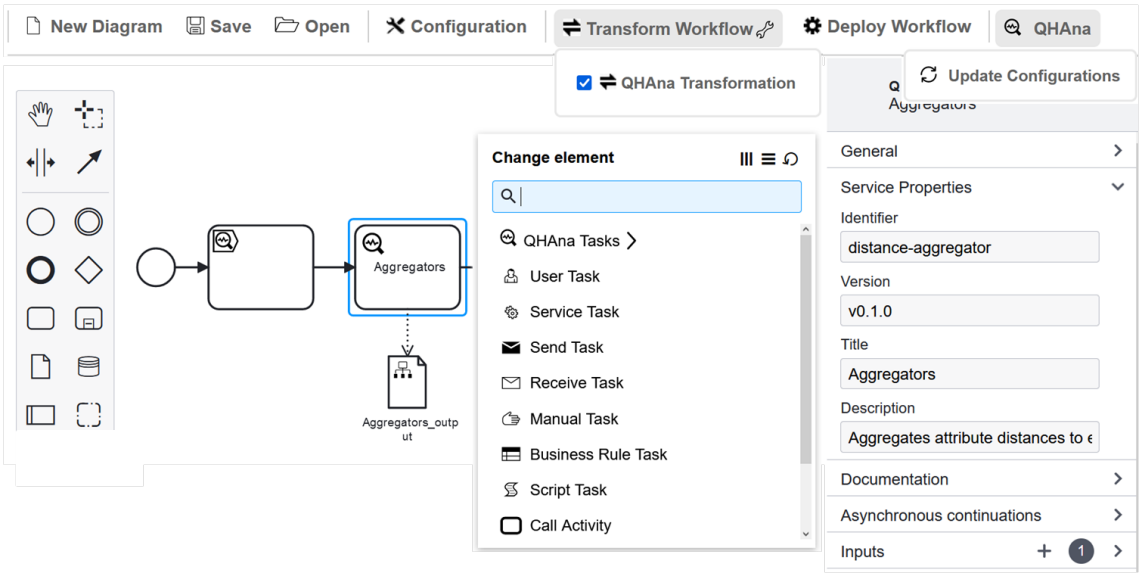

Abbildung 6.13.: Erweiterungen der Benutzeroberfläche und der Modellierungselemente des QHAna Plugins.

Dadurch kann der Modellierer auf einen Blick die erzeugten Daten eines Services erfassen und deren Aufbau und Inhalt dem `DataMapObject` entnehmen. Dadurch kann der Modellierer die Manipulation der Output-Daten eines Services durch die in Abschnitt 5.3 spezifizierten Datentransformationen modellieren.

Um konkrete QHAna Services zu modellieren, wurde im Plugin ein Configurations Endpoint entwickelt. Dieser lädt aus der QHAna Plugin Registry [QPR23] QHAna Services und wandelt diese dann in Configurations für den `QHAnaServiceTask` um, die der Modellierer über das Replace-Menü im Workflow-Modell verwenden kann (siehe Abbildung 6.13). Durch die Erweiterung der Editor Toolbar um den 'Update Configurations'-Button, können nach dem Laden der Website neue QHAna Services geladen werden. Um die URL zum QHAna Plugin Registry zur Laufzeit konfigurieren zu können, wurde dem Konfigurationsdialog des Editors ein neuer Tab hinzugefügt, über den die URL angepasst werden kann.

Um die schrittweise Ausführung der QHAna Services modellieren zu können, führt das Plugin den `QHAnaServiceStepTask` ein. Dieser erbt von der Task-Klasse und definiert ein `QHAnaNextStep`-Attribut, welches den als nächstes auszuführenden QHAna Service definiert.

In der vom QHAna Plugin spezifizierten Transformationsfunktion werden alle `QHAnaServiceTasks` und `QHAnaServiceStepTasks` durch ServiceTasks ersetzt, wobei Camunda konforme Attribute, wie Input-Variablen, in die ServiceTasks übernommen werden. Die zusätzlichen Attribute der Modellierungselemente werden den Input-Variablen der Service-Tasks hinzugefügt. Die durch `DataMapObjects` modellierten Output-Daten eines QHAna Services werden durch die im Datenfluss-Plugin definierte Transformationsfunktion ersetzt und müssen nicht vom QHAna Plugin transformiert werden.

