Programmierkonzepte für die Umsetzung von Nutzungsrichtlinien in industriellen Datenräumen

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r I n g e n i e u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Fabian Bruckner

Dortmund

2022

Tag der mündlichen Prüfung: 11.08.2022
Dekan: Prof. Dr.-Ing. Gernot A. Fink
Gutachter: Prof. Dr. Falk Howar & Prof. Dr. Jan Jürjens

Kurzfassung

Fabian Bruckner

Programmierkonzepte für policy-agnostische Datenverarbeitung in industriellen Datenräumen

Daten haben sich im Laufe der Zeit immer mehr zu einem wertvollem Asset entwickelt. Aus diesem Grund ist für Rechteinhaber die Kontrolle über die eigenen Daten von zentraler Bedeutung. Die Fähigkeit des Rechteinhabers selbstbestimmt über die Nutzung seiner Daten zu verfügen wird als Datensouveränität bezeichnet. Diese Arbeit beschäftigt sich mit der Frage, wie die Erlangung sowie der Erhalt der Datensouveränität technisch durch Usage Control Mechanismen unterstützt werden kann.

In der vorliegenden Arbeit wird eine flexible und erweiterbare Programmiersprache entwickelt, welche über integrierte Usage Control Mechanismen verfügt und den Namen D° trägt. Durch die Umsetzung des Programmierparadigmas der policy-agnostischen Programmierung wird die Komplexität der Usage Control Mechanismen gekapselt und kann durch Experten adressiert werden. Ein Teil dieser Komplexität ist in den Compiler verlagert und gelöst worden und muss von Anwendern der Sprache nicht mehr beachtet werden. Hierdurch wird der Applikationsentwickler entlastet und die korrekte Nutzung von Usage Control Mechanismen vereinfacht.

Des Weiteren wird präsentiert, wie das Remote Evaluation Paradigma für D° umgesetzt werden kann. Das Paradigma zielt auf Szenarien der kooperativen Datennutzung ab und verzichtet auf den Versand von Daten an Dritte, welche die Daten verwenden möchten. Stattdessen werden die datenverarbeitenden Applikationen und deren Berechnungsergebnisse hin- und hergeschickt. Hierdurch verbleiben die Daten stets auf den Systemen des Rechteinhabers, welche gleichzeitig auf die Vorteile der Usage Control Mechanismen in D° zurückgreifen können. Dies erlaubt die kooperative Datennutzung in Szenarien, in denen die Weitergabe von Daten ausgeschlossen ist und technische Maßnahmen zur Datennutzungskontrolle notwendig sind.

Die erzielten Ergebnisse werden mithilfe eines größeren Demonstrators präsentiert und validiert. Dabei werden die einzelnen Aspekte von D° anhand von Beispielen praktisch vorgestellt. Außerdem findet eine Einordnung der Lösung in die International Data Spaces statt, welche die vorliegende Arbeit maßgeblich motiviert und geprägt haben. Bei dieser Einordnung wird gezeigt, dass die Mächtigkeit der Usage Control Mechanismen von D° gleich oder besser zu der von anderen Usage Control Mechanismen, welche in den International Data Spaces verwendet werden, ist.

Schlagwörter: Policy-agnostische Programmierung, Programmiersprachen, Codegenerierung, Datennutzungskontrolle, Remote Evaluation, Mobile Code

Abstract

Fabian Bruckner

Programming Concepts for Policy-Agnostic Data Processing in Industrial Dataspaces

Nowadays, data has evolved into a valuable asset. For this reason, control over one's own data is of central importance for data owners. The ability of the rights holder to determine the use of his or her data in a self-determined manner is referred to as data sovereignty. This thesis deals with the question of how the establishment and maintenance of data sovereignty can be technically supported by usage control mechanisms.

In this thesis, a flexible and extensible programming language is developed which has built-in usage control mechanisms and is named D°. By implementing the programming paradigm of policy-agnostic programming, the complexity of the usage control mechanisms is encapsulated and can be addressed by experts. Part of this complexity has been moved and solved in the compiler and no longer needs to be addressed by users of the language. This relieves the application developer and simplifies the correct use of usage control mechanisms.

Furthermore, an implementation of the remote evaluation paradigm for D° is presented. The paradigm targets scenarios of cooperative data usage and omits the need to send data to third parties who want to use the data. Instead, the data-processing applications and their computation results are sent back and forth. As a result, the data always remains on the rights holder's systems, which at the same time can take advantage of the usage control mechanisms in D°. This allows cooperative data usage in scenarios where data sharing is prohibited and usage control mechanisms are required.

The results obtained are presented and validated by using a larger showcase. The individual aspects of D° are presented in practice using examples. In addition, the solution is introduced into the International Data Spaces, which have significantly motivated and shaped the presented work. In this context, it is shown that the power of the usage control mechanisms of D° is equal to or better than that of other usage control mechanisms used in the International Data Spaces.

Key words: policy-agnostic programming, programming languages, code generation, usage control, remote evaluation, mobile code

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Für Rechteinhaber ist die Kontrolle über die eigenen Daten von zentraler Bedeutung. Dies bezieht sich nicht nur auf den Schutz der gespeicherten Daten, sondern umfasst auch die Kontrolle darüber, wer Zugriff auf die Daten erhält und welche Aktionen mit den Daten durchgeführt werden dürfen. In diesem Kontext wird häufig über die sogenannte Datensouveränität gesprochen.

> „Data sovereignty refers to the self-determination of individuals and organizations with regard to the use of their data."
> — Jarke, Matthias and Otto, Boris and Ram, Sudha [Jar19, S. 550]

Dabei bezieht sich die Selbstbestimmung auf die zuvor genannte Kontrolle der Rechteinhaber darüber, wie und durch wen die eigenen Daten verwendet werden. Die Erlangung bzw. Wahrung dieser Datensouveränität ist heutzutage von höchster Relevanz für eine Vielzahl von Unternehmen und Personen [Bad19]. Ein Grund hierfür ist, dass immer mehr Unternehmen datengetriebene Geschäftsmodelle verfolgen und zum Teil vollständig abhängig von diesen sind [Zol16]. Die Daten dieser Unternehmen sind ihr wertvollstes Asset und bedürfen eines entsprechenden Schutzes, da ein Kontrollverlust über dieses Asset katastrophale Folgen für das Unternehmen hätte. Dabei bezieht sich der Schutz nicht ausschließlich auf Maßnahmen gegen Angriffe auf Systeme und Transportkanäle. Ein ebenso wichtiger Aspekt ist die Kontrolle darüber, wie die eigenen Daten genutzt werden.

Auch abseits datengetriebener Geschäftsmodelle und der Unternehmen, die auf diese setzen, ist das Thema der Datensouveränität von zentraler Bedeutung. Im Kontext der Industrie 4.0 werden Maschinen, welche von produzierenden Unternehmen eingesetzt werden, zunehmend mit Sensorik ausgestattet, welche große Mengen an Daten erfassen, wodurch Cyper-physische Produktionssysteme entstehen [Mon14]. Dabei ist das Ziel die Verarbeitung dieser Daten, um neue, nützliche Informationen zu generieren, welche einen Mehrwert für die Unternehmen erzeugen [Lee14]. Anwendungsfälle, die in diesem Zusammenhang häufig verwendet werden, sind die prädiktive Wartung [Wan16; Yan17; Zon20] und das smarte Lieferketten Management [Pas20; Sha21; Tja17].

Des Weiteren ist auch für Privatpersonen die Erlangung und Wahrung der Datensouveränität ein wichtiges Ziel. Für jede Person existieren (besondere) personenbezogene Daten. Dabei erzeugen Personen unter Umständen nicht unerhebliche Mengen dieser Daten. Dies geschieht beispielsweise durch die Nutzung von Webangeboten wie Google und Facebook. Sowohl für die betroffenen Personen als auch Unternehmen, welche diese Daten verwenden möchten, haben diese Daten einen Wert [Li15]. Da diese Datensätze häufig sensible Daten

enthalten [Fun10], besteht für die Personen ein berechtigtes Interesse den Umgang mit diesen Daten kontrollieren zu können.

Dass die Datensouveränität heutzutage ein zentrales Thema ist, zeigt sich auch an der Präsenz des Themas auf politischer Ebene. Auf dem Digitalgipfel der Bundesregierung 2019 war Datensouveränität ein zentrales Thema [Bun].

> „Wir haben die Soziale Marktwirtschaft, die mit ihren Prinzipien und Werten erhalten bleiben und zugleich in die digitale Welt transformiert werden muss. Deshalb spielt das Wort Datensouveränität eine so wichtige Rolle für uns. Der Umgang mit Daten, die Selbstbestimmtheit der Bürgerinnen und Bürger und das Wissen der Bürgerinnen und Bürger sowie der Unternehmen um die Fragen ‚Wo sind meine Daten?', ‚Wer arbeitet mit meinen Daten?' und ‚Habe ich dazu das Einverständnis erklärt?' – das sollte ein Grundthema sein."
> — Dr. Angela Merkel, Bundeskanzlerin [Mer]

Im Jahr 2020 wurde von der europäischen Kommission eine europäische Datenstrategie veröffentlicht. Auch diese adressiert unter anderem Aspekte, welche Teil der Datensouveränität sind. So ist eine der Aussagen der europäischen Datenstrategie, dass „die Art und Weise wie Daten gesammelt und verwendet werden, zuallererst den Interessen des Einzelnen entsprechen" [Kom] müssen.

Während die Wahrung der eigenen Datensouveränität bereits in einer geschlossenen Umgebung, wie beispielsweise einem Unternehmen, eine hohe Komplexität aufweisen kann, sind reale Szenarien heutzutage weitaus komplexer. Eine Vielzahl von Unternehmen setzt inzwischen Cloud-Umgebungen ein [Bit20; Eur14]. Darunter befinden sich auch Unternehmen, welche keine eigene IT-Infrastruktur betreiben und vollständig auf Cloud-Plattformen setzen. Hiervon sind insbesondere kleine und mittelständische Unternehmen betroffen [Che12]. Das bedeutet, dass zumindest Teile der Unternehmensdaten mindestens zeitweise auf Systemen verarbeitet bzw. gespeichert werden, über die das Unternehmen keine vollständige Kontrolle hat. Diese Unternehmen sind darauf angewiesen, dass die Betreiber der jeweiligen Cloud-Plattform vertrauenswürdig sind und die abgelegten Daten nicht missbrauchen.

Darüber hinaus ist das Teilen von Daten zwischen Unternehmen ein notwendiger Bestandteil diverser Geschäftsprozesse, beispielsweise der effektiven Verwaltung von Lieferketten [Ste02]. Gleichzeitig werden durch das Teilen von Daten neue Möglichkeiten der Wertschöpfung erschlossen. Dies betrifft unter anderem datengetriebene Geschäftsmodelle [Eck14]. Beispielsweise können mehr Informationen aus Daten gewonnen werden, wenn die verfügbare Datenmenge (bei gleicher Qualität) größer ist [Ric19]. In diesen Situationen werden Daten bewusst an Dritte übertragen und auf Systemen verarbeitet bzw. gespeichert, über die der Rechteinhaber keinerlei Kontrolle hat. Dabei ist es für den Rechteinhaber von hoher Relevanz, dass die Parteien, mit denen die Daten geteilt werden, diese sachgemäß und vertrauensvoll verwenden.

Eine Gemeinsamkeit in all den beschriebenen Szenarien ist, dass die Daten einen intrinsischen Wert besitzen und dass ein potentieller Kontrollverlust über die Daten droht, welcher zu erheblichen Schäden führen kann. Ein unzureichender Schutz der Kontrolle über

Daten kann somit schädliche Auswirkungen für den Rechteinhaber bewirken. Beispielsweise könnten Geschäftsgeheimnisse veröffentlicht oder die eigene Verhandlungsposition geschwächt werden. Daraus wird deutlich, dass Mechanismen notwendig sind, welche es ermöglichen feingranular Rechte an den eigenen Daten zu gewähren und sicherzustellen, dass die definierten Restriktionen eingehalten werden.

Mit der zunehmenden Betrachtung von Daten als wertvolles Wirtschaftsgut entwickelten sich immer mehr organisatorische Prozesse, welche einen verantwortungsvollen Umgang mit den Daten gewährleisten sollen [Glu18]. Diese Maßnahmen haben eine zentrale Rolle bei der Erlangung und Wahrung der Datensouveränität. Die Definition von Zuständigkeiten, Verantwortlichkeiten und feingranularen Zugriffsrechten verhindert einen unkontrollierten Datenzugriff und die Entstehung eines Datensumpfs [Pas15]. Hierdurch wird eine qualitativ hochwertige Datenbasis geschaffen, welche mit klaren Verantwortlichkeiten und Rechten ausgestattet ist. Werden auf dieser Datenbasis Richtlinien definiert, welche Nutzungsbedingungen für die Daten enthalten und deren Einhaltung durch entsprechende Prozesse gewährleistet, kann die Datensouveränität erreicht werden. Diese Methoden und Prozesse werden unter dem Begriff Data Governance zusammengefasst.

Während unter dem Begriff Data Governance die organisatorischen Maßnahmen zusammengefasst werden, fallen unter den Begriff Usage Control technische Maßnahmen, welche darauf abzielen Nutzungsbedingungen technisch zu prüfen und durchzusetzen, und somit einen direkten Beitrag zur Erlangung und Wahrung der Datensouveränität leisten können. So handelt es sich beispielsweise bei der Zuweisung von Verantwortlichkeiten für Datenquellen zu Personen um eine Data Governance Maßnahme. Die Verwendung von spezialisierter Software, welche sicherstellt, dass Richtlinien, welche für einen Datensatz definiert wurden, eingehalten werden, ist dagegen eine Usage Control Maßnahme.

Der Begriff Usage Control wurde von Jaehong Park u. a. [Par04] geprägt und basiert auf einer Erweiterung der klassischen Modelle für Access Control. Usage Control erweitert die Nutzungskontrolle, welche bei Access Control Modellen während des Zugriffs stattfindet dahingehend, dass auch während und nach der Datennutzung Überprüfungen stattfinden können [Wu15]. Das nachfolgende Zitat stellt noch einmal heraus, wie das Access Control Modell durch Usage Control erweitert wird.

> „The distinguishing properties of UCON [usage control; Anmerk. d. Verf.] beyond traditional access control models are the continuity of access decisions and the mutability of subject and object attributes. In UCON, authorization decisions are not only checked and made before an access, but may be repeatedly checked during the access and may revoke the access if some policies are not satisfied, according to the changes of the subject or object attributes, or environmental conditions.“
> — Zhang u. a. [Zha05, S. 352]

Unter Subjekt ist dabei der Akteur gemeint, welcher auf das Objekt zugreifen möchte. Diese Erweiterung ist notwendig, da die Modelle nicht ausreichend sind, um die Anforderungen an die Sicherheit von verwendeten Daten in modernen Szenarien zu gewährleisten [Par02; San03]. Dabei existieren für verschiedene Einsatzgebiete unterschiedliche Ansätze für Usage

Control. Die existierenden Lösungen unterscheiden sich hinsichtlich der technischen Verfahren, aber auch in den Nutzungsbedingungen, welche mit der jeweiligen Lösung umgesetzt werden können [Laz10; Nyr11].

Durch die technische Repräsentation, Überprüfung und Durchsetzung von Nutzungsbedingungen adressiert Usage Control direkt das Thema der Datensouveränität. Dabei erlaubt die Verwendung von Usage Control Mechanismen die Unterstützung und stellenweise die Ablösung von Data Governance Verfahren, welche auf die Durchsetzung von Datennutzungsrichtlinien abzielen. Gleichzeitig entwickeln sich die Usage Control Mechanismen mit dem Laufe der Zeit stets weiter und erlauben es immer mehr Nutzungsbedingungen durchzusetzen. Dabei ist zu beachten, dass Usage Control nicht gänzlich ohne Data Governance auskommt. Dies liegt darin begründet, dass Data Governance eine wichtige Grundlage für funktionierende Usage Control Mechanismen darstellt. Darüber hinaus lassen sich beliebige Richtlinien zur Nutzungskontrolle definieren. Darunter befinden sich automatisch auch Nutzungsbedingungen, welche den aktuellen Stand der Usage Control Mechanismen übersteigen und daher technisch nicht umsetzbar sind.

Auch existieren verschiedene Initiativen und Projekte, welche u.a. die Erlangung und Wahrung der Datensouveränität zum Ziel haben. Bei FIWARE handelt es sich um eine Open Source Initiative, welche zum Einsatz in verschiedensten Szenarien, wie beispielsweise Smart Cities und IoT-Umgebungen, gedacht ist und als eines der Ziele die Datensouveränität der Anwender hat [Ara19; Rod18]. Dabei erlaubt FIWARE die Definition und Umsetzung von Nutzungsbedingungen auf Basis der XACML.

Eine weitere Initiative sind die International Data Spaces (IDS) [Ott18]. Die IDS entwickeln eine Referenzarchitektur und dazugehörige Komponenten und Verfahren, welche ein Ökosystem bilden, das den Datenaustausch zwischen den Teilnehmern ermöglicht [Ott19a]. Dabei ist die Wahrung und Erlangung der Datensouveränität eines der zentralen Ziele der IDS [Ott16]. Insbesondere findet in den IDS eine ausgiebige Betrachtung des Themas Usage Control inklusive der Entwicklung entsprechender Technologien, statt [Eit21; Eit19]. Eine Spezialisierung auf einzelne Fachdomänen sowie die Umsetzung der domänenspezifischen Anforderungen findet über Vertikalisierungen statt. Beispiele für solche Vertikalisierungen sind der Medical Data Space und der Industrial Data Space. Die vorliegende Arbeit wurde im Kontext der IDS entwickelt.

Während FIWARE und die IDS zunächst auf nationaler Ebene entstanden sind und sich mit der Zeit internationalisierten, existiert auch auf europäischer Ebene eine entsprechende Initiative namens GAIA-X. Das Ziel von GAIA-X ist die Schaffung einer europäischen Dateninfrastruktur [Bie20]. Dabei ist die Erlangung und Wahrung der Datensouveränität eines der zentralen Ziele [Mül21]. Dabei lässt sich feststellen, dass es sich bei der Datensouveränität aktuell primär um ein europäisches Thema handelt. Der Grund hierfür ist, dass amerikanische und asiatische Datenplattformen anderen Prinzipien folgen und durch andere Einflüsse geprägt sind [Ott19b].

Eine Gemeinsamkeit der zuvor genannten Initiativen ist die Unterstützung der Erlangung und Wahrung von Datensouveränität durch Usage Control Mechanismen. Dabei gib es unterschiedliche Möglichkeiten, wie Usage Control Mechanismen entwickelt und in Systemen verwendet werden können. Nachfolgend findet sich eine kurze Darstellung der Ziele für die vorliegende Arbeit und des Lösungsansatzes.

Im Folgenden wird der Aufbau des restlichen Dokuments beschrieben. In Kapitel 2 wird ein ausführlicher Überblick über verwandte Arbeiten gegeben. Dabei werden Arbeiten zum Thema der policy-agnostischen Programmierung betrachtet. Anschließend werden unterschiedliche Usage Control Ansätze, aufgeteilt in verschiedene Kategorien, betrachtet, gefolgt von einer dedizierten Betrachtung, inwiefern das Thema der Datennutzungskontrolle in Arbeiten zum Thema Mobile Code adressiert wird. Außerdem werden weitere Themengebiete, wie bspw. aspektorientierte Programmierung, betrachtet, welche ebenfalls wichtige Grundlagen für die vorliegende Arbeit darstellen.

Im Anschluss wird in Kapitel 3 der Forschungsansatz beschrieben. Dabei wird auf die Methodik eingegangen und aufgezeigt, wie Anforderungen für die vorliegende Arbeit definiert wurden. Anschließend werden die Forschungsfragen, welche diese Arbeit motiviert haben, ausführlich präsentiert. Es wird ebenfalls ein kurzer Überblick über die iterativ erzielten Arbeitsergebnisse gegeben.

In Kapitel 4 wird die Gesamtlösung der Arbeit vorgestellt. Dabei werden die unterschiedlichen Aspekte der Lösung nacheinander präsentiert und aufgezeigt, wie diese miteinander verbunden sind. Darüber hinaus enthält das Kapitel eine Übersicht über die einzelnen Veröffentlichungen, welche einen Beitrag zur vorgestellten Gesamtlösung haben.

Kapitel 5 präsentiert ausführlich einen Demonstrator, welcher verwendet wird, um die Arbeitsergebnisse zu präsentieren und zu validieren. Dabei wird zunächst das Szenario beschrieben und Anforderungen für die Datennutzungskontrolle definiert. Daraufhin werden die einzelnen Komponenten des Demonstrators betrachtet. Anschließend wird auf Basis eines ausführlichen Beispiels aufgezeigt, wie Datennutzungskontrolle in einer der entwickelten Applikationen integriert wird, und der Effekt von verwendeten Usage Control Richtlinien zur Laufzeit betrachtet.

Eine Bewertung der erzielten Arbeitsergebnisse findet in Kapitel 6 statt. Dabei wird untersucht, wie die in Kapitel 3 definierten Forschungsfragen durch einzelne Aspekte der Gesamtlösung adressiert und gelöst werden. Ebenfalls werden Limitierungen der präsentierten Lösung aufgezeigt.

Abschließend wird in Kapitel 7 eine kurze Zusammenfassung präsentiert. Ebenfalls findet sich in diesem Kapitel ein Ausblick, welcher mögliche zukünftige Arbeiten identifiziert, die zusätzliche Beiträge zur vorgestellten Lösung leisten können. Außerdem wird auf die ursprüngliche Motivation der Arbeit, welche in Kapitel 1 dargestellt wird, eingegangen und der Beitrag zur beschriebenen Problemstellung beschrieben.

KAPITEL 2

Grundlagen

In diesem Kapitel werden Arbeiten betrachtet und zu D° positioniert, welche verwandt zum Thema der vorliegenden Arbeit sind. Dies umfasst Arbeiten, welche sich mit der Entwicklung und Anwendung des Programmierparadigmas der Policy-agnostischen Programmierung befassen. Darüber hinaus werden Usage Control Ansätze betrachtet, welche nicht direkt Teil einer Programmiersprache sind, aber in den Programmcode von Applikationen integriert werden und somit Berücksichtigung von Usage Control bereits während der Anwendungsentwicklung erlauben. Auch betrachtet werden Ansätze, welche die Durchsetzung von Richtlinien in Mobile Code Szenarien ermöglichen. Des Weiteren werden andere Themengebiete wie beispielsweise die modellgetriebene Softwareentwicklung kurz betrachtet, da sie ebenfalls wichtige Grundlagen für die vorliegende Arbeit darstellen.

2.1 Policy-agnostische Programmierung

Das Programmierparadigma der Policy-agnostischen Programmierung ist aus der Arbeit von J. Yang [Yan15a] hervorgegangen. Mit der Programmiersprache Jeeves wird ein vielseitiges Werkzeug zur Verfügung gestellt, welches verwendet wird, um Richtlinien zur Kontrolle des Informationsflusses in Applikationen zu integrieren [Aus13; Yan12]. Das Ziel dabei ist, dass der Programmcode während der Entwicklung, unabhängig von den verwendeten Richtlinien, so aussieht, als ob keine Richtlinien vorhanden sind [Yan15b]. Diese Trennung von Applikationslogik und Richtlinien im Programmcode ist der zentrale Aspekt des Programmierparadigmas. Bei der Umsetzung von D° wurde der Grundgedanke des Programmierparadigmas – die Trennung von Applikationslogik und Richtlinien während der Entwicklung – übernommen und auf Usage Control Richtlinien anstelle von Informationsflussrichtlinien angewandt. Bei der Kontrolle des Informationsflusses durch Richtlinien handelt es sich um eine Teilmenge von Usage Control.

In Jeeves werden die Richtlinien mit einzelnen Datenobjekten verknüpft und während der Laufzeit wird von der Laufzeitumgebung sichergestellt, dass diese Richtlinien eingehalten werden. Dabei wird bei einem Verstoß die Ausführung nicht sofort abgebrochen. Besteht die Möglichkeit, die Ausführung durch eine Modifikation der Daten fortzusetzen, wird die entsprechende Anpassung durchgeführt. Beispielsweise kann die Verwendung von genauen Positionsdaten mithilfe einer Richtlinie verboten und die Daten zur Laufzeit dynamisch so angepasst werden, dass nur gröbere Daten (zum Beispiel die Stadt oder das Land anstelle einer vollständigen Anschrift) verwendet werden.

Dieses Vorgehen ist gut geeignet für Anwendungen, in denen Anwender direkt mit dem System interagieren und Daten bzw. Berechnungsergebnisse angezeigt bekommen. In der vorliegenden Arbeit wird die Programmiersprache D°, welche zur Datenverarbeitung in

industriellen Datenräumen verwendet wird, präsentiert. Die dynamische Modifikation der verwendeten Daten ist in solchen Szenarien nicht wünschenswert. Denn hierdurch können ungenaue bzw. falsche Berechnungsergebnisse entstehen, welche gegebenenfalls in Folgeschritten von Arbeitsabläufen verwendet werden und daraufhin Probleme verursachen. Aus diesem Grund verfügt die vorgestellte Programmiersprache D° nicht über eine solche Funktionalität. Darüber hinaus beschränkt D° sich nicht auf die Kontrolle des Informationsflusses, sondern ermöglicht es auch die eigentliche Nutzung der Daten zu kontrollieren und zu beschränken.

Da die Richtlinien bei Jeeves an die einzelnen Dateninstanzen geheftet werden, eignet sich die Sprache gut für Systeme, welche mit Daten arbeiten, bei denen jeder Datensatz potentiell über individuelle Richtlinien verfügt. D° kann ebenfalls für die Entwicklung solcher Systeme verwendet werden, adressiert aber primär andere Szenarien: In mehrschrittigen, datenverarbeitenden Prozessen werden eine Vielzahl von gleichartigen Datensätzen verarbeitet. Dabei soll die Verarbeitung jedes einzelnen Datensatzes einen festen Satz von Richtlinien einhalten. Ausnahmen von diesem festen Satz treten nur in Ausnahmefällen auf. Um die einzelnen Applikationen, welche die Aktivitäten des Prozesses darstellen, umzusetzen und dabei die notwendigen Richtlinien zu integrieren, kann D° verwendet werden. Somit werden im Kontext von D° die Richtlinien als ein Bestandteil der Applikation aufgefasst, wohingegen Jeeves Richtlinien als Metadaten, welche an den eigentlichen Dateninstanzen hängen, interpretiert. Aus diesem Grund erlaubt es D° die umzusetzenden Richtlinien sowohl an die eigentlichen Daten, als auch an die Elemente, welche innerhalb von Applikationen verwendet werden, zu heften.

Das Paradigma der Policy-agnostischen Programmierung bildet die Grundlage für diverse Arbeiten. Die Arbeit von Palesha [Pal17] integriert das Paradigma in JavaScript, um Richtlinien zur Kontrolle des Informationsflusses clientseitig direkt im Browser umsetzen zu können. Hierdurch wird es möglich, Konstrukte der Policy-agnostischen Programmierung in einer interpretierten Umgebung auszuführen, welche nicht selbstständig lauffähig ist. D° wird verwendet, um alleine lauffähige Applikationen zu entwickeln und adressiert somit eine andere Domäne. Analog zu bewerten sind weitere Arbeiten, die das Paradigma für Webanwendungen bzw. Schnittstellen selbiger umsetzen [Bic18; Han14; Mil15; Yan15b].

Micinski u. a. [Mic18] stellen in ihrer Arbeit eine Implementation des Paradigmas auf Basis von Makros für die Programmiersprache Racket vor. Bei Racket handelt es sich um eine Programmiersprache, welche verwendet wird, um andere Programmiersprachen zu definieren und in Form von Bibliotheken verfügbar zu machen [Fel15]. Racket erlaubt somit die effektive Entwicklung von (eingebetteten) domänenspezifischen Programmiersprachen. Durch die Arbeit von Micinski u. a. [Mic18] kann die Informationsflusskontrolle während der Entwicklung neuer Programmiersprachen berücksichtigt werden, indem Makros für Faceted Evaluation [Aus12] verwendet werden. Faceted Evaluation erlaubt es Faceted Values zu definieren. Dabei handelt es sich um Daten, welche mehrere mögliche Werte enthalten. Zur Laufzeit wird auf Basis des Aufrufers entschieden, welcher Wert verwendet wird. Dabei verfügt die Arbeit über zusätzliche Abstraktionen, die statische Analysen erlauben, wodurch die Anzahl der notwendigen Laufzeitüberprüfungen reduziert werden kann [Mic20]. Die Entwicklung neuer Programmiersprachen ist eine andere Domäne als die von D° adressierte.

Bei LIFTY handelt es sich um eine Programmiersprache, welche über integrierte Mechanismen zur Definition und Integration von Informationsflussrichtlinien verfügt und von Polikarpova u. a. [Pol16] entwickelt wurde. Dabei setzt LIFTY das Paradigma der Policy-agnostischen Programmierung um. Während der Entwicklung muss der Entwickler die notwendigen Richtlinien in der Applikation annotieren. Der LIFTY-Compiler stellt durch Programmsynthese sicher, dass die erzeugte Applikation über die notwendigen Überprüfungen an den korrekten Stellen verfügt. Genau wie D° implementiert LIFTY das Paradigma statisch, d.h. die notwendigen Überprüfungen für Richtlinien werden nicht während der Laufzeit in den Ablauf der Applikation integriert. LIFTY erfordert vom Entwickler, dass die notwendigen Richtlinien an den korrekten Stellen im Applikationscode annotiert werden. Demgegenüber zielt D° darauf ab die Verwendung von Richtlinien gar nicht im Applikationscode sichtbar zu machen, um eine möglichst starke Trennung zwischen der Applikationslogik und den Richtlinien zu erreichen und mögliche Fehlerquellen zu vermeiden.

2.2 Usage Control Ansätze

Bei Usage Control handelt es sich um eine Weiterentwicklung des bekannten Access Control Modells. Diese Weiterentwicklung ist notwendig, da Access Control alleine nicht mehr ausreichend ist, um die Anforderungen an den Schutz von Daten, über die moderne Applikationen und Szenarien verfügen, zu erfüllen [Laz10; Par04; San03]. Dabei wurde der Begriff Usage Control maßgeblich von Jaehong Park u. a. [Par04] geprägt, welche die UCON_{ABC} definieren [Par04; Par02; San03].

Traditionelle Access Control Modelle basieren auf Attributen, die an Objekten und Subjekten hängen. Diese Attribute werden während der Autorisierung verwendet, um Entscheidungen über den Zugriff auf Daten zu treffen [Lin06; San98a; San98b]. Dabei basieren diese Modelle im Kern typischerweise auf dem Access Matrix Modell [Sam00]. Bei diesem Modell wird eine Matrix angelegt, in der Zeilen zu Subjekten und Spalten zu Objekten zugeordnet werden. Die einzelnen Zellen enthalten die Aktionen, für die das Subjekt auf das Objekt zugreifen darf. Bei Access Control werden zur Zugriffszeit Entscheidungen getroffen, welche den Zugriff auf Daten regulieren. Die Nutzung von Daten ist jedoch eine fortlaufende Aktivität, und zur Regulierung dieser Datennutzung reicht eine einmalige Prüfung zur Zugriffszeit nicht aus. Es sind zusätzliche Überprüfungen während der Verarbeitung notwendig, welche sich beispielsweise auf die Ausführungsumgebung oder der Art und den Zweck der Datennutzung beziehen.

Das UCON_{ABC} Modell erweitert das Access Control Modell um Verpflichtungen (Obligations) und Bedingungen (Conditions). Verpflichtungen werden dabei verwendet, um Anforderungen zu definieren, welche von Subjekten vor bzw. während dem Zugriff auf geschützte Objekte erfüllt werden müssen. Beispiele für solche Verpflichtungen sind die Hinterlegung bestimmter Daten eines Subjekts vor dem Zugriff auf Dokumente oder die regelmäßige Übermittlung von Logdaten während der Nutzung einer Software. Bedingungen definieren keine Anforderungen an die Subjekte, welche den Zugriff tätigen, sondern solche, die die ausführenden Systeme oder die Umgebung betreffen. Beispielsweise können Bedingungen verwendet werden, um Anforderungen an den Zeitpunkt der Ausführung, den Ort der Ausführung oder die Hardware der ausführenden Systeme zu definieren.

Durch die Erweiterung um Verpflichtungen und Bedingungen erlaubt es Usage Control neben einfachen Zugriffsrichtlinien auch Nutzungsrichtlinien umzusetzen, welche weitergehende Prüfung vor und während des Zugriffs erfordern. Zu diesem Zweck werden bei Usage Control Mechanismen neben der Autorisierung auf Basis von Rechten und Attributen auch Verpflichtungen und Bedingungen mit in die Entscheidungsfindung einbezogen. Abbildung 2.1 enthält eine Darstellung der einzelnen Komponenten des UCON_{ABC} Modells, welche bei der Entscheidungsfindung berücksichtigt werden.

Aufgrund der wachsenden Relevanz des Themas Usage Control sind in der Vergangenheit diverse Arbeiten entstanden, die sich mit dem Thema beschäftigten[Kas15; Laz08; Laz10; Nyr11; Pre08]. Diese lassen sich grob in drei Klassen unterteilen, welche nachfolgend betrachtet werden.

2.2.1 Usage Control in Applikationen

Soll eine Applikation mit Usage Control Mechanismen versehen werden, ist eine direkte Integration der entsprechenden Mechanismen in die Applikation eine naheliegende Lösung. Hierbei werden die entsprechenden Usage Control Mechanismen bereits während der Entwicklung, oder zumindest Teilen selbiger, berücksichtigt und in den Programmcode der Applikation integriert. Der Vorteil bei diesem Vorgehen ist, dass die Usage Control Mechanismen ein fester Bestandteil der entwickelten Applikation und somit untrennbar mit dieser verbunden sind. Darüber hinaus ist eine große Menge an Informationen über die internen Abläufe der Applikation vorhanden, da der Programmcode vorliegt und gegebenenfalls von den selben Personen entwickelt wurde, welche die Usage Control Mechanismen integrieren. Diese Informationen sind notwendig, um viele (komplexere) Nutzungsrichtlinien durchzusetzen.