7. Anwendungsbeispiel

In Abbildung 7.1 ist ein Anwendungsfall abgebildet, der als Workflow mit den durch die Plugins des Quantum Workflow Modelers eingeführten Modellierungselementen modelliert wurde. Er demonstriert, wie die Modellierungserweiterungen des Modelers verwendet werden können, um Quanten-Workflows zu modellieren. Der modellierte Workflow ist ein theoretisches Beispiel, dass in dieser Form momentan nicht ausgeführt werden kann. Er modelliert einen Prozess, der für einen Stadtplan, sowie eine gegebene Menge an Paketzustellern und ihrer Empfängeradressen die kürzeste Route für jeden Paketzusteller findet, die alle Empfängeradressen enthält. Dafür wird dies als Quadratic Unconstrained Binary Optimization (QUBO) Problem [KHG+14], formuliert und durch den Quantum Approximate Optimization Algorithm (QAOA) [FGG14] gelöst. Dafür wird das Problem als Optimierungsproblem betrachtet und durch wiederholte Ausführung eine Lösung des Problems, in diesem Fall die kürzeste Route, gesucht. Der Workflow enthält nicht den vollständigen Datenfluss, um die Übersichtlichkeit zu wahren. Stattdessen sind die für das Verständnis des Prozesses notwendigen Datenelemente modelliert. Der vollständige Workflow ist im GitHub Repository des Quantum Workflow Modelers zu finden [QUC23].

Der Stadtplan wird durch ein `DataMapObject` modelliert und zunächst in einen Graphen umgewandelt, modelliert durch einen `TransformationTask`. Dieser Graph wird anschließend in Subgraphen aufgeteilt, wobei ein Subgraph für jeden Paketzusteller generiert wird. Dieses Clustering wird durch den KMeans QHAna Service durchgeführt, modelliert durch einen `QHAnaServiceTask`, dem die Paketzusteller als `DataStoreMap` und der Stadtplangraph als Input dienen. Die entstehenden

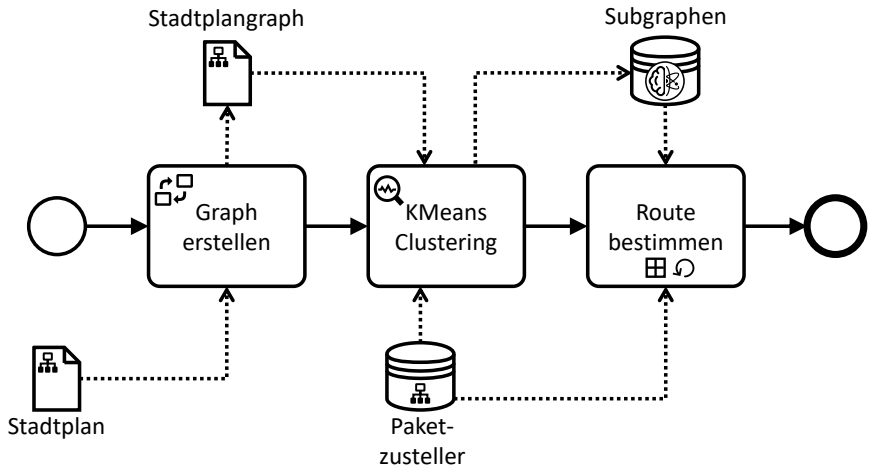

Abbildung 7.1.: Beispiel eines Anwendungsfalls, der mit dem Quantum Workflow Modeler und dessen Plugins modelliert wurde.

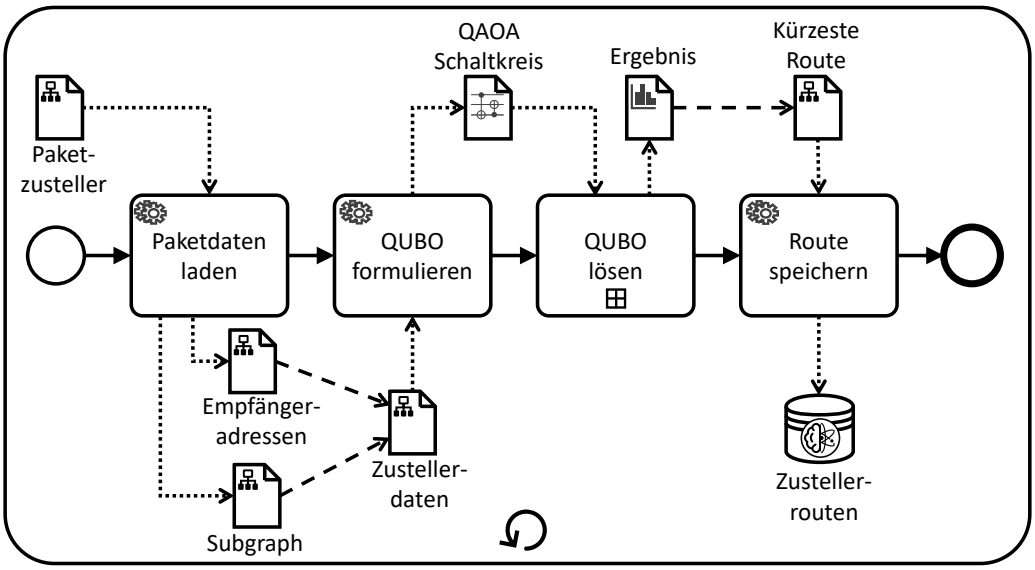

Abbildung 7.2.: SubProcess, der die Ausführung einer Schleife zur Bestimmung der kürzesten Route für einen gegebenen Paketzusteller modelliert.

Subgraphen werden in einem `DataPool` gespeichert. Die Bestimmung der kürzesten Route für jeden Paketzusteller wird durch den SubProcess *Route bestimmen* modelliert, der als Schleife für jeden Paketzusteller ausgeführt wird. Der SubProcess ist in Abbildung 7.2 abgebildet.

Um die kürzeste Route für jeden Paketzusteller zu bestimmen, werden zunächst alle benötigten Daten für den, als Input des SubProcess übergebenen, Paketzusteller geladen. Diese Informationen werden dann durch `TransformationAssociations` zu einem neuen `DataMapObject` *Zustellerdaten* zusammengefasst, das dem Task *QUBO formulieren* als Input dient. Dieser Task formuliert das QUBO Problem für den Paketzusteller und seine Empfängeradressen und generiert einen Quantenschaltkreis für QAOA. Dieser Schaltkreis wird als `QuantumCircuitObject` der QuantME Erweiterung modelliert und dient dem *QUBO lösen* SubProcess als Input.

Dieser SubProcess, dargestellt in Abbildung 7.3, löst das QUBO Problem durch die Ausführung des QAOA Optimierungsalgorithmus. Dies wird für jeden Paketzusteller ausgeführt. Dafür wird im SubProcess in einer Schleife zunächst der übergebene QAOA Schaltkreis durch einen `QuantumCircuitExecutionTask` der QuantME Erweiterung ausgeführt. Das Ergebnis wird durch ein QuantME `ResultObject` modelliert und durch einen QuantME `ReadoutErrorMitigationTask` nachbearbeitet, um den Fehler abzuschwächen. Anschließend wird im Task *Evaluiere Ergebnis* evaluiert, ob das Ergebnis eine Lösung für das QUBO Problem enthält, d.h. die kürzeste Route eines Paketzustellers. Ist dies der Fall, wird der SubProcess beendet. Ist dies nicht der Fall, werden die Optimierungsparameter des QAOA Algorithmus angepasst und die Schritte des SubProcesses werden wiederholt, wobei der Inhalt des `ResultObjects` in jeder Ausführung überschrieben wird, sodass das endgültige Ergebnis des SubProcesses in diesem `ResultObject` gespeichert wird. Dieses Ergebnis wird dem SubProcess *Route bestimmen* übergeben. Aus dem Ergebnis wird durch eine `TransformationAssociation` die kürzeste Route extrahiert und im `DataMapObject` *Kürzeste Route* gespeichert. Durch den letzten Task wird diese Route in einem `DataPool` auf der PlanQK Plattform gespeichert.