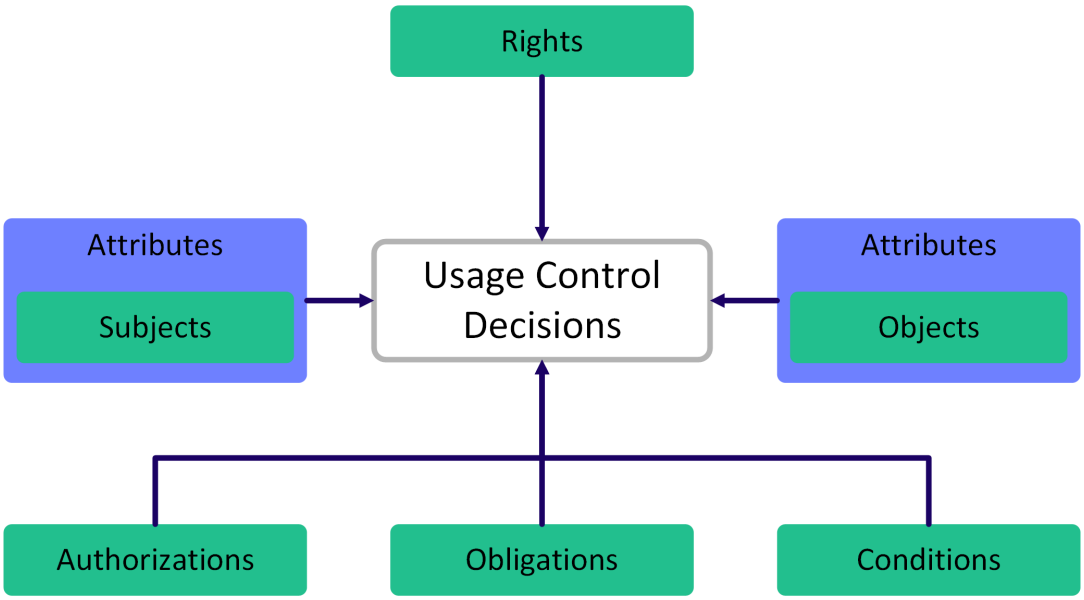

Abbildung 2.1: Darstellung der einzelnen Komponenten des UCON_{ABC} Modells. Angelehnt an [Par04, S. 136]

Sollen beispielsweise schreibende Zugriffe auf Festplatten durch eine Nutzungsrichtlinie kontrolliert und eingeschränkt werden, ist es notwendig zu wissen, ob und an welchen Stellen die Applikation entsprechende Aktionen durchführt. Je detaillierter dieses Wissen über die Funktionsweise und Abläufe der Applikation vorhanden ist, desto präziser und feingranularer lassen sich die entsprechenden Usage Control Mechanismen an den notwendigen Stellen integrieren. Dieses feingranulare Vorgehen kann sicherstellen, dass die Usage Control Mechanismen an allen notwendigen Stellen integriert werden und gleichzeitig keine Verwendung in anderen Bereichen der Applikation finden.

Eine Integration von Usage Control direkt in die Applikation bietet einen weiteren Vorteil: Den Mechanismen kann unmittelbarer Zugriff auf relevante interne Daten der Applikationen gewährt werden, ohne dass diese nachgebildet, kopiert oder übertragen werden müssen. Durch die Verfügbarkeit dieser Daten, in Kombination mit den Kenntnissen über die internen Abläufe der Applikation, ist es möglich eine große Anzahl von Nutzungsrichtlinien umzusetzen. Einige dieser Richtlinien sind bei den anderen Ansätzen nicht oder nur mit Mehraufwänden realisierbar.

Dieses Vorgehen zur Verwendung von Usage Control birgt auch Nachteile. Bei Usage Control handelt es sich um einen querschneidenden Belang, welcher die Komplexität der Applikationsentwicklung erheblich steigern kann [Raj09]. Diese Komplexitätssteigerung tritt bereits bei Access Control Mechanismen auf [Jea01]. Zum einen müssen die entsprechenden Usage Control Mechanismen an allen relevanten Stellen integriert werden. Zum anderen müssen diese Mechanismen auch korrekt verwendet werden. Dies ist ein komplexer Prozess, der sehr anfällig für menschliche Fehler ist.

Darüber hinaus ist dieses Vorgehen nur realistisch für Software, welche sich noch in der Entwicklung befindet bzw. bei der die Entwicklung noch nicht gestartet ist. Zum einen ist dieses Vorgehen für bereits fertiggestellte Software, welche sich unter Umständen bereits im Betrieb befindet, häufig nicht realisierbar, da gegebenenfalls die notwendige Verfügbarkeit des Programmcodes nicht gegeben ist. Zum anderen ist die nachträgliche Integration von Usage Control Mechanismen in bereits fertiggestellte Applikationen mit einem massiven Aufwand verbunden. Durch eventuell wechselnde Entwickler ist das ursprünglich vorhanden gewesene Wissen über die internen Abläufe der Applikation möglicherweise inzwischen verloren gegangen. Hierdurch erreicht die nachträgliche Integration von Usage Control Mechanismen in Applikationen eine Komplexität, welche einer Neuentwicklung nahe kommt, oder dieser sogar übersteigt. Auch eine Neuentwicklung bestehender Software unter Berücksichtigung und Integration von Usage Control Mechanismen ist nicht zielführend, da auch in diesem Fall die zuvor genannte hohe Komplexität auftritt. In diesen Szenarien ist die Verwendung eines anderen Usage Control Ansatzes zielführender.

Abbildung 2.2 enthält eine schematische Darstellung für eine Umsetzung von Usage Control Mechanismen direkt in einer Applikation. Es ist erkennbar, dass die Mechanismen nur an den Stellen der Applikation integriert werden, an denen auch tatsächlich Usage Control notwendig ist. Die Abbildung macht zum einen deutlich, dass diese Kategorie von Usage Control Mechanismen ein stark individualisiertes und maßgeschneidertes Usage Control System ermöglichen, veranschaulicht jedoch gleichzeitig, dass ein hohes Maß von Expertise und Aufmerksamkeit bei der Umsetzung dieses Ansatzes notwendig ist.

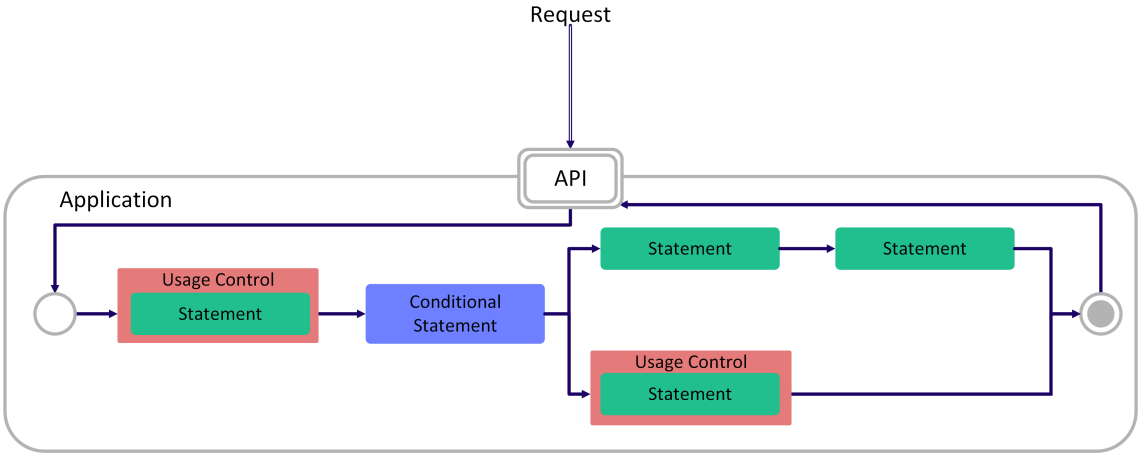

Abbildung 2.2: Schematische Darstellung, wie Usage Control Mechanismen als fester Bestandteil in eine Applikation integriert werden und relevante Teile der Applikation ummanteln.

Bei D°, der Programmiersprache, welche im vorliegenden Dokument beschrieben wird, handelt es sich um eine Lösung, die den Ansatz der direkten Integration von Usage Control Mechanismen in die Applikation verfolgt. Durch die Implementation des Programmierparadigmas der policy-agnostischen Programmierung wird die Integration der Mechanismen an den relevanten Stellen vereinfacht und menschliche Fehler reduziert. Darüber hinaus verwendet D° Codegenerierung, um eine korrekte Verwendung der integrierten Usage Control Mechanismen sicherzustellen. Daraus folgt, dass bei der korrekten Verwendung der einzelnen Usage Control Mechanismen keine Fehler durch den Entwickler entstehen können. Ist sichergestellt, dass die Codegenerierung von D° korrekt arbeitet, kann eine Vielzahl von Applikationen mit integrierten Usage Control Mechanismen entwickelt werden, welche diese korrekt integrieren.

Eine populäre Umsetzung von Usage Control sind Systeme für das Digital Rights Management (DRM) [Sub06]. DRM Systeme finden sich häufig in Software, welche zur Verwendung von Mediendateien (bspw. Textdokumenten und Videos) verwendet werden. Dabei hängen die Nutzungsrichtlinien an den zu nutzenden Medien und die DRM Systeme stellen sicher, dass eine Nutzung der Daten nur gemäß der anhängenden Richtlinien erfolgt. Es existieren unterschiedliche DRM Lösungen für die verschiedenen Medienformate [Fet03; Liu03], wie beispielsweise Musik [Kwo02] oder Dokumente [Bao04]. Die verwendbaren Nutzungsrichtlinien der jeweiligen DRM Systeme sind dabei maßgeschneidert für die unterstützten Medienformate und eignen sich nicht zur Abbildung allgemeiner Nutzungsrichtlinien. Darüber hinaus sind DRM Systeme spezialisiert auf die jeweils unterstützten Medienformate und verfügen häufig über eine Herstellerbindung. Somit lassen sich DRM Systeme nicht in beliebige Applikationen integrieren, um Nutzungsbedingungen für beliebige Daten durchzusetzen.

In der Arbeit von Costa [Cos11] wird eine Erweiterung der Programmiersprache Java präsentiert, welche es erlaubt Java Applikationen mit zusätzlichen Richtlinien auszustatten [Bar09]. Eine statische Analyse des Codes stellt sicher, dass nur Richtlinien, welche nicht

immer erfüllt sind, zur Laufzeit überprüft werden. Zu diesem Zweck müssen feingranulare Richtlinien definiert werden. Dabei werden die Richtlinien auf Basis einzelner Methoden und deren Eingabeparametern definiert. Im Applikationscode muss Code, welcher Richtlinien einhalten muss, in einer Sandbox ausgeführt werden, welche die relevanten Richtlinien enthält. Dies ist sowohl anfällig für menschliche Fehler, als auch einfach zu umgehen für bösartige Entwickler, da die Richtlinien für einzelne Methoden definiert werden. Bei D° handelt es sich um eine eigenständige Programmiersprache, welche unter anderem darauf abzielt die Komplexität der korrekten Verwendung von Usage Control Mechanismen durch Codegenerierung zu abstrahieren, um menschliche Fehler zu vermeiden. Darüber hinaus sind die Richtlinien, welche von D° verwendet werden, abstrakter und nicht so feingranular wie in der beschriebenen Erweiterung für Java.

2.2.2 Usage Control im Parallelbetrieb

Aus zuvor genannten Gründen ist die direkte Integration von Usage Control Mechanismen in Applikationen nicht immer ein praktikabler Weg. Dennoch gibt es häufig den berechtigten Bedarf Applikationen, welche sich bereits in Entwicklung oder im Betrieb befinden, mit Usage Control Mechanismen auszustatten. In diesen Fällen können die Usage Control Mechanismen wie eine Schale um die Schnittstellen der Applikation gelegt werden. Bei diesem Vorgehen erfolgen keine direkten Aufrufe an die Applikation. Stattdessen wird die Usage Control Schale aufgerufen, welche nur Anfragen, die den definierten Nutzungsrichtlinien nicht widersprechen, an die eigentliche Applikation weiterleitet. Die Funktionsweise dieser Usage Control Schale ist vergleichbar mit einem Aspekt im Kontext der aspektorientierten Programmierung.

Dabei muss sichergestellt werden, dass keine Aufrufe direkt an die Applikation durchgeführt werden können, da dies die Usage Control Mechanismen umgehen würde. Für Applikationen, welche über eine REST-Schnittstelle verfügen, kann dies beispielsweise durch die Verwendung eines Proxy-Servers erfolgen, welcher keine Aufrufe der REST-Schnittstelle zulässt. Dabei muss beachtet werden, dass dieser Ansatz für Usage Control nicht für alle Applikationen verwendet werden kann. Wenn eine Applikation auf eine integrierte graphische Nutzeroberfläche setzt, gibt es keine Schnittstellen, um die Usage Control Mechanismen gelegt werden können. Applikationen, die über getrennte Front- und Backends verfügen, sind hiervon nicht betroffen.

Im Vergleich zur direkten Integration von Usage Control Mechanismen gibt es bei diesem Vorgehen einige Unterschiede bei der Umsetzung von Nutzungsrichtlinien. Diese werden an dieser Stelle nur grob beschrieben, eine ausführliche Betrachtung der Mächtigkeit der unterschiedlichen Ansätze findet sich in Abschnitt 4.1.3. Richtlinien, welche kein Wissen über die internen Abläufe von Applikationen erfordern, können mit diesem Ansatz ohne Probleme umgesetzt werden. Sollen mit diesem Ansatz Nutzungsrichtlinien umgesetzt werden, welche Wissen über die internen Abläufe der Applikation erfordern, fallen zusätzliche Arbeiten an.

Als Beispiel betrachten wir eine Nutzungsrichtlinie, welche einer Applikation verbietet die erhaltenen Eingabedaten zu persistieren. Um diese Richtlinie mit einem separat betriebenem Usage Control Mechanismus umzusetzen, sind diverse Fragen zu beantworten:

- Welche Endpunkte der Applikationsschnittstelle führen persistierende Aktionen aus?

- Werden für alle möglichen Eingabeparameter persistierende Aktionen ausgeführt?
- Werden die persistierenden Aktionen auf allen Ausführungspfaden der jeweiligen Endpunkte ausgeführt?

Die vorherige Auflistung von Fragen ist nicht erschöpfend und zeigt, dass abhängig von der jeweiligen Richtlinie eine nicht unerhebliche Menge von Fragen beantwortet werden muss. Dabei kann die Erlangung des hierzu notwendigen Wissens eine beliebige Komplexität erreichen. Ein wichtiger Faktor hierbei ist welche Ressourcen zur Applikation verfügbar sind. Zugriff auf den Programmcode der Applikation oder eine ausführliche Dokumentation können diese Aufgabe erheblich vereinfachen.

Sind diese Fragen beantwortet worden, kann eine Umsetzung der Richtlinie erfolgen. Abhängig von den Antworten auf diese Fragen ist es anschließend notwendig die Usage Control Mechanismen entsprechend zu gestalten bzw. zu konfigurieren. Hierbei wird ein Teil der eigentlichen Applikationslogik im verwendeten Usage Control Mechanismus repliziert, um die internen Abläufe der Applikation abzubilden. Dies ist notwendig, um eine passgenaue Umsetzung der Richtlinien zu erreichen. Andernfalls wird die Richtlinie beispielsweise auf Ausführungspfade von Endpunkten angewendet, welche eigentlich nicht von der Richtlinie betroffen sind, oder betroffene Ausführungspfade sind nicht durch die Richtlinienumsetzung abgedeckt.

Alternativ kann ein Usage Control Mechanismus die Interaktion der Applikation mit dem ausführenden System überwachen und bei entsprechenden Ereignissen Richtlinien überprüfen und umsetzen. Dabei wird der Usage Control Mechanismus parallel zur Applikation betrieben. Der Vorteil bei diesem Vorgehen ist, dass es auch bei Applikationen mit integrierten graphischen Nutzeroberflächen verwendet werden kann. Der Nachteil ist, dass man bei der Überprüfung und Umsetzung von Richtlinien auf die Interaktionspunkte der Applikation mit dem ausführenden System (beispielsweise Systemaufrufe) beschränkt ist. Wird eine solche Interaktion, und damit der Usage Control Mechanismus ausgelöst, sind wenig Informationen über den Kontext der Interaktion verfügbar. Beispielsweise sind bei einem Systemaufruf neben den verwendeten Parametern keine Informationen verfügbar, welche bei der Überprüfung und Umsetzung von Richtlinien hilfreich sein können. Sofern aus den Parametern nicht abgeleitet werden kann aus welchem Kontext der Systemaufruf getätigt wurde, fehlen diese Informationen, die unter Umständen für feingranulare Richtlinien notwendig sind. Beispielsweise könnte eine Richtlinie, welche schreibende Operationen auf ein bestimmtes Verzeichnis nur zum Zwecke des Loggings erlaubt, mit diesem Vorgehen häufig nicht umgesetzt werden.