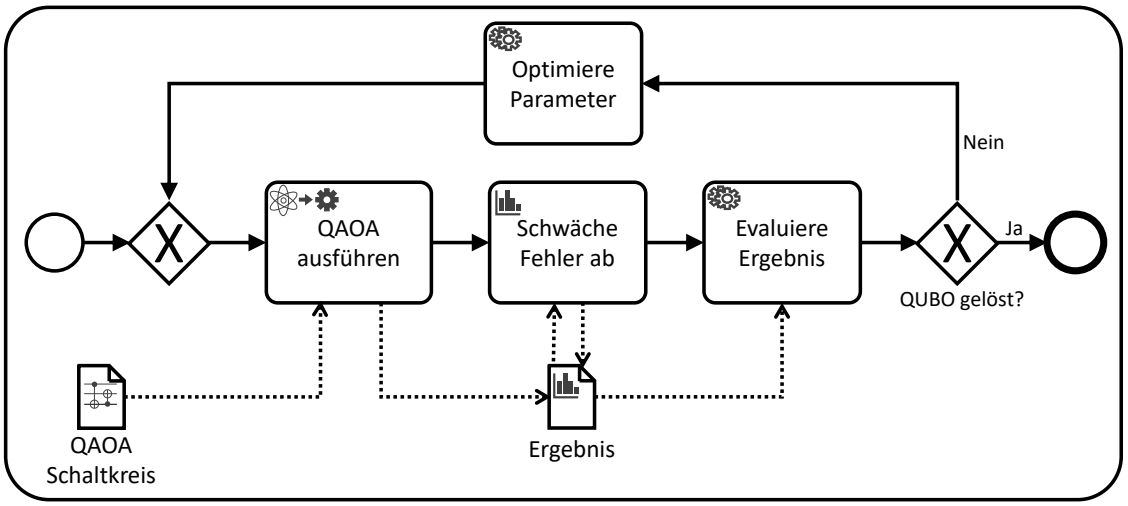

Abbildung 7.3.: SubProcess, der die Ausführung der Schleife zur Lösung des QUBOs durch Ausführung von QAOA modelliert.

8. Zusammenfassung und Ausblick

In dieser Arbeit wurde der Quantum Workflow Modeler, ein Modellierungswerkzeug zur graphischen Modellierung von Quantum Workflows inklusive ihres Datenflusses und Datentransformation, vorgestellt. Der Quantum Workflow Modeler ermöglicht die Plugin-basierte Integration von Modellierungserweiterungen, um Elemente aus der Quantendomäne modellieren zu können. Um die Modellierung zu erleichtern und die Wiederverwendbarkeit zu erhöhen wurden Configurations entwickelt, um konkrete Instanzen von Modellierungselementen zur Laufzeit in den Modeler laden zu können, ohne dass Anpassungen im Code vorgenommen werden müssen. Um den Datenfluss und Datentransformationen eines Workflows modellieren zu können, wurde die Datenflusserweiterung eingeführt. Sie ermöglicht die Modellierung eines Datenflusses, der in der Ausführung des Workflows in der Workflow Engine berücksichtigt wird. Datentransformationen werden in der Erweiterung direkt im Datenfluss oder über einen separaten Task modelliert, abhängig von der Komplexität der Transformation. Um die Portabilität der mit den Erweiterungen des Quantum Workflow Modelers modellierten Workflow-Modelle zu gewährleisten, bietet der Modeler eine Schnittstelle an, über die Plugins eine Transformationsfunktion spezifizieren können. Diese Funktion ermöglicht es, Workflow-Modelle, die mit Erweiterungen modelliert wurden, in BPMN konforme Workflow-Modelle zu transformieren.

Diese Konzepte wurden in einer JavaScript Single-Page Web-Anwendung realisiert, die als benutzerdefinierte HTML Komponente in andere Web-Anwendungen integriert werden kann. Der Quantum Workflow Modeler basiert auf dem bpmn-js Modeler, der die Modellierung von BPMN Workflows ermöglicht. Dieser Modeler wird durch die in dieser Arbeit eingeführten Plugins zur Modellierung des Datenflusses und Quantenanwendungen, erweitert. Das QuantME Plugin ermöglicht die Modellierung von Quantenalgorithmen einschließlich Pre- und Post-Processing Tasks. Durch das PlanQK Plugin können Elemente der PlanQK Plattform in Workflows modelliert werden. Über das QHAna Plugin können Services der QHAna Plugin Registry im Modeler orchestriert werden. Die im Quantum Workflow Modeler realisierten Konzepte erlauben die Integration von weiteren Plugins, um auch zukünftige Entwicklungen im Quanten-Computing im Modeler abbilden zu können.

Erweiterungen des Modelers sollten Erweiterungen nur für neue Typen definieren. Änderungen an nativen BPMN Elementen können ungewollte Nebeneffekte in anderen Erweiterungen auslösen, die von dem im BPMN Standard spezifizierten Verhalten der Elemente ausgehen. Erweiterungen des Modelers, die native BPMN Elemente verändern werden nicht unterstützt und sind lediglich durch einen benutzerdefinierten Typ möglich. Die Verwendung von Collections, wie sie im BPMN Standard definiert sind, werden von `DataMapObjects` nicht unterstützt. Stattdessen kann dieses Verhalten durch entsprechende Key-Value-Paare, die dem `Content`-Attribut hinzugefügt werden modelliert werden. Der Wert eines Elementes der Collection kann als String codiert als Wert des Paares gespeichert werden. Die Verwendung von `DataMapObjects` und den parallelen, sequentiellen und Schleifenmarkern ist im Modeler nur eingeschränkt möglich, da Camunda die Outputs-Attribute

von Activities mit diesen Markern entfernt und somit die Transformation der `DataMapObjects` nicht mehr gewährleistet werden kann. Daher können `DataMapObjects` lediglich zur Modellierung der Inputs von Activities mit diesen Markern verwendet werden.

Weiterführenden Arbeiten können sich mit der Frage beschäftigen, wie die in einem Task definierten Inputs graphisch dargestellt werden können, sodass der Modellierer direkt im Workflow sieht, welche Input-Daten der Task benötigt und diese dann über `DataMapObjects` und DataAssociations entsprechend modellieren kann. Momentan wird diese Information den Attributen des Tasks entnommen. Ein weiteres offenes Problem ist die graphische Modellierung der Struktur der modellierten Daten, sodass im Workflow-Modell die Struktur bzw. der Typ des modellierten Inputs und des vom Task benötigten Inputs graphisch modelliert werden können.

Literaturverzeichnis

[BBMC20] B. Bauer, S. Bravyi, M. Motta, G. K.-L. Chan. „Quantum algorithms for quantum chemistry and quantum materials science“. In: *Chemical Reviews* 120.22 (2020), S. 12685–12717 (zitiert auf S. 11).

[BHK+15] U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, M. Wieland. „A situation-aware workflow modelling extension“. In: *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*. 2015, S. 1–7 (zitiert auf S. 23).

[BPMNIO23] Camunda Service GmbH. *Web-based tooling for BPMN, DMN and Forms.* 2023. URL: https://bpmn.io/ (zitiert auf S. 39).

[BPMNJS23] Camunda Service GmbH. *bpmn-js - BPMN 2.0 for the web*. 2023. URL: https://github.com/bpmn-io/bpmn-js/ (zitiert auf S. 39, 45).

[BPMNJSDOC23] Camunda Service GmbH. *bpmn-js - Walkthrough*. 2023. URL: https://bpmn.io/toolkit/bpmn-js/walkthrough/ (zitiert auf S. 39).