Neben der Möglichkeit eine Vielzahl von Applikationen nachträglich mit Usage Control Mechanismen auszustatten, bietet dieser Ansatz einen weiteren Vorteil: Die Entwicklung der Usage Control Mechanismen ist entkoppelt von der Entwicklung der eigentlichen Applikation. Somit wird die Komplexität der Applikationsentwicklung nicht durch die Usage Control Mechanismen beeinflusst. Abbildung 2.3 zeigt eine schematische Darstellung für diese Art von Usage Control Anwendungen. Es ist zu erkennen, dass sich die Usage Control Komponente um die komplette Applikation legt und so direkte Zugriffe auf die Applikation verhindert, ohne selber ein Bestandteil der Applikation zu sein.

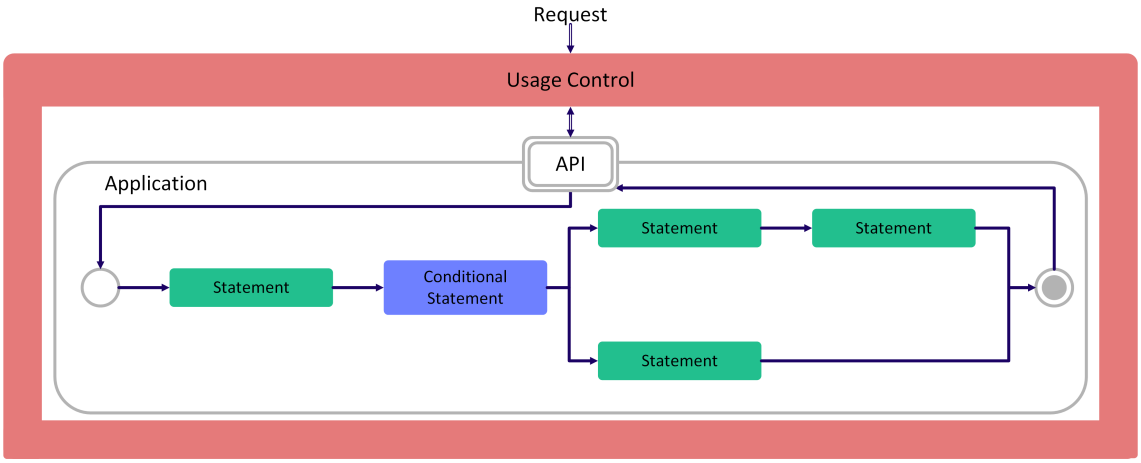

Abbildung 2.3: Schematische Darstellung, wie Usage Control Mechanismen als Schale um eine Applikation gelegt werden und alle Aufrufe entgegennehmen.

Ein Beispiel für eine Usage Control Lösung, welche auf dieses Verfahren setzt ist MYDATA, welches auf dem IND^{2}UCE Framework basiert [Jun15; Ste16]. MYDATA ist eine der Usage Control Lösungen, welche in den International Data Spaces vorangetrieben und eingesetzt wird. Dabei handelt es sich bei MYDATA um eine Erweiterung der eXtensible Access Control Markup Language (XACML).

Bei XACML handelt es sich um eine Sprache zur Definition von Datenzugriffsrichtlinien auf Basis von XML [Sta13]. Neben der eigentlichen Sprache umfasst die Spezifikation Modelle zum Datenfluss, welche verwendet werden können, um eine Architektur zur Durchsetzung der Richtlinien abzuleiten. Dabei sehen die Modelle vier zentrale Komponenten vor, welche an der Richtliniendurchsetzung beteiligt sind.

Abbildung 2.4 zeigt eine schematische Darstellung der einzelnen XACML Komponenten. Die grünen Elemente der Abbildung sind Teil von XACML. Die blauen Elemente gehören zu dem umliegenden System, welches durch XACML mit Usage Control angereichert werden soll.

PAP Der Policy Administration Point wird verwendet, um die Richtlinien für das System festzulegen.

PEP Der Policy Enforcement Point empfängt die Aufrufe an die Applikation und leitet sie an die Applikation weiter, sofern die definierten Richtlinien durch den jeweiligen Aufruf nicht verletzt werden.

PDP Der Policy Decision Point wird vom PEP aufgerufen und trifft die eigentliche Entscheidung, ob die definierten Richtlinien durch einen Aufruf eingehalten werden oder nicht. Dabei werden die Richtlinien vom PAP geliefert.

PIP Der Policy Information Point kann vom PDP verwendet werden, um zusätzliche Informationen, welche zur Durchsetzung der Richtlinien notwendig sind, abzufragen.

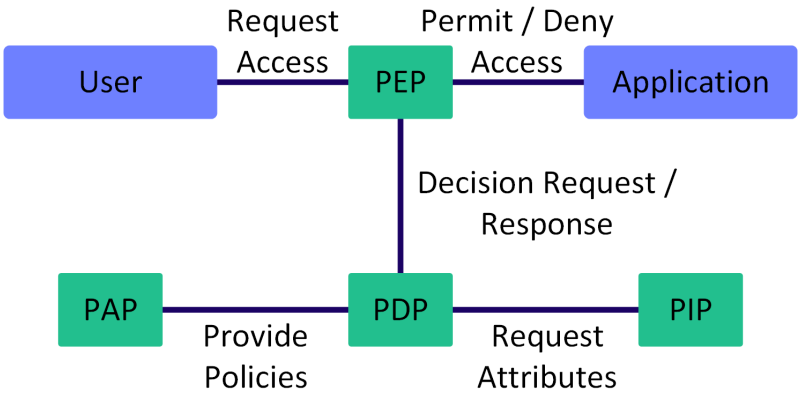

Abbildung 2.4: Schematische Darstellung der einzelnen XACML Komponenten und der Verbindungen untereinander.

MYDATA besitzt eine eigene XML-basierte Sprache zur Definition von Datennutzungsrichtlinien und setzt nicht auf die Sprache von XACML. Dennoch verwendet es die beschriebene XACML Architektur und erweitert sie um zusätzliche Elemente, um weitere Funktionalitäten zu bieten und die Umsetzung von Usage Control Richtlinien zu erlauben. MYDATA fügt die folgenden neuen Elemente zur XACML-Architektur hinzu:

Abbildung 2.5 zeigt eine schematische Darstellung, welche die Komponenten von MYDATA enthält und darstellt, wie diese Komponenten miteinander verbunden sind. Blaue Elemente der Abbildung gehören nicht zu MYDATA, sondern dem umliegenden System, in welchem MYDATA eingesetzt wird. Grüne Elemente sind Teil von MYDATA und bereits aus XACML bekannt, wohingegen die roten Elemente neue Elemente von MYDATA darstellen.

Es ist erkennbar, dass sich die Verbindungen der Komponenten untereinander leicht geändert hat gegenüber dem Aufbau von XACML. Der PAP hat keine direkte Verbindung zum PDP mehr. Stattdessen ist der MYDATA exklusive Policy Management Point (PMP) mit diesen beiden Komponenten verbunden und bietet einen erweiterten Funktionsumfang.

PXP Der Policy Execution Point erlaubt es Aktionen unabhängig von eingehenden Anfragen auszuführen und ermöglicht dadurch die Umsetzung zusätzlicher Richtlinien, beispielsweise wenn periodisch Daten protokolliert werden müssen. Der PXP kann vom PDP bei der Auswertung von Richtlinien aufgerufen werden.

PMP Der Policy Management Point nimmt Richtlinien vom PAP entgegen. Der PMP erlaubt das Ausrollen und Zurückrufen von Richtlinien im PDP.

PRP Der Policy Retrieval Point dient als gesicherter Ablageort für verwendete Richtlinien und wird vom PMP befüllt und abgefragt.

Eine weitere Usage Control Lösung, welche diesen Ansatz verwendet, ist UC4Win [Wüc12]. UC4Win wird verwendet, um die Systemaufrufe, welche von Windows bereitgestellt werden, zu überwachen. Das System erlaubt die Definition von XML-basierten Richtlinien,

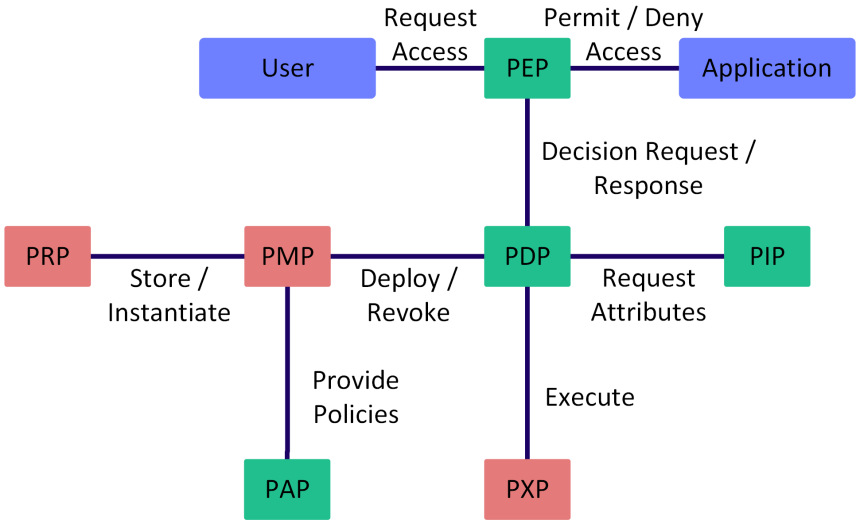

Abbildung 2.5: Schematische Darstellung der einzelnen MYDATA Komponenten und der Verbindungen untereinander.

welche beim Aufruf von Systemfunktionen von Windows ausgewertet werden. Dabei zielt UC4Win auf das Verhindern von Datenverlusten ab und adressiert nicht das Umsetzen von Nutzungseinschränkungen für Daten.

2.2.3 Usage Control als zentrale Komponente

Nicht immer ist es das Ziel einzelne Applikationen mit Usage Control Mechanismen auszustatten. Häufig sollen ganze Arbeitsabläufe, welche mehrere verschiedene Applikationen einsetzen, mit Usage Control Mechanismen versehen werden. In solchen Szenarien ist es möglich, eine der bereits vorgestellten Klassen von Uage Control Mechanismen zu verwenden. Es bietet sich aber eine weitere Möglichkeit an, welche es erlaubt den Einsatz von Usage Control Mechanismen zu zentralisieren.

Anstatt einer Betrachtung und Implementation auf Applikationsebene, findet eine Betrachtung auf der Ebene des gesamten Szenarios statt. In vielen Fällen wird in solchen Szenarien eine zentrale Komponente verwendet, welche von den Applikationen zur Kommunikation verwendet wird, beispielsweise ein Nachrichtenbus. Ist dies der Fall, kann dieser Nachrichtenbus mit Usage Control Mechanismen ausgestattet werden, welche die Richtlinien für alle im Szenario verwendeten Applikationen enthält und umsetzt.

Dabei sind die Vor- und Nachteile dieses Vorgehens stark überschneidend mit denen der Usage Control Schale. Die verwendeten Usage Control Mechanismen und die zu schützenden Applikationen sind voneinander entkoppelt. Bedingt durch diese Entkoppelung ist die Umsetzung von Richtlinien, welche von den internen Abläufen der Applikation abhängen, schwieriger durchzusetzen. Dies führt dazu, dass Teile des Applikationsverhaltens in dem Usage Control Mechanismus nachgebaut werden müssen, um eine passgenaue Umsetzung der Richtlinien zu gewährleisten. Dabei muss beachtet werden, dass bei diesem Vorgehen ein Usage Control Mechanismus für eine größere Menge von Applikationen verwendet wird,

was zu einer erheblichen Komplexität innerhalb des Usage Control Mechanismus führen kann.

Abbildung 2.6 zeigt eine Schematische Darstellung eines Systems, welches aus zwei Komponenten besteht. Es ist erkennbar, dass in der einen Komponente eine einzelne Applikation betrieben wird, welche über eine Schnittstelle nach außen verfügt. In der anderen Komponente wird eine Applikation betrieben, welche die Schnittstelle der Applikation in der zuerst genannten Komponente aufruft. Dabei findet die Kommunikation über einen Kanal statt, welche mit Usage Control Mechanismen ausgestattet ist. Aufgrund der Integration der Usage Control Mechanismen in den zentralen Kommunikationskanal, kann auf die Verwendung von Usage Control in den individuellen Komponenten und Applikationen verzichtet werden.

Ein Beispiel für einen solchen Ansatz ist LUCON aus den International Data Spaces. LUCON (Logic-based Usage Control) ist dabei ein fester Bestandteil des Trusted Connectors. Der Connector ist im Kontext der IDS ein zentrales Infrastrukturelement, welches den Teilnehmern die Kommunikation und den Datenaustausch untereinander ermöglicht [Ott19a]. LUCON setzt auf einen Label-basierten Ansatz [Sch18]. Dieser erlaubt es Datensätze mit Labeln zu versehen. Diese Labels können sich während der Verarbeitung ändern und beispielsweise den aktuellen Zustand des Datensatzes beschreiben. Auf Basis dieser Labels werden Richtlinien definiert, welche zwischen den einzelnen Schritten eines Arbeitsablaufs überprüft werden und darüber entscheiden, ob die Ausführung fortgesetzt werden kann. Die Richtlinien für LUCON werden in einer eigenen DSL Sprache definiert, welche zu Prolog-Programmen übersetzt werden und es ermöglichen den Datenfluss zwischen den einzelnen Applikationen auf Basis der Labels einzuschränken.

Es existieren verschiedene Arbeiten, welche Usage Control Mechanismen speziell für Cloud-Umgebungen beschreiben [Car16; Laz12; Tav12]. Diese Arbeiten legen den Fokus dabei auf die Nutzung von Hardwareressourcen innerhalb der Cloud-Umgebung und haben daher geringe Überschneidungen mit D°. Demgegenüber existieren auch Arbeiten, welche Usage Control Mechanismen in Cloud Umgebungen integrieren und dabei den Fokus auf den Schutz von Daten legen. Die Arbeit von Huang u. a. [Hua14] präsentiert einen DRM Mechanismus für Cloud-Umgebungen auf Basis von attributbasierter Verschlüsselung [Sah05]. Dabei beschränkt sich die Arbeit auf den Zugriff auf Daten sowie Nutzungsrechte und führt keine Überprüfungen zur Laufzeit von Applikationen durch, nachdem der Zugriff auf Daten gewährt wurde.

Die Arbeit von La Marra u. a. [La 17a] erweitert das Protokoll MQTT um Usage Control Mechanismen. Dabei wird der zentrale Broker, welcher in MQTT Umgebungen verwendet wird, um Nachrichten entgegen zu nehmen und zu verteilen, um entsprechende Mechanismen erweitert, um IoT-Umgebungen mit Usage Control Mechanismen auszustatten. Dabei können Richtlinien definiert werden, welche durch den Broker ausgewertet werden, sobald ein Client eine Subscription für ein Topic durchführen möchte. Hierdurch wird es ermöglicht Zugriffe auf Topics und die darin enthaltenen Daten auf Basis der Richtlinien zu gestatten oder zu verhindern. Der Ansatz lässt sich auf weitere Anwendungsgebiete anwenden, beispielsweise verteilte Datenbanken wie Apache Cassandra [La 17b].

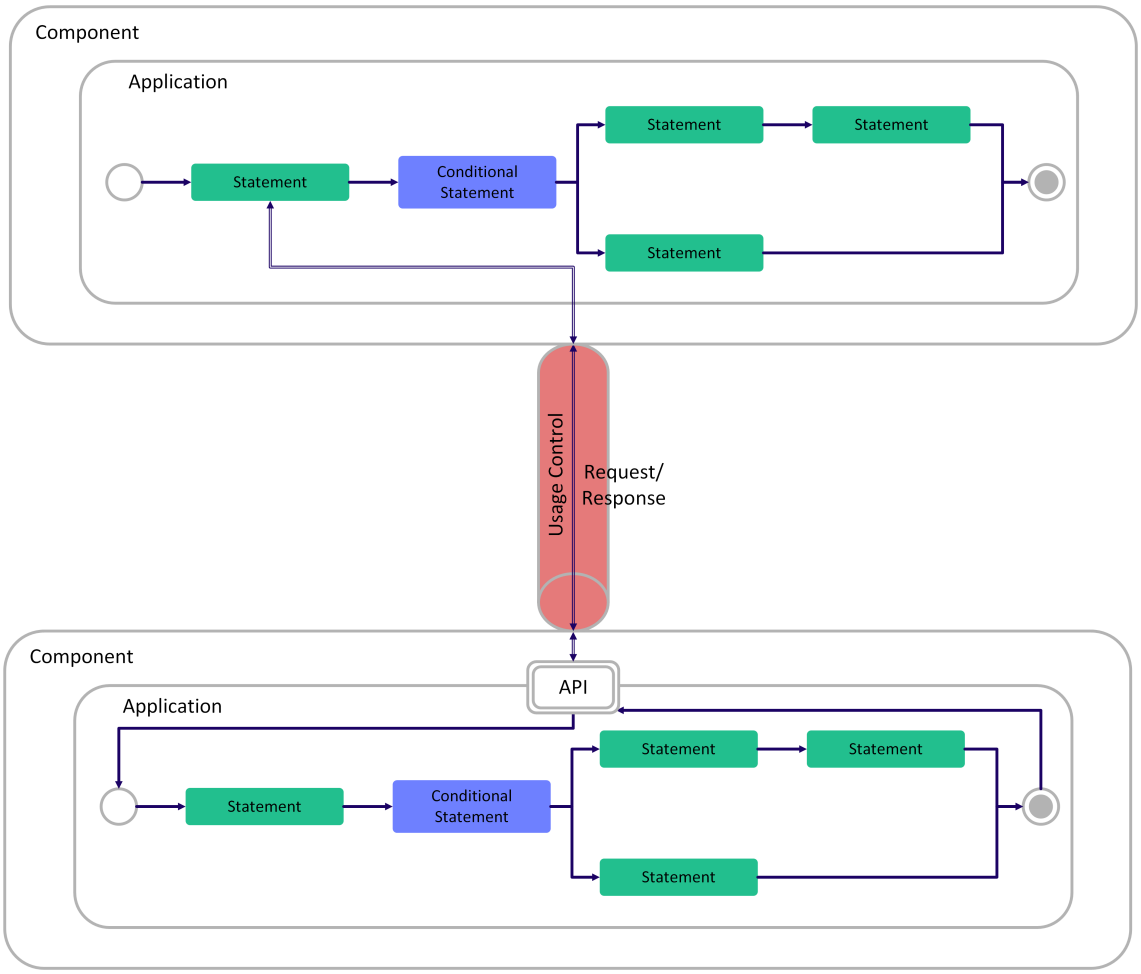

Abbildung 2.6: Schematische Darstellung, wie Usage Control Mechanismen als zentraler Bestandteil in einem System mit mehreren Applikationen integriert werden kann.

2.3 Usage Control in Mobile Code Ansätzen

Das Remote Evaluation Paradigma [Sta90a; Sta90b] aus dem Bereich der Code Mobility [Fug98] kann einen wichtigen Beitrag zur Wahrung der eigenen digitalen Souveränität leisten, da die zu verarbeitenden Daten nicht die eigenen Systeme verlassen müssen. Dies bewirkt allerdings nicht, dass die Verwendung von Usage Control Mechanismen obsolet wird. Die bestehenden Schutzbedürfnisse an den zu verarbeitenden Daten werden durch Remote Evaluation nicht adressiert. Stattdessen ergeben sich zusätzliche Schutzbedürfnisse, welche nicht den zu verarbeitenden Daten entspringen, sondern dem ausführenden System. Dies liegt darin begründet, dass potentiell fremde Applikationen, welche von Dritten bereitgestellt werden, auf den eigenen Systemen, welche die zu verarbeitenden Daten beinhalten, ausgeführt werden. Aus diesem Grund ist es für den Rechteinhaber bzw. Systembetreiber erstrebenswert die ausgeführten Applikationen in ihrer Funktionalität einzuschränken, um schadhaftes Verhalten zu verhindern. Für diese zusätzlichen Schutzbedürfnisse können

auch Usage Control Mechanismen verwendet werden, sofern die Mächtigkeit der jeweiligen Lösung es erlaubt die notwendigen Richtlinien zu definieren.

Zu diesem Zweck können die zuvor vorgestellten Usage Control Mechanismen verwendet werden. Dabei muss beachtet werden, dass die Verwendung von Usage Control Mechanismen nicht vor bösartiger Software mit versteckten Funktionalitäten schützen kann. Möchte die Partei, welche die Applikation für die Remote Evaluation bereitstellt, Funktionalitäten in der Applikation verstecken, um Schäden zu verursachen oder Daten abzuschöpfen, würden auf den entsprechenden Ausführungspfaden keine Usage Control Mechanismen integriert bzw. in der Konfiguration der externen Usage Control Lösung ausgelassen werden. Usage Control Mechanismen können im Kontext von Remote Evaluation dazu verwendet werden, um die Schutzbedürfnisse der zu verarbeitenden Daten zu erfüllen. Darüber hinaus kann, abhängig von der Mächtigkeit des Usage Control Mechanismus, sichergestellt werden, dass die Applikation sich gemäß der Vereinbarung verhält und unbeabsichtigtes Fehlverhalten durch entsprechende Richtlinien entdeckt werden.

Die vorliegende Arbeit beschreibt das Remote Processing System, welches das Remote Evaluation Paradigma als Grundlage für die Ausführung von Applikationen, welche mit D° entwickelt wurden, verwendet. Es erlaubt die Übertragung, Übersetzung und Ausführung von D°-Applikationen auf entfernten Systemen und liefert anschließend die Berechnungsergebnisse zurück.

Abseits der Möglichkeit existierende Usage Control Mechanismen in Kombination mit Remote Evaluation zu verwenden, sind dem Autor keine Arbeiten bekannt, die spezielle Mechanismen für den Einsatz in Umgebungen, welche auf Codemobilität setzen, verwenden. Es existieren Arbeiten, welche die Sicherheit von Codemobilität im Allgemeinen adressieren [Bro04; Che98; Dea99; Nil08; Rub98]. Darüber hinaus gibt es Arbeiten, welche die Sicherheit der Systeme, welche den mobilen Code ausführen, betrachten und stärken. Dies wird von Remote Processing zumindest in Teilen ebenfalls adressiert.

Bei NOMADS handelt es sich um eine Kombination aus einer speziellen Ausführungsumgebung für Agenten und einer angepassten Java VM, welche eine starke Mobilität der Agenten gewährleistet und die Ressourcen, welche die Agenten verwenden, beschränken kann [Sur00]. Starke Mobilität des Codes, welche es erlaubt laufende Agenten zu beliebigen Zeitpunkten auf andere Systeme zu übertragen, ist für das Remote Processing nicht relevant. Beim Remote Processing handelt es sich nicht um ein Agenten-basiertes System, sondern um ein Remote Evaluation System. Die verfügbaren Sicherheitsmechanismen in NOMADS sind dazu geeignet die Nutzung von Hardwareressourcen einzuschränken, eignen sich aber nicht zur Umsetzung von Usage Control Richtlinien. Auf Basis von NOMADS wurde MAST entwickelt, welches sicherstellt, dass Systeme aktuelle Software verwenden, und unsichere Konfigurationen erkennen kann [Ste].

Beim DAGO-System handelt es sich ebenfalls um einen Ansatz für mobile Agenten [Fel05]. Dieser basiert darauf, dass Sicherheitsrichtlinien definiert werden, welche anschließend auf betriebssystemspezifische Konzepte abgebildet werden. Somit liegt auch bei dieser Arbeit ein Fokus auf der Sicherheit der ausführenden Systeme und nicht der zu verarbeitenden Daten. Darüber hinaus beschreibt die Arbeit ein Agenten-basiertes System.

2.4 Entwicklung domänenspezifischer Programmiersprachen

Die Entwicklung von domänenspezifischen Programmiersprache (DSLs) ist kein neues Themengebiet [Fow10]. Es existiert eine Vielzahl von Ansätzen, welche alle eigene Vor- und Nachteile bieten. Typischerweise unterscheidet man zwischen interpretierten, eingebetteten und compilierten DSLs [Van00].

Bei einer interpretierten DSL wird der Programmcode nicht von einem Compiler in ein ausführbares Format überführt. Stattdessen wird der Programmcode direkt von einem sogenannten Interpreter ausgeführt. Der Vorteil dabei ist, dass kein plattformabhängiges Kompilat erzeugt wird. Da der Programmcode zur Laufzeit zunächst interpretiert werden muss, sind interpretierte DSLs typischerweise langsamer als compilierte.

Bei eingebetteten DSLs handelt es sich um eine Erweiterung einer bestehenden Programmiersprache, welche in diesem Zusammenhang Hostsprache genannt wird. Einige General Purpose Programmiersprachen, wie beispielsweise Scala, verfügen über integrierte Mechanismen, welche die Entwicklung von eingebetteten DSLs ermöglichen [Rit17]. Der große Vorteil bei diesem Vorgehen ist, dass der Entwicklungsaufwand im Vergleich zu den anderen Ansätzen geringer ausfällt, da kein eigener Compiler bzw. Interpreter entwickelt werden muss. Ein Nachteil ist, dass die Hostsprache ggf. nicht flexibel genug ist, um die DSL optimal abbilden zu können. Des Weiteren kann bei der Verwendung einer eingebetteten DSL der gesamte Funktionsumfang der Hostsprache verwendet werden. Ob dies ein Vor- oder Nachteil ist, hängt vom jeweiligen Ziel und Einsatzgebiet der DSL ab.

Bei compilierten DSLs existiert ein Compiler, welcher den Programmcode in ein ausführbares Format übersetzt. Dabei kann die Übersetzung direkt in ein ausführbares Format oder in eine andere Programmiersprache (sog. Hostsprache) erfolgen. Die Vorteile hierbei sind die große Flexibilität bei der Gestaltung der DSL und die verbesserte Ausführungsgeschwindigkeit gegenüber interpretierten DSLs. Nachteilig zu bewerten ist, dass der Entwicklungsaufwand für diese Art von Sprachen am größten ist, da ein eigener Compiler entwickelt werden muss.

Bei D° handelt es sich um eine übersetzte Sprache, welche in eine andere Programmiersprache übersetzt wird. Dabei verwendet D° eine Hostsprache, um komplexe Usage Control Mechanismen abzubilden und eine Ausführbarkeit auf allen Systemen, auf denen die Hostsprache verwendet werden kann, zu erreichen. Die untrennbare Verbindung von Usage Control Mechanismen und Applikationslogik in der ausführbaren Applikation wird durch D° sichergestellt. In einer interpretierten Sprache ist diese Untrennbarkeit nicht gegeben, weshalb dieser Ansatz nicht für D° in Frage kommt. Da bei einer eingebetteten DSL auch auf Konstrukte und Funktionen der Hostsprache zugegriffen werden kann, ist dieser Ansatz ebenfalls ungeeignet für D°. Wäre es möglich innerhalb von D°-Applikationen beliebigen Code der Hostsprache (bspw. Scala) zu verwenden, könnten die Usage Control Mechanismen einfach ausgehebelt oder umgangen werden.

Darüber hinaus sind zwei verschiedene Ansätze zur Softwareentwicklung aus dem Bereich domänenspezifischer Sprachen entstanden. Beim Language-Oriented-Programming (LOP) handelt es sich um ein Verfahren zur Entwicklung großer Softwaresysteme. Dabei werden in einem ersten Schritt eine oder mehrere domänenspezifische Programmiersprachen entwickelt, welche maßgeschneidert für das zu entwickelnde Softwaresystem sind [Pic09]. Anschließend

wird, unter Verwendung der entwickelten Sprachen, das Softwaresystem umgesetzt. Dabei sind die Vorteile bei diesem Vorgehen unter anderem die hohe Produktivität während der Entwicklung des Softwaresystems und die Möglichkeit zur Wiederverwendung der entwickelten Programmiersprachen. Dabei handelt es sich bei den entwickelten Sprachen häufig um eingebettete Programmiersprachen, welche beispielsweise in Racket [Fel15] umgesetzt werden. D° wurde nicht im Kontext des LOP entwickelt. Stattdessen zielt D° darauf ab für die Entwicklung möglichst vieler datenverarbeitender Applikationen mit Anforderungen an die Datennutzungskontrolle verwendet werden zu können.

Der zweite Ansatz ist das Language-Driven Engineering (LDE). Das Ziel beim LDE ist die Entwicklung (graphischer) DSLs, welche vollständige, ausführbare Applikation erzeugen können. Während beim LPO die Anwender der entwickelten DSLs Programmierer sind, ist das Ziel von LDE Experten ohne Programmierkenntnisse zur Applikationsentwicklung zu befähigen [Ste19]. Eine Anwendung des LDE findet sich in DIME. Bei DIME handelt es sich um eine IDE, welche diverse graphische DSLs unterstützt und die Entwicklung von Webanwendungen erlaubt [Boß16]. Dabei existieren unterschiedliche DSLs, beispielsweise zur Definition von Benutzeroberflächen, Definition von Geschäftsmodellen durch Prozessmodelle und zur Beschreibung von Datenmodellen. Demgegenüber zielt D° darauf ab Programmierer, welche über keine bzw. nicht ausreichende Expertise im Bereich der Datennutzungskontrolle verfügen, dazu zu befähigen datenverarbeitende Applikationen mit integrierten Usage Control Mechanismen zu entwickeln. Somit handelt es sich bei D° um keine Programmiersprache, welche im Kontext des LDE entstanden ist.

2.5 Modellgetriebene Softwareentwicklung

Das Ziel der modellgetriebenen Softwareentwicklung ist es die Komplexität bei der Softwareentwicklung besser handhabbar zu machen. Zu diesem Zweck werden Modelle verwendet, welche automatisiert in ausführbare Software übersetzt werden [Bey05]. Dabei verfügen diese Modelle häufig über ein hohes Maß an Abstraktion von der technischen Umsetzung und fokussieren sich auf die Abbildung von Abläufen, Komponenten und Kommunikationswegen. Häufig verwendete Werkzeuge sind die Modelltransformation und regelbasierte Modellvalidierungen.

Bei diesen Modelltransformationen findet eine automatische Überführung von einem oder mehreren Modellen in eine Menge funktional äquivalenter Modelle statt. Dabei können die Sprachen, welche die Ein- und Ausgabemodelle beschreiben, unterschiedlich sein [Men06]. Für diese Transformationen werden Regeln benötigt, welche eine eindeutige Überführung der Eingabemodelle in die Zielmodelle erlaubt [Cza03].

Modelltransformation ist ein zentrales Werkzeug für den D°-Compiler. Der D°-Compiler führt zunächst eine Transformation des Abstrakten Syntaxbaums (AST) der D°-Applikation durch und erzeugt dabei einen AST der Applikation in der verwendeten Hostsprache. Dieser AST wird verwendet, um den Programmcode in der Hostsprache zu generieren, welcher anschließend in eine ausführbare Applikation übersetzt wird.

Bei der modellgetriebenen Softwareentwicklung ist das Ziel häufig auf der Modellierungsebene Details der technischen Umsetzung so stark wie möglich durch Abstraktion zu kapseln. Durch diese Abstraktionsebene muss sich der Anwender bei der Modellierung nur um das Abbilden des zu lösenden Problems kümmern. Die technischen Details

werden später durch die automatisierte Überführung in reale Software eingefügt. Demgegenüber verwendet D° Werkzeuge der modellgetriebenen Softwareentwicklung, um auf der technischen Ebene eine Abstraktion von Usage Control Mechanismen einzuführen. Hierdurch kann der Anwender, welcher mit D° Applikationen entwickelt, die Entwicklung ohne Berücksichtigung und Kenntnis von Usage Control Mechanismen und Richtlinien durchführen. Durch den D°-Compiler werden diese Konzepte während der Übersetzung mit der Applikation verwoben.

Mit UMLsec existiert eine Arbeit aus dem Bereich der modellgetriebenen Softwareentwicklung, welche auf die Modellierung sicherheitskritischer Systeme abzielt [Jür02b]. Dabei verwendet UMLsec die definierten Erweiterungsmechanismen der Unified Modeling Language (UML), um ein UML Profil zu definieren, welches es ermöglicht sicherheitsrelevante Informationen in Modellen auszudrücken. Mit UMLsec können unter anderem Bedrohungsszenarien und Sicherheitsanforderungen definiert und verwendete Sicherheitsmechanismen in UML Modelle integriert werden [Jür02a]. Hierdurch wird es möglich die erstellten Modelle auf Schwachstellen zu untersuchen. Auch wenn UMLsec andere Ziele adressiert und der Fokus auf der Modellierung sicherheitskritischer Systeme liegt, ist es mögliche einzelne Aspekte von Usage Control Richtlinien mit UMLsec zu modellieren. Beispielsweise kann die Umsetzung von Richtlinien modelliert werden, welche sichere Transportkanäle oder die Verschlüsselung von Daten erfordern.

2.6 Correct-by-Construction Codegenerierung

Correct-by-Construction bezeichnet einen Ansatz zur Entwicklung von nachweislich korrekten Applikationen. Dabei wird mit einer groben Spezifikation, welche schrittweise verfeinert wird, begonnen. Solange sichergestellt ist, dass jede Operation, welche zur Verfeinerung verwendet wird, korrekt ist, ist auch das Endresultat korrekt [Kou12]. Bei den Operationen, welche zur Verfeinerung verwendet werden, handelt es sich um Hoare-Tripel. Jedes dieser Tripel besteht aus einer Vor- und einer Nachbedingung, sowie einem (abstraktem) Statement, welches die Verfeinerung beschreibt.

Der D°-Compiler arbeitet während der Codegenerierung nach diesem Vorgehen. Im Rahmen dieser Codegenerierung findet eine Modelltransformation statt, welche aus dem AST des D°-Codes einen AST in der Hostsprache erzeugt. Dabei wird der D°-AST topdown betrachtet und schrittweise in einen AST in der Hostsprache überführt. Jeder Überführungsschritt entspricht einer Verfeinerung.

Für D° existiert keine formale Menge von Hoare-Tripeln, welche diese schrittweise Überführung beschreiben. Dies ist für D° nicht notwendig, da es sich bei der Eingabe für die Codegenerierung um valide ASTs handelt, welche aus der D°-Grammatik instanziiert worden sind. Die entsprechenden Vor- und Nachbedingungen, welche einschränken, an welchen Stellen welche Elemente eingesetzt werden können, finden sich bereits in dieser Grammatik. Durch diesen generativen Ansatz ist sichergestellt, dass die Usage Control Mechanismen, welche in D° verwendet werden können, in jeder Applikation korrekt verankert werden.