[CAM23] Camunda Service GmbH. *Camunda Modeler: Design von Geschäftsprozessen und Entscheidungsmodellen*. 2023. URL: https://camunda.com/de/platform/modeler/ (zitiert auf S. 39).

[CAMP723] Camunda Services GmbH. *The Camunda Platform 7 Manual*. 2023. URL: https://docs.camunda.org/manual/latest/ (zitiert auf S. 11, 17, 39, 45–48).

[CAMP823] Camunda Services GmbH. *Camunda Platform 8 Docs*. 2023. URL: https://docs.camunda.io/docs/reference/ (zitiert auf S. 25, 39, 45, 50, 51).

[DKB08] G. Decker, O. Kopp, A. Barros. „An Introduction to Service Choreographies (Servicechoreographien–eine Einführung)“. In: *it-Information Technology* 50.2 (2008), S. 122–127 (zitiert auf S. 18).

[Ell99] C. A. Ellis. „Workflow technology“. In: *Computer Supported Cooperative Work, Trends in Software Series* 7 (1999), S. 29–54 (zitiert auf S. 13).

[FAG23] Flowable AG. *Flowable Open Source Documentation*. 2023. URL: https://www.flowable.com/open-source/docs/bpmn/ch02-GettingStarted (zitiert auf S. 18).

[FGG14] E. Farhi, J. Goldstone, S. Gutmann. „A quantum approximate optimization algorithm“. In: *arXiv preprint arXiv:1411.4028* (2014) (zitiert auf S. 61).

[FHBL19] G. Falazi, M. Hahn, U. Breitenbücher, F. Leymann. „Modeling and execution of blockchain-aware business processes“. In: *SICS Software-Intensive Cyber-Physical Systems* 34 (2019), S. 105–116 (zitiert auf S. 23).

[FUN22] D. Funk. *Data Objects in SpiffWorkflow*. 2022. URL: https://www.spiffworkflow.org/posts/deep_dives/data_objects/ (zitiert auf S. 18).

[HBKL18] M. Hahn, U. Breitenbücher, O. Kopp, F. Leymann. „Modeling and execution of data-aware choreographies: an overview“. In: *Computer Science-Research and Development* 33 (2018), S. 329–340 (zitiert auf S. 18, 34, 35).

[HBL+18] M. Hahn, U. Breitenbücher, F. Leymann, M. Wurster, V. Yussupov. „Modeling data transformations in data-aware service choreographies“. In: *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE. 2018, S. 28–34 (zitiert auf S. 18).

[HBLW17] M. Hahn, U. Breitenbücher, F. Leymann, A. Weiß. „TraDE-a transparent data exchange middleware for service choreographies“. In: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I*. Springer. 2017, S. 252–270 (zitiert auf S. 18).

[HW11] P. Harmon, C. Wolf. „Business process modeling survey“. In: *Business process trends* 36.1 (2011), S. 1–36 (zitiert auf S. 13, 14).

[JS23] Mozilla Developer Network. *JavaScript*. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript?retiredLocale=de (zitiert auf S. 39).

[JUEL14] Odysseus Software GmbH. *Java Unified Expression Language*. 2014. URL: https://juel.sourceforge.net/ (zitiert auf S. 45).

[KHG+14] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, Y. Wang. „The unconstrained binary quadratic programming problem: a survey“. In: *Journal of combinatorial optimization* 28 (2014), S. 58–81 (zitiert auf S. 61).

[LBF+20] F. Leymann, J. Barzen, M. Falkenthal, D. Vietz, B. Weder, K. Wild. *Quantum in the Cloud: Application Potentials and Research Opportunities*. 2020. arXiv: 2003.06256 [quant-ph] (zitiert auf S. 11).

[LR99] F. Leymann, D. Roller. *Production workflow: concepts and techniques*. Prentice Hall PTR, 1999 (zitiert auf S. 11, 13).

[META23] Meta Open Source. *React - The library for web and native user interfaces*. 2023. URL: https://react.dev/ (zitiert auf S. 39).