2.7 Aspektorientierte Programmierung

Bei aspektorientierter Programmierung handelt es sich um ein Programmierparadigma, welches die Implementation von Querschnittsthemen, wie beispielsweise Autorisierung und

Logging, vereinfacht. Dabei wird die Logik der Querschnittsthemen gesondert von der eigentlichen Applikationslogik in sog. Aspekten implementiert. Über sog. Pointcuts wird definiert an welchen Stellen des Applikationscodes welche Aspekte ausgeführt werden sollen. Auf Basis dieser Pointcuts werden die Aspekte mit dem Applikationscode verwoben. Dies kann entweder dynamisch zur Laufzeit oder statisch zur Übersetzungszeit passieren [Kes03].

Bei Usage Control handelt es sich ebenfalls um ein Querschnittsthema, welches mit aspektorientierter Programmierung adressiert werden kann. Richtlinien können durch entsprechende Pointcuts mit den entsprechenden Funktionen der Applikationslogik verwoben werden. Werden die Aspekte mit der Applikationslogik statisch verwoben, ergibt sich eine untrennbare Verbindung zwischen Applikationslogik und Usage Control Mechanismen in der ausführbaren Applikation. Gleichzeitig werden die beiden Aspekte während der Entwicklung getrennt voneinander betrachtet.

Bei der Umsetzung von D° wird die aspektorientierte Programmierung nicht verwendet. Stattdessen generiert der Compiler Code, welcher vom Aufbau ähnlich zum Ergebnis eines statischen Verwebens im Kontext der aspektorientierten Programmierung ist. Hierfür gibt es mehrere Gründe:

Zum einen setzt der D°-Compiler Codegenerierung ein, um aus einer D°-Applikation Programmcode in der verwendeten Hostsprache zu erzeugen. Dieser Code in der Hostsprache ist nur ein Zwischenergebnis, aus welchem die ausführbare Applikation erzeugt wird. Somit ist der Programmcode in der Hostsprache nicht dazu gedacht von Entwicklern betrachtet oder bearbeitet zu werden. Aus diesem Grund profitiert D° nicht von der Isolation der Usage Control Mechanismen, welche durch die aspektorientierte Programmierung im Code der Hostsprache erreicht wird. Stattdessen würde die Komplexität von D° weiter ansteigen, da eine zusätzliche Technologie verwendet wird. Somit bringt die zusätzliche Abstraktionsebene, welche durch die aspektorientierte Orientierung eingeführt wird, keinen Vorteil gegenüber der Codegenerierung, welche D° einsetzt.

Darüber hinaus sind die Usage Control Mechanismen in D° flexibel einsetzbar. Diese Flexibilität kann durch aspektorientierte Programmierung nur mit hohem Aufwand erreicht werden, kann aber durch den gewählten Ansatz der Codegenerierung einfach abgebildet werden. Beispielsweise können Funktionalitäten mehrfach in einer D°-Applikation verwendet und bei jeder Verwendung mit unterschiedlichen Richtlinien versehen werden. Aus diesen Gründen verzichtet D° auf eine Umsetzung der aspektorientierten Programmierung und verwendet stattdessen ausschließlich Codegenerierung.

KAPITEL 3

Forschungsfragen und Forschungsansatz

In diesem Kapitel wird die Forschungsmethode, welche verwendet wurde, um die Ergebnisse der vorliegenden Arbeit zu erlangen, aufgezeigt. Es wird beschrieben, wie die notwendigen Informationen und Impulse für die Entwicklung der Ergebnisse erlangt und Feedback relevanter Akteure eingeholt wurden. Darüber hinaus werden die Forschungsfragen, welche die vorliegende Arbeit motivieren, dargestellt und die jeweiligen Aspekte der Lösung umrissen. Abschließend wird erläutert, wie die einzelnen Arbeitsschritte zur Beantwortung der Forschungsfragen beitragen und wie diese zeitlich angeordnet sind.

3.1 Methodik

Die Ergebnisse, die in der vorliegenden Arbeit vorgestellt werden, wurden auf Basis einer Kombination der Prototyping-Methode [Ala84] und Design Science Research [Hev10; Hev04] entwickelt. Dabei wurde die Prototyping-Methode als Heuristik zur Erarbeitung der Ergebnisse verwendet, jedoch das strukturierte Vorgehen von Design Science Research angewandt. Abbildung 3.1 zeigt den schematischen Ablauf des Prototypings.

Nach einer initialen Analyse der Anforderungen in mehreren Iterationen wurde ein funktionaler Prototyp (D°) entwickelt, welcher die erarbeiteten Ergebnisse nutzbar macht. Um die vorgestellten Ergebnisse zu erreichen, wurden insgesamt 12 Iterationen durchlaufen. Die erste Iteration dauerte dabei vier Monate, die zweite zwei Monate, die neunte sechs Monate und alle weiteren drei Monate an. Die vorliegende Arbeit ist im Rahmen eines Forschungsprojektes, welches sich mit der Weiterentwicklung der IDS beschäftigt, entstanden. Auch die IDS befinden sich noch in Entwicklung. Aus diesem Grund war der Prototyping-Ansatz gut geeignet, um den neuen bzw. sich ändernden Anforderungen, welche sich über die Projektlaufzeit ergeben haben, zu begegnen. Dabei wurde während der gesamten Entwicklungszeit von D° ein Backlog gepflegt, welcher verwendet wurde, um Arbeiten zu priorisieren und nachfolgende Iterationen zu planen. Darüber hinaus erlaubte

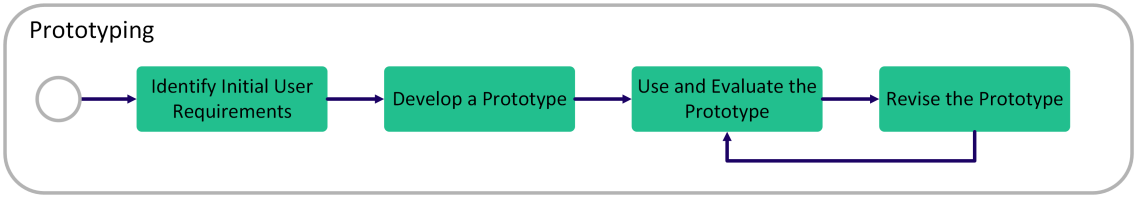

Abbildung 3.1: Schematischer Ablauf der Prototyping-Methode. Angelehnt an [Ala84, S. 559]

es die Verwendung des Prototyping-Ansatzes die entwickelten Konzepte frühzeitig im Bezug auf ihre Anwendbarkeit hin zu überprüfen, zu bewerten und ggf. anzupassen.

Zum Zwecke der Evaluation wurde im Rahmen einer Fallstudie ein Demonstrator mit D° entwickelt. Dieser dient dem Nachweis, dass die entwickelten Konzepte und Modelle praktisch einsetzbar sind und korrekt funktionieren. Darüber hinaus wird dieser Demonstrator verwendet, um Limitationen und Einschränkungen der vorgeschlagenen Lösung aufzuzeigen.

3.2 Anforderungen, Feedback & Impulse

Die Grundidee für D° ist im Kontext der IDS entstanden. Da diese sich ebenfalls noch in der Entwicklung befinden, musste während der Entwicklung von D° häufig auf neue bzw. sich ändernde Anforderungen reagiert werden. Dabei können für die Ergebnisse der vorliegenden Arbeit verschiedene relevante Quellen identifiziert werden, welche einen Einfluss auf die Entwicklung gehabt haben.

Zuallererst haben die verschiedenen Dokumente der IDS einen großen Einfluss auf die Entwicklung von D° gehabt. Diese umfassen unter anderem Anforderungen [Ott17], Architekturaspekte der IDS [Ott19a] und Informationen zu Usage Control Mechanismen und Richtlinien innerhalb der IDS [Eit21; Eit19]. Diese Dokumente hatten sowohl in der Anfangsphase, als auch während der fortlaufenden Entwicklung von D°, einen starken Einfluss auf die erzielten Ergebnisse, da sie die Rahmenbedingungen einer möglichen Einsatzumgebung von D° – nämlich den IDS – beschreiben.

Des Weiteren fanden unterschiedliche, regelmäßige Veranstaltungen statt, welche zu neuen Impulsen und Anpassungen an der Entwicklung von D° geführt haben. Über den gesamten Entwicklungszeitraum von D° fanden quartalsmäßig die sog. IDS Days statt. Dabei handelt es sich um ein kombiniertes Gesamtprojekttreffen mehrerer Forschungsprojekte, welche Arbeiten an den IDS vorantreiben. Diese wurden genutzt, um erzielte Ergebnisse zu präsentieren und zu diskutieren, wodurch fortlaufend die Kompatibilität zu den IDS sichergestellt wurde. Zusätzlich findet seit dem vierten Quartal 2018 jedes Quartal das sog. IDS Plugfest statt. Bei diesen Veranstaltungen kommen Entwickler unterschiedlichster Technologien und Komponenten, welche in den IDS eingesetzt werden sollen, zusammen. Diese stammen, anders als bei den IDS Days, nicht nur aus den beteiligten Forschungsinstituten, sondern auch aus Industrieunternehmen, welche die IDS einsetzen oder Softwarekomponenten für selbige bereitstellen wollen. Dabei werden im Rahmen der IDS Plugfests verschiedene Ziele verfolgt, welche positive Einflüsse auf die Entwicklung von D° haben:

- Präsentation von aktuellen Arbeitsständen.
- Sicherstellung der Interoperabilität einzelner Komponenten.
- Diskussion und Vorantreiben spezifischer Aspekte der IDS.

Darüber hinaus wurde während der Entwicklung an diversen Workshops und Diskussionen zu spezifischen Themen mit relevanten Akteuren – bspw. Entwicklern anderer Usage Control Mechanismen – teilgenommen.

3.3 Forschungsfragen und Lösungen

In diesem Abschnitt werden die Forschungsfragen, die in der vorliegenden Arbeit beantwortet werden, beschrieben, motiviert und detailliert betrachtet, um eine strukturelle Sicht auf die vorliegende Arbeit zu geben. Darüber hinaus wird aufgeführt, wie die einzelnen Forschungsfragen bzw. deren Aspekte durch die erarbeiteten Lösungen adressiert werden.

Das zentrale Ziel der vorliegenden Arbeit ist die Entwicklung einer Programmiersprache, welche für die Entwicklung datenverarbeitender Applikationen verwendet werden kann und Usage Control Mechanismen als festen Bestandteil der entwickelten Applikationen betrachtet. Dabei soll die Komplexität, welche mit der korrekten Verwendung von Usage Control Mechanismen und deren Integration in Applikationen einhergeht, reduziert und die verbleibende Komplexität so gekapselt werden, dass sie von Entwicklern mit der notwendigen Expertise bearbeitet werden können. Um einen zusätzlichen Beitrag im Bereich der digitalen Souveränität zu leisten, ist ein weiteres Ziel, eine Möglichkeit zu finden, wie die Kontrolle über die eigenen Daten in Szenarien, welche auf der kooperativen Nutzung der Daten basieren, gestärkt werden kann. Außerdem wird untersucht, welche Arten von Usage Control Richtlinien es gibt und wodurch sich diese unterschiedlichen Gruppen auszeichnen. Hierdurch werden sowohl einzelne Aspekte der Richtlinien, als auch Usage Control Lösungen besser miteinander vergleichbar. Wie diese Ziele erreicht wurden, wird nachfolgend für die einzelnen Forschungsfragen beantwortet.

Forschungsfrage 1 – Welche Arten von Usage Control Richtlinien gibt es und was zeichnet die unterschiedlichen Gruppen aus?

Motivation Durch die Erweiterung der klassischen Access Control Modelle, welche durch Usage Control eingeführt werden, können beliebig viele Richtlinien definiert werden. Nicht alle dieser Richtlinien lassen sich (vollständig) durch Usage Control umsetzen.

Mithilfe eines Verfahrens, welches es erlaubt Usage Control Richtlinien in Kategorien zu unterteilen, ergeben sich daraus verschiedene Vorteile. Zum einen wäre es möglich die einzelnen Kategorien hinsichtlich ihrer Umsetzungskomplexität zu beschreiben und zu vergleichen. Zum anderen könnten unterschiedliche Usage Control Mechanismen einfacher miteinander verglichen werden, indem betrachtet wird, welche Kategorien von den unterschiedlichen Lösungen umgesetzt werden können und welche nicht.

Eine Taxonomie für Usage Control Richtlinien Zu diesem Zweck wird eine Taxonomie entwickelt, welche es erlaubt Usage Control Richtlinien zu gruppieren. Dabei orientiert sich die Taxonomie am UCON_{ABC} Modell und unterscheidet zwischen Richtlinien, welche auf Conditions, Obligations oder Authorizations basieren. Für diese drei Kategorien werden verschiedene Gruppen definiert, welche spezifizieren, worauf sich eine Richtlinie dieser Gruppe bezieht.

Dabei muss beachtet werden, dass eine eindeutige Einordnung in die Taxonomie nicht für alle Usage Control Richtlinien möglich ist. Komplexe Richtlinien können durchaus Elemente aus verschiedenen Gruppen enthalten und müssen für eine Einordnung in die Taxonomie unter Umständen zerlegt werden.

Vergleichbarkeit von Usage Control Richtlinien und Mechanismen Die Taxonomie erlaubt es Gruppen von Usage Control Richtlinien miteinander zu vergleichen. Beispielsweise kann bestimmt werden, in welche Kategorie(n) eine Richtlinie gehört und damit, wie sich die Richtlinie in das UCON_{ABC} einsortiert. Außerdem kann die Komplexität von Richtlinien bestimmt werden, indem betrachtet wird in welche bzw. in wie viele Gruppen die Richtlinie einsortiert werden kann.

Auch lassen sich die unterschiedlichen Gruppen untereinander vergleichen, da alle Richtlinien innerhalb einer Gruppe ähnlich sind. Für die unterschiedlichen Gruppen kann bestimmt werden, welche Bedingungen erfüllt sein müssen, damit die enthaltenen Richtlinien technisch umgesetzt werden können.

Ebenso erlaubt es die Taxonomie unterschiedliche Usage Control Lösungen miteinander zu vergleichen. Für eine Usage Control Lösung kann bestimmt werden, welche Richtliniengruppen sie unterstützt. Bestimmt man die unterstützten Richtlinien für mehrere Usage Control Lösungen, können sie in ihrer Mächtigkeit verglichen werden. Hierdurch wird die Auswahl einer geeigneten Usage Control Lösung für individuelle Anwendungsfälle erleichtert.

Forschungsfrage 2 – Wie können erweiterbare Usage Control Mechanismen als fester Bestandteil in ein Programmiersystem integriert werden?

Motivation In Abschnitt 2.2 wurden bereits die Vorteile beschrieben, welche auftreten, wenn Usage Control Mechanismen direkt in den Programmcode von Applikationen integriert werden. Aus diesem Grund bietet es sich an, Usage Control Mechanismen direkt in eine Programmiersprache zu integrieren, welche zur Entwicklung von datenverarbeitenden Applikationen verwendet wird. Dies ermöglicht anschließend eine einfachere Integration der Mechanismen in Applikationen. Da Usage Control bei diesem Vorgehen vom Beginn der Entwicklung der Programmiersprache an berücksichtigt wird, können die Sprachelemente und die Usage Control Komponenten ideal aufeinander abgestimmt werden. Hiermit wird es möglich komfortable Methoden zur Verwendung der Usage Control Mechanismen in die Sprache zu integrieren und gleichzeitig ausdrucksstarke Usage Control Komponenten zu entwickeln.

Durch ein entsprechendes Design der Sprache kann die korrekte Nutzung von Usage Control vereinfacht werden. Beispielsweise kann für die einzelnen Sprachelemente vorgesehen werden, welche Usage Control Komponenten verwendet werden können. Anhand diese Vorgaben ist es für den Entwickler einfacher in den jeweiligen Situationen die entsprechenden Komponenten auszuwählen.

Bei diesem Vorgehen muss sichergestellt werden, dass sowohl die Programmiersprache, als auch die verwendeten Usage Control Komponenten, flexibel einsetzbar und erweiterbar sind. Dadurch wird sichergestellt, dass die Sprache für zukünftige Anforderungen und Einsatzgebiete adaptiert werden kann. Gleichzeitig wird die Programmiersprache durch die Möglichkeit, für jeden Anwendungsfall individuell passende Nutzungsbedingungen zu definieren, flexibel einsetzbar.

Feste Integration von Usage Control D° verfügt über zwei verschiedene Konzepte, welche verwendet werden, um die Datennutzungsrichtlinien technisch abzubilden und durchzusetzen. Das erste Konzept sind Constraints. Diese repräsentieren einzelne Regeln, welche atomar sind und die Logik enthalten, welche notwendig ist, um die jeweils definierte Regel zu überprüfen. Das zweite Konzept sind Policies, welche verwendet werden, um die einzelnen Constraints zu aggregieren und hierdurch komplexe Richtlinien abzubilden. Policies verfügen dabei über keine eigene Überprüfungslogik.

Diese beiden Konzepte sind ein fester Bestandteil in jeder D°-Applikation, genauso wie Definitionen und Instanzen von Datentypen und Aktivitäten. Bei Aktivitäten handelt es sich im Kontext von D° um atomare Funktionen, welche innerhalb von Applikationen verwendet werden, um die Applikationslogik abzubilden. Dabei kann sich in einer Aktivität beliebig (komplexe) Logik befinden, welche bei einem Aufruf der Aktivität ausgeführt wird. Hierdurch ist sichergestellt, dass Möglichkeiten zur Definition und Umsetzung von Richtlinien ein fester Bestandteil von D° und den Applikationen, welche mit der Programmiersprache entwickelt werden, sind.

Für einen vollständigen Usage Control Mechanismus fehlen an dieser Stelle noch Konzepte, welche sicherstellen, dass die Richtlinien, welche in Applikationen integriert sind, an den notwendigen Stellen durchgesetzt werden. Zu diesem Zweck verwendet D° die zwei Konzepte Sandbox und Security Manager. Wird eine Applikation mit D° entwickelt, besteht der Programmcode aus Konstrukten, welche aus den meisten Programmiersprachen bekannt sind, beispielsweise Sequenzen und bedingte Verzweigungen. Die ausführbare Applikation, welche vom D°-Compiler erzeugt wird, enthält komplexere Konstrukte, um die Applikationslogik abzubilden. Jeder Aktivitätsaufruf – und damit die gesamte Applikationslogik – wird nicht direkt, sondern von der Sandbox ausgeführt. Diese Sandbox stellt dabei sicher, dass Richtlinien, welche an einzelnen Elementen der Applikation (bspw. der Aktivität) angehängt sind, an den relevanten Punkten der Ausführung die Möglichkeit haben, Überprüfungen durchzuführen. Dabei stellt die Sandbox den Richtlinien auch diverse Informationen zur Verfügung, welche während der Überprüfung verwendet werden können. Durch die Sandbox ist es möglich Richtlinien zu festgelegten Zeitpunkten zu überprüfen. Dies erlaubt es beispielsweise Vorbedingungen für einzelne Aktionen zu definieren.

Der Security Manager erweitert die Mächtigkeit durch das Hinzufügen von dynamischen Überprüfungen von Richtlinien. Mithilfe des Security Managers ist es möglich bestimmte Schnittstellen zu überwachen (bspw. Datei- und Netzwerkoperationen). Wird eine der überwachten Schnittstellen verwendet, unterbricht der Security Manager die Ausführung und ermöglicht es den Richtlinien Überprüfungen durchzuführen. Hierdurch wird die Verwendung von Richtlinien ermöglicht, welche nur ausgewertet werden, wenn innerhalb der Logik einer Applikation bestimmte Operationen verwendet werden.

Durch die Gesamtheit der beschriebenen Konzepte entsteht ein Usage Control Mechanismus, welcher fest in D° und Applikationen, welche mit D° entwickelt werden, verankert ist. Dabei bildet der Mechanismus sowohl die Definition von Richtlinien und deren Umsetzungslogik, als auch die feste Integration in Applikationen ab. Bei der Beschreibung der Konzepte offenbaren sich zwei Ebenen.

Zum einen gibt es die Entwicklungsebene, auf welcher die Entwicklung von Applikationen mit D° stattfindet. Auf dieser Ebene werden die einzelnen Sprachelemente von D°

verwendet und komponiert, um Applikationslogik zu definieren. Zum anderen existiert die Ausführungsebene. Auf dieser Ebene wird die Sandbox verwendet, um die Logik der Aktivitäten, welche in der Applikation verwendet werden, in der korrekten Reihenfolge auszuführen. Gleichzeitig stellen die Sandbox und der Security Manager sicher, dass die Überprüfungslogik der verwendeten Richtlinien an den notwendigen Stellen ausgeführt wird.

Es wird deutlich, dass auf der Ausführungsebene zusätzliche Komponenten verwendet werden, welche auf der Entwicklungsebene nicht existieren. Hierdurch verfügt die Ausführungsebene über zusätzliche Komplexität, welche vom D°-Compiler adressiert werden muss. Um diese Komplexität ein Stück weit zu reduzieren, wird D°-Programmcode nicht direkt in ein ausführbares Format übersetzt. Stattdessen verwendet D° eine Hostsprache. Der D°-Compiler übersetzt den D°-Programmcode zunächst in Programmcode der verwendeten Hostsprache, welcher anschließend in ein ausführbares Format übersetzt wird. Das Ergebnis dieses Übersetzungsvorgangs ist eine ausführbare Applikation, welche die Applikationslogik und die Usage Control Mechanismen als feste Bestandteile beinhaltet.

Die Möglichkeit D° als eine interpretierte Sprache zu entwickeln wurde ausgeschlossen, da hierbei keine untrennbare Verbindung von Applikationslogik und Usage Control Mechanismen erreicht wird. Durch Manipulation des Interpreters kann die Überprüfung von Richtlinien verhindert und somit der gesamte Usage Control Mechanismus umgangen werden.

Flexibilität des Usage Control Mechanismus Die von D° adressierte Domäne der Datenverarbeitung ist sehr breit und allgemein. Dies führt dazu, dass verschiedene Fachdomänen teils stark unterschiedliche Anforderungen in diesem Bereich haben. Dies betrifft insbesondere auch die Datennutzungskontrolle. Um D° für eine Vielzahl von verschiedenen Szenarien nutzbar zu machen, wurde der Usage Control Mechanismus möglichst flexibel gestaltet.

Ein wichtiger Aspekt dieser Flexibilität ist die Möglichkeit die Überprüfungen von Richtlinien an verschiedenen Stellen durchzuführen und dabei unterschiedliche Daten verwenden zu können. Zu diesem Zweck verfügen D°-Richtlinien über drei getrennte Überprüfungspunkte, welche zu unterschiedlichen Zeitpunkten ausgeführt werden:

- Vor einer auszuführenden Aktion (bspw. Aktivitätsausführung und Ausführung der Applikation).
- Während der Ausführung einer Aktivität, wenn bestimmte Operationen verwendet werden (bspw. Datei- und Netzwerkoperationen).
- Nach einer ausgeführten Aktion.

Während der Ausführung der Überprüfungslogik können die Richtlinien auf verschiedene Informationsquellen zugreifen, welche hilfreiche Daten bereitstellen können:

- Eingabeparameter, welche an Aktivitäten übergeben und von der jeweiligen Richtlinie adressiert werden.
- Ein erweiterbares System für Kontextinformationen, welches beispielsweise Informationen über den aufrufenden Nutzer enthält.

- Tags, welche an Aktivitäten und Daten hängen können und dazu verwendet werden geplante Aktivitäten bzw. den Status von Elementen zu beschreiben.
- Beliebige interne Daten, welche innerhalb der Richtlinie gespeichert und verwaltet werden.

Es ist auch möglich D° in Umgebungen zu betreiben, welche unterschiedliche Usage Control Mechanismen verwenden oder den Austausch von Daten zwischen verschiedenen Usage Control Instanzen Mechanismen erfordern. In diesen Fällen bietet es sich an einen zentralen Service zu betreiben, welcher diese gemeinsam genutzten Usage Control Daten verwaltet. Im Kontext von XACML wäre dies beispielsweise der PIP. Steht ein solcher Service zur Verfügung, kann er ebenfalls innerhalb der Richtlinien verwendet werden, ist aber kein Bestandteil von D° oder der Applikation.

Ein weiterer Aspekt, welcher einen positiven Einfluss auf die Flexibilität des Usage Control Mechanismus hat, ist die Möglichkeit individuelle Richtlinien durch die Verwendung von D°-Policies zu erzeugen. Durch die Komposition von Constraints und bereits verfügbaren Policies können für jeden Anwendungsfall individuelle und maßgeschneiderte Richtlinien definiert werden. Darüber hinaus können Sprachelemente (beispielsweise Aktivitäten) beliebig mit Richtlinien verknüpft werden, um für jede Applikation eine individuell angepasste Menge von Richtlinien bereitzustellen.

Erweiterbarkeit des Usage Control Mechanismus Die Erweiterbarkeit des Usage Control Mechanismus ist eng verbunden mit der Flexibilität, da sie gewährleistet, dass D° angemessene Richtlinien für eine Vielzahl von Anwendungsfällen bereitstellen kann. Erweiterbarkeit ist ein wichtiges Attribut für den Usage Control Mechanismus von D°. Anforderungen an die Datennutzungskontrolle wandeln sich im Laufe der Zeit und individuelle Anwendungsfälle können immer Richtlinien erfordern, welche nicht im Usage Control Mechanismus enthalten sind. Nur über die Erweiterbarkeit kann gewährleistet werden, dass D°, in einem gewissen Rahmen, für diese Situationen angepasst und verwendet werden kann.

Es ist Entwicklern möglich neue Richtlinien in Form von Constraints und Policies zu D° hinzuzufügen. Dabei muss der Entwickler eine Definition des jeweiligen Sprachelements liefern, welches unter anderem die Eingabeparameter der Richtlinie enthält. Im Falle von Constraints muss darüber hinaus noch die eigentliche Überprüfungslogik mitgeliefert werden. Diese zusätzlichen Richtlinien sind in Gruppen organisiert und werden in Form von Libraries der verwendeten Hostsprache ausgeliefert. Hierdurch wird es möglich verschiedene solcher Erweiterungen parallel zu verwenden und selbige auf einfache Art und Weise zu teilen.

Analog dazu ist es möglich verschiedene andere Systeme innerhalb von D° zu erweitern. Dies umfasst Datentypen, Aktivitäten und die zuvor genannten Kontextinformationen. Damit ist sichergestellt, dass sowohl der Usage Control Mechanismus von D°, als auch D° selber, erweiterbar ist und an sich wandelnde Anforderungen angepasst werden kann.

Forschungsfrage 3 – Wie kann die Komplexität der korrekten Anwendung von Usage Control Mechanismen gekapselt werden, damit sie nicht Aufgabe des Applikationsentwicklers ist?

Motivation In Abschnitt 2.2 wurde erläutert, dass die korrekte Verwendung von Usage Control Mechanismen schnell eine beachtliche Komplexität erreicht und hierdurch fehleranfällig ist. Diese Komplexität kann zu einem gewissen Grad reduziert werden, indem die zuvor erwähnte Programmiersprache verwendet wird.

Dennoch verbleibt ein hohes Maß an Komplexität, die von den Entwicklern gehandhabt werden muss. Aus diesem Grund ist ein Ziel dieser Arbeit eine saubere Trennung zwischen der Applikationsentwicklung und der Verwendung von Usage Control Mechanismen direkt in der Programmiersprache.

Das Ergebnis dieser Trennung ist, dass es für den Entwickler keine Rolle spielt, ob in der entwickelten Applikation Nutzungsbedingungen definiert und umgesetzt werden. Idealerweise kann der Entwickler dem Programmcode nicht einmal entnehmen, ob die finale Applikation später mit Nutzungsbedingungen ausgestattet ist, oder nicht. Da für den Entwickler kein Unterschied erkennbar ist, wird die Komplexität der Usage Control Mechanismen in eine separate Phase der Applikationsentwicklung gekapselt. Diese kann von Entwicklern mit der entsprechenden Expertise durchgeführt werden und somit das Fehlerrisiko senken. D° verfügt sowohl auf der Ebene der Applikationsentwicklung, als auch der Übersetzung von Applikationen über Mechanismen, um diese Komplexität handhabbar zu machen.

Kapselung auf Entwicklungsebene Die erste Hürde im Bezug auf die Komplexität wird bei der Entwicklung von Applikationen mit D° sichtbar: Der Entwickler muss sicherstellen, dass die notwendigen Richtlinien an allen notwendigen Stellen im Programmcode verwendet werden, da die Datennutzungskontrolle ansonsten lückenhaft ist und somit nicht den angedachten Zweck erfüllt.

Um hier eine Vereinfachung zu schaffen, verwendet D° das Programmierparadigma der policy-agnostischen Programmierung. Dieses entspringt ursprünglich dem Gebiet der Informationsflusskontrolle und wird in D° für die Obermenge der Datennutzungskontrolle verwendet. Dabei basiert das Paradigma auf der Trennung von Datennutzungskontrolle und Applikationslogik während der Entwicklung und der Kombination dieser Elemente in einer späteren Phase des Applikationslebenszyklus.

Zu diesem Zweck verwendet D° für Sprachelemente (beispielsweise Aktivitäten) eine Trennung zwischen Definitionen und Instanzen. Dabei sind Definitionen nicht direkt in Applikationen verwendbar. Die Erzeugung von Instanzen erfolgt nicht im Programmcode, sondern auf der selben Ebene wie die Erstellung neuer Sprachelemente. Dabei erlaubt die Erzeugung von Instanzen unter anderem die Verknüpfung von Elementen mit Richtlinien, welche bei der Verwendung des Elements umgesetzt werden müssen. Während der Entwicklung von Applikationen mit D° werden diese Instanzen im Programmcode verwendet. Der Entwickler muss dabei jedoch keine Kenntnisse über die zusätzlichen Inhalte der Instanz haben. Es müssen nur die Informationen aus der Definition (beispielsweise Ein- und Ausgabeparameter einer Aktivität) vorhanden sein, um die Elemente korrekt in Applikationen verwenden zu können.

Diese Trennung bietet mehrere Vorteile. Zum einen bildet D°-Programmcode ausschließlich die Applikationslogik ab. Der Programmcode ist somit nicht mit Usage Control

Konstrukten angereichert, was sich positiv auf die Lesbarkeit des Programmcodes auswirkt. Aufgrund dessen kann es dem Programmcode nicht angesehen werden, ob die einzelnen verwendeten Elemente mit Usage Control Richtlinien verknüpft sind. Für den Entwickler spielt dies auch keine Rolle und er kann sich vollständig auf die Entwicklung der Applikationslogik fokussieren.

Zum anderen kann die Verknüpfung von Richtlinien mit anderen Sprachelementen gebündelt an einer Stelle, abseits des Programmcodes, erfolgen. Diese Verknüpfung kann von Spezialisten oder Entwicklern mit entsprechenden Kenntnissen vorgenommen werden. Dabei wird die Aufgabentrennung durch die Trennung von Applikationslogik und Verknüpfung mit Richtlinien unterstützt.

Durch die Umsetzung des Programmierparadigmas der policy-agnostischen Programmierung in D° ist somit sichergestellt, dass die Komplexität der Verwendung von Usage Control in Applikationen sich nicht in der Applikationslogik und -entwicklung widerspiegelt. Stattdessen wird diese Komplexität in die Verknüpfung von Richtlinien mit anderen Sprachelementen ausgelagert. Dies resultiert darin, dass es für Entwickler möglich ist D° vollständig ohne Kenntnis über die enthaltenen Usage Control Mechanismen zu verwenden.

Automatisierte Verknüpfung durch Codegenerierung Die Komplexität, welche durch das Programmierparadigma der policy-agnostischen Programmierung auf der Entwicklungsebene gekapselt und deshalb isoliert betrachtet werden kann, muss während einer späteren Phase der Applikationsentwicklung aufgelöst werden. Es muss sichergestellt werden, dass die Applikationslogik und die Richtlinien, welche mit den verwendeten Sprachelementen verknüpft sind, vor der Ausführung so miteinander verwoben werden, dass eine Ausführung der Richtlinien zu den korrekten Zeitpunkten sichergestellt ist. Somit hat sich die Komplexität der korrekten Anwendung des Usage Control Mechanismus nur verlagert.

D° verwendet Codegenerierung, um diese Komplexität zu adressieren. Dieses Vorgehen bietet einige Vorteile. Zuallererst fügt es sich sehr gut in das Gesamtkonzept von D° ein. Denn der D°-Compiler führt bei der Übersetzung zunächst eine Transformation in eine Hostsprache durch, welche anschließend in ein ausführbares Format übersetzt wird. Durch entsprechende Gestaltung der Codegenerierung können die Konzepte, welche D° für Usage Control verwendet, sowie die notwendigen Überprüfungen der Richtlinien in diese Transformation mit einbezogen werden.

Bei diesem Vorgehen wird die Applikationslogik mit der Datennutzungskontrolle verwoben, bevor eine ausführbare Applikation erzeugt wird. Hierdurch ist sichergestellt, dass die Usage Control Mechanismen ein fester Bestandteil der finalen Applikation sind, welche untrennbar mit der Applikationslogik verwoben sind.

Darüber hinaus werden durch die Automatisierung der Verknüpfung von Applikationslogik und Datennutzungskontrolle menschliche Fehler bei diesem Schritt vermieden. Dies betrifft sowohl versehentliche als auch absichtliche Fehler. Außerdem bewirkt diese Automatisierung, dass Entwickler, welche mit D° Applikationen entwickeln, abseits der Definition von Richtlinien und der Verknüpfung mit anderen Sprachelementen, keine Berührungspunkte mit den verwendeten Usage Control Mechanismen haben.

Forschungsfrage 4 – Durch welche Designentscheidungen können Usage Control Mechanismen bei der Erlangung der Datensouveränität unterstützt werden?

Motivation Die Verwendung von Usage Control Mechanismen verhindert nicht, dass Daten in Szenarien, welche auf kooperative Datennutzung setzen, an Dritte übertragen werden müssen. Dies kann ein Hindernis sein, da beispielsweise rechtliche Vorgaben diese Datenweitergabe verbieten. Diese Hindernisse können häufig adressiert werden, indem eine Möglichkeit gefunden wird, bei der die Daten während der kooperativen Datennutzung auf den Systemen des Rechteinhabers verbleiben.