[NIA22] N. Deehan. *Camunda Platform 8 for Camunda Platform 7 Users – What You Need to Know*. 2022. URL: https://camunda.com/blog/2022/04/camunda-platform-8-for-camunda-platform-7-users-what-you-need-to-know/ (zitiert auf S. 40).

[OMG11] Object Management Group. *Business Process Model and Notation (BPMN) - Version 2.0*. 2011. URL: https://www.omg.org/spec/BPMN/2.0 (zitiert auf S. 3, 14).

[ORQ23] Zapata. *Orquestra*. 2023. URL: https://www.orquestra.io/ (zitiert auf S. 17).

[PQK23] Consortium PlanQK. *PlanQK Plattform*. 2023. URL: https://platform.planqk.de/ (zitiert auf S. 12, 56).

[QHA23] *QHAna - Quantum Humanities Analysis Tool*. 2023. URL: https://github.com/UST-QuAntiL/qhana (zitiert auf S. 58).

[QME23] UST-QuAntiL Group. *QuantME Modeling and Transformation Framework*. 2023. URL: https://github.com/UST-QuAntiL/QuantME-TransformationFramework (zitiert auf S. 17).

[QPR23] F. Bühler. *The QHAna Plugin-Registry*. 2023. URL: https://qhana-plugin-registry.readthedocs.io/en/latest/readme.html# (zitiert auf S. 58, 59).

[QUC23] F. Belz. *Quantum Workflow Modeler Use Case*. 2023. URL: https://github.com/PlanQK/workflow-modeler/tree/master/doc/use-case (zitiert auf S. 61).

[QWM23] F. Belz. *Quantum Workflow Modeler*. 2023. URL: https://github.com/PlanQK/workflow-modeler (zitiert auf S. 39).

[RSM+20] S. B. Ramezani, A. Sommers, H. K. Manchukonda, S. Rahimi, A. Amirlatifi. „Machine learning algorithms in quantum computing: A survey“. In: *2020 International joint conference on neural networks (IJCNN)*. IEEE. 2020, S. 1–8 (zitiert auf S. 11).

[SPI23] Sartography. *SpiffWorkflow Dokumentation*. 2023. URL: https://spiffworkflow.readthedocs.io/en/latest/intro.html (zitiert auf S. 11, 18).

[TVS09] N. Trčka, W. M. Van der Aalst, N. Sidorova. „Data-flow anti-patterns: Discovering data-flow errors in workflows“. In: *Advanced Information Systems Engineering: 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009. Proceedings 21*. Springer. 2009, S. 425–439 (zitiert auf S. 18).

[UEL10] Oracle. *Unified Expression Language*. 2010. URL: https://docs.oracle.com/javaee/5/tutorial/doc/bnahq.html (zitiert auf S. 32).

[VBLW22] D. Vietz, J. Barzen, F. Leymann, B. Weder. „Splitting Quantum-Classical Scripts for the Generation of Quantum Workflows“. In: *Enterprise Design, Operations, and Computing*. Hrsg. von J. P. A. Almeida, D. Karastoyanova, G. Guizzardi, M. Montali, F. M. Maggi, C. M. Fonseca. Cham: Springer International Publishing, 2022, S. 255–270. ISBN: 978-3-031-17604-3 (zitiert auf S. 17).

[WBL21] B. Weder, J. Barzen, F. Leymann. „MODULO: modeling, transformation, and deployment of quantum workflows“. In: *2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE. 2021, S. 341–344 (zitiert auf S. 17).

[WBLS21] B. Weder, J. Barzen, F. Leymann, M. Salm. „Automated Quantum Hardware Selection for Quantum Workflows“. In: *Electronics* 10.8 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10080984. URL: https://www.mdpi.com/2079-9292/10/8/984 (zitiert auf S. 17).

[WBLW20] B. Weder, U. Breitenbücher, F. Leymann, K. Wild. „Integrating quantum computing into workflow modeling and execution“. In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE. 2020, S. 279–291 (zitiert auf S. 11, 17, 19, 21, 23, 24, 41, 55).

[WBLZ21] B. Weder, J. Barzen, F. Leymann, M. Zimmermann. „Hybrid quantum applications need two orchestrations in superposition: a software architecture perspective“. In: *2021 IEEE International Conference on Web Services (ICWS)*. IEEE. 2021, S. 1–13 (zitiert auf S. 11).

[ZAP23] Zapata. *Zapata - Quantum AI Software for Enterprise*. 2023. URL: https://www.zapatacomputing.com/ (zitiert auf S. 17).

Alle URLs wurden zuletzt am 04. 05. 2023 geprüft.

A. XSD-Schema der Datenflusserweiterung

In Listing A.1 ist das vollständige XSD-Schema der in Abschnitt 5.3 eingeführten Erweiterung des Datenflusses eines BPMN Workflows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
        xmlns="http://extension.org/dataflow"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://qflow.org/dataflow"
        xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">

    <xsd:import namespace="http://www.omg.org/spec/BPMN/20100524/MODEL"
schemaLocation="Semantic.xsd"/>

    <xsd:annotation>
        <xsd:documentation>
            This XML Schema defines and documents BPMN 2.0 extension elements and
            attributes introduced to support the data flow extension.
        </xsd:documentation>
    </xsd:annotation>

    <xsd:element name="dataMapObject" type="tDataMapObject" substitutionGroup="bpmn:flowElement"/>
    <xsd:complexType name="tDataMapObject">
        <xsd:annotation>
            <xsd:documentation>
                Data object type to define a set of data.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="bpmn:tDataObject">
                <xsd:attribute name="content" type="tKeyValueMap" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="dataStoreMap" type="tDataStoreMap" substitutionGroup="bpmn:flowElement"/>
    <xsd:complexType name="tDataStoreMap">
        <xsd:annotation>
            <xsd:documentation>
                Data store type to define a persistent data storage.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="bpmn:tDataStore">
                <xsd:attribute name="details" type="tKeyValueMap" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
```

```
45      </xsd:complexType>
46
47      <xsd:element name="transformationAssociation" type="tTransformationAssociation"
    substitutionGroup="bpmn:baseElement"/>
48      <xsd:complexType name="tTransformationAssociation">
49          <xsd:annotation>
50              <xsd:documentation>
51                  DataAssociation type to define a transformation of DataMapObjects.
52              </xsd:documentation>
53          </xsd:annotation>
54          <xsd:complexContent>
55              <xsd:extension base="bpmn:tDataAssociation">
56                  <xsd:attribute name="expressions" type="tKeyValueMap" use="required"/>
57              </xsd:extension>
58          </xsd:complexContent>
59      </xsd:complexType>
60
61      <xsd:element name="transformationTask" type="tTransformationTask"
    substitutionGroup="bpmn:flowElement"/>
62      <xsd:complexType name="tTransformationTask">
63          <xsd:annotation>
64              <xsd:documentation>
65                  Task type to define a complex transformation of data.
66              </xsd:documentation>
67          </xsd:annotation>
68          <xsd:complexContent>
69              <xsd:extension base="bpmn:tTask">
70                  <xsd:attribute name="parameters" type="tKeyValueMap" use="required"/>
71              </xsd:extension>
72          </xsd:complexContent>
73      </xsd:complexType>
74
75      <xsd:complexType name="tKeyValueMap">
76          <xsd:sequence>
77              <xsd:element name="keyValuePair" type="tKeyValuePair" minOccurs="0" />
78          </xsd:sequence>
79      </xsd:complexType>
80
81      <xsd:complexType name="tKeyValuePair">
82          <xsd:sequence>
83              <xsd:element name="key" type="xsd:string" maxOccurs="1" minOccurs="1" />
84              <xsd:element name="value" type="xsd:string" maxOccurs="1" minOccurs="1" />
85          </xsd:sequence>
86      </xsd:complexType>
87  </xsd:schema>
```

Listing A.1: XSD-Schema der in Abschnitt 5.3 beschriebenen Erweiterung des BPMN Datenflusses.

Erklärung

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

Ort, Datum, Unterschrift