Falls das Hindernis nur das mangelnde Vertrauen des Rechteinhabers in den Datennutzer ist, kann ein treuhänderischer Ansatz eine Lösung darstellen. Bei einem solchen Ansatz werden sowohl Daten, als auch datenverarbeitende Applikation an einen Treuhänder übergeben, welcher die Datenverarbeitung ausführt und das Berechnungsergebnis an den Datennutzer weitergibt. Für andere Hindernisse ist dieser Ansatz nicht immer eine gangbare Lösung. Aus diesem Grund wird ein Verfahren entwickelt, welches eine kooperative Datennutzung erlaubt, bei der die Daten auf den Systemen des Rechteinhabers verbleiben und gleichzeitig die Usage Control Mechanismen von D° verwendet werden können.

Remote Evaluation für D°-Applikationen Auf Basis des Remote Evaluation Paradigmas wird ein System entwickelt, welches die entfernte Ausführung von D°-Applikationen erlaubt. Dabei entspringt das Remote Evaluation Paradigma dem Forschungsbereich der Codemobilität. Dieses System wird Remote Processing genannt. Der Betreiber des Remote Processing (und Rechteinhaber der zu verarbeitenden Daten) stellt genau die Daten zur Verfügung, welche durch die Applikation verarbeitet werden dürfen. Der Bereitsteller der Applikation kontrolliert dabei die Applikation und bestimmt somit den Zeitpunkt der Applikationsausführung.

Durch diesen Prozess wird es möglich Daten durch Dritte auf den eigenen Systemen verarbeiten zu lassen und gleichzeitig Datennutzungskontrolle für die verarbeiteten Daten durchzusetzen. Für einen Einsatz in realen Umgebungen reicht dieses Vorgehen allerdings nicht aus. Die ungeprüfte Ausführung fremder Software auf den eigenen Systemen birgt erhebliche Sicherheitsrisiken. Dies betrifft nicht nur die Daten, welche zur Verfügung gestellt werden, sondern auch das ausführende System selbst.

Zusätzliche Sicherheitsmaßnahmen Aus diesem Grund enthält der Remote Processing diverse Mechanismen, welche Risiken minimieren und die Sicherheit des Betreibers stärken. Remote Processing erlaubt zwar die Ausführung von D°-Applikationen auf entfernten Systemen, tauscht diese Applikationen aber nicht direkt aus. Stattdessen werden der Code der Applikation und alle weiteren Ressourcen (beispielsweise Aktivitäts- und Richtliniendefinitionen), welche notwendig sind, um die Applikation zu bauen, an den Betreiber des Remote Processings geliefert. Der Betreiber muss den Programmcode zunächst selber in eine ausführbare Applikation übersetzten, bevor der Bereitsteller der Applikation diese verwenden kann.

Dabei hat der Betreiber des Remote Processing die Möglichkeit vor der Übersetzung der Applikation ein Audit durchzuführen. Das Ziel dieses Audits ist sicherzustellen, dass die notwendigen (und vereinbarten) Richtlinien an den relevanten Stellen der Applikation verwendet werden. Die richtige Umsetzung der Datennutzungskontrolle innerhalb der Applikation muss dabei nicht überprüft werden, da sie vom D°-Compiler automatisch sichergestellt wird.

Nach der erfolgreichen Übersetzung der Applikation in ein ausführbares Programm muss die Applikation zur Ausführung vorbereitet werden. Dabei wird die Applikation nicht direkt auf dem Remote Processing System ausgeführt. Stattdessen werden Applikationscontainer verwendet, um die Applikationen vom ausführenden System zu isolieren. Darüber hinaus erlaubt die Verwendung von Applikationscontainern die Definition von Einschränkungen für die ausgeführte Applikation. Beispielsweise können die verfügbaren Ressourcen (z.B. RAM und CPU) beschränkt werden. Außerdem kann durch die Konfiguration des Applikationscontainers sichergestellt werden, dass die Applikation nur Zugriff auf die Daten hat, welche vereinbart worden sind.

Sobald der Applikationscontainer erstellt ist bestimmt der Bereitsteller der Applikation, wann dieser gestartet und gestoppt wird, sowie wann die Applikation mit welchen Parametern aufgerufen wird. Dabei hat der Bereitsteller der Applikation keinen direkten Zugriff auf die ausgeführte Applikation. Stattdessen müssen alle Aufrufe an das Remote Processing System erfolgen. Gültige Aufrufe werden vom System anschließend an die eigentliche Applikation weitergeleitet. Nach der erfolgreichen Ausführung der Applikation kann der Bereitsteller der Applikation die Ausführungsergebnisse beim Remote Processing System abfragen.

3.4 Iterative Arbeitsergebnisse

Wie eingangs bereits beschrieben, wurde die Entwicklung von D° in 13 einzelnen Iterationen durchgeführt, welche im Mittel eine Laufzeit von jeweils dreieinhalb Monaten hatten. Dabei hatte jede einzelne Iteration ein eigenes zentrales Thema, welches bearbeitet wurde. Abbildung 3.2 zeigt einen Zeitstrahl, welche die einzelnen Iterationen und ihr jeweils zentrales Thema aufzeigen. Für jede Iteration wird angegeben, wie viele Monate nach Start der Entwicklung sie abgeschlossen war. Beispielsweise bedeutet die Beschriftung M9 eines Knotens, dass dieser Schritt 9 Monate nach Beginn der Entwicklung erreicht wurde. Es ist erkennbar, dass die gesamte Entwicklungszeit für die Ergebnisse der vorliegenden Arbeit dreieinhalb Jahre umfasst. Nachfolgend werden die einzelnen Iterationen grob umrissen, um eine bessere Einordnung zu ermöglichen.

Anforderungsanalyse Auf Basis der zu dem Zeitpunkt verfügbaren Dokumente der IDS und einer Literaturrecherche wurden Anforderungen für D° abgeleitet. Darüber hinaus wurde eine Evaluation verschiedener Technologien, welche für die Implementation von D° in Frage kamen, durchgeführt.

Grammatik v1 In dieser Iteration wurde die erste Version der Grammatik von D° entwickelt. Dazu wurde im Vorfeld erarbeitet, welche Form die Applikationen, welche mit D° entwickelt

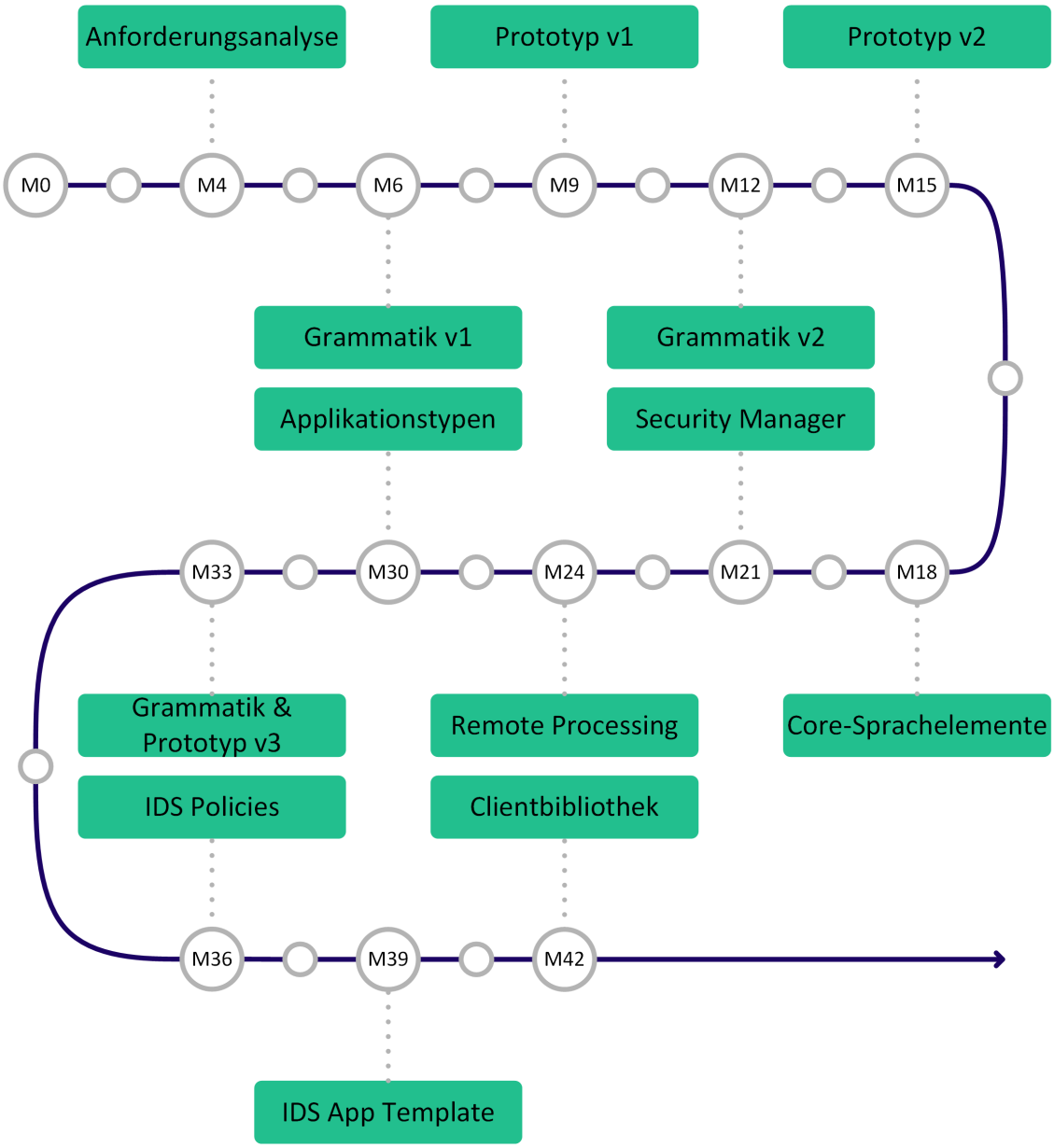

Abbildung 3.2: Zeitstrahl, welcher erarbeitete Ergebnisse der vorliegenden Arbeit im Laufe der Zeit darstellt

werden, haben sollen und welche Sprachkonstrukte diese unterstützen sollen. Außerdem wurden wichtige Entscheidungen getroffen, beispielsweise die Verwendung einer Hostsprache und die Umsetzung des Programmierparadigmas der policy-agnostischen Programmierung.

Prototyp v1 Auf Basis der zuvor durchgeführten Evaluation von Technologien wurde die erste Version der Grammatik in Form eines funktionalen Prototyps implementiert. Durch einfache Experimente wurde die Nutzbarkeit des entstandenen Prototyps bewertet. Des Weiteren wurde festgelegt, dass der entwickelte Prototyp Java als Hostsprache verwendet.

Grammatik v2 Auf Grundlage der Experimente, welche in der vorherigen Iteration durchgeführt wurden, konnten einige Punkte identifiziert werden, welche an der Grammatik verbessert werden können, um eine bessere Nutzbarkeit zu ermöglichen. Diese Verbesserungen wurden, gemeinsam mit zusätzlichen Konstrukten, in eine überarbeitete Version der Grammatik eingearbeitet.

Prototyp v2 Der funktionale Prototyp wurde entsprechend der neuen Grammatikversion überarbeitet, damit stets der aktuelle Entwicklungsstand getestet werden kann.

Core-Sprachelemente Im Auslieferungszustand verfügt D° über keine Datentypen, Aktivitäten und Richtlinien. Um Experimente und Tests zu vereinfachen, wurde eine kleine Auswahl dieser Elemente implementiert und kann als Kern für D°-Applikationen verwendet werden.

Security Manager Die bisher im Prototyp verstreuten Usage Control Mechanismen wurden erweitert und in den Konzepten Security Manager und Sandbox gebündelt. Diese beiden Konzepte sind die zentralen Usage Control Komponenten von D°.

Remote Processing Das Remote Processing System wurde entwickelt. Dieses erlaubt die Übertragung von D°-Code an Dritte, die entfernte Übersetzung des Programmcodes in ausführbare Applikationen, die entfernte und durch Application Container isolierte Ausführung der Applikation und die anschließende Rücklieferung der Berechnungsergebnisse.

Applikationstypen Um D° für ein breiteres Spektrum an Einsatzszenarien nutzbar zu machen, wurde zum Prototypen die Möglichkeit hinzugefügt Applikationen mit unterschiedlichen Schnittstellen zu erzeugen (bspw. CLI und HTTP). Der bisherige Entwicklungsstand erlaubte nur die Übersetzung von Applikationen mit einer HTTP-Schnittstelle.

Grammatik & Prototyp v3 Fortlaufende Experimente mit dem Prototypen und neue Funktionalitäten der Sprache ermöglichten eine Anpassung und Erweiterung der Sprache. Darüber hinaus wurde der Prototyp ebenfalls, entsprechend der neuen Grammatik, überarbeitet.

IDS Policies In den IDS wurde eine feste Menge von Nutzungsrichtlinien verabschiedet, zu deren Umsetzung die einzelnen Usage Control Mechanismen Stellung beziehen müssen. Zu diesem Zweck wurden diese Richtlinien untersucht und (soweit möglich) in D° implementiert. Dies hat keine Auswirkungen auf die Möglichkeit zusätzliche Richtlinien innerhalb von D° zu verwenden, sondern stellt nur eine Anwendung der bereits entwickelten Konzepte dar.

IDS App Template Der App Store ist eine zentrale Komponente innerhalb der IDS, welche es den Teilnehmern erlaubt IDS konforme Applikationen zu beziehen bzw. bereitzustellen. Im Rahmen der Entwicklung dieses App Stores ist ein Template für IDS-Applikationen (sog. Data Apps) entstanden. Es fand eine Evaluation dieses Templates statt, die im Nachgang in einem neuen Applikationstypen resultierte, welcher mit D° erzeugt werden kann.

Clientbibliothek D° verwendet intern ein Typsystem, welches Nukleus genannt wird. Dies hat zur Folge, dass D°-Applikationen als Eingabe Daten im Nukleus-Format erwarten und entsprechend zurückliefern. Dies ist innerhalb eines Prozesses, welcher nur aus D°-Applikationen besteht, wenig problematisch, sorgt jedoch für zusätzliche Komplexität an Systemgrenzen zu anderen Arten von Applikationen und graphischen Nutzeroberflächen. Aus diesem Grund wurde eine Clientbibliothek für D° entwickelt und als fester Bestandteil in jede D°-Applikation integriert. Diese ermöglicht D°-Applikationen mit einfachen JSON-Daten aufzurufen und entsprechende Rückgabewerte zu erhalten.

KAPITEL 4

Beitrag der Arbeit

Die in der vorliegenden Arbeit vorgestellten Ergebnisse sind in mehreren wissenschaftlichen Beiträgen veröffentlicht worden. In diesem Kapitel wird die Gesamtlösung im Überblick dargestellt. Anschließend werden die Beiträge der einzelnen Veröffentlichungen zum Gesamtergebnis der vorliegenden Arbeit aufgezeigt.

4.1 Die Gesamtlösung im Überblick

In den Abschnitten 3.3 und 3.4 wurden bereits diverse Aspekte der erarbeiteten Ergebnisse benannt und stellenweise grob umrissen. In diesem Abschnitt werden die Arbeitsergebnisse strukturiert und ihrer Gesamtheit präsentiert. Dabei wird zunächst D° vorgestellt. Die Beschreibungen umfassen den Aufbau des Compilers, der erzeugten Applikationen und relevante Prozesse innerhalb der Komponenten. Anschließend wird das Remote Processing beschrieben. Auch hier wird ein Überblick über die Architektur und relevante Konzepte gegeben. Dabei sollen die Beschreibungen nur einen Überblick über die Gesamtergebnisse der vorliegenden Arbeit geben. Aus diesem Grund gehen diese nicht in die Tiefe und umfassen nicht jeden einzelnen Aspekt der Lösung.

Im Anschluss an diese Beschreibungen wird eine Taxonomie für Usage Control Richtlinien vorgestellt. Dabei wird der Aufbau und Verwendungszweck der Taxonomie beschrieben. Bei der Taxonomie handelt es sich ebenfalls um ein Ergebnis der vorliegenden Arbeit, welches die vierte Forschungsfrage adressiert. Abweichend zu den anderen beschriebenen Lösungsaspekten existiert keine wissenschaftliche Veröffentlichung zu diesem Aspekt der Gesamtlösung.

4.1.1 Die Programmiersprache D°

In diesem Abschnitt wird ein Überblick über die Programmiersprache D° gegeben. Abbildung 4.1 zeigt eine schematische Darstellung des Aufbaus von D° und mit D° entwickelter Applikationen (sog. Data Apps). Darüber hinaus sind wichtige Beziehungen zwischen den einzelnen Komponenten dargestellt.

Für die nachfolgenden Beschreibung von D° wird das Gesamtergebnis in Laufzeitaspekte und Entwicklungsaspekte unterteilt, welche nachfolgend getrennt voneinander betrachtet werden. Dabei umfassen die Entwicklungsaspekte, neben der Grammatik von D°, die erweiterbaren Subsysteme von D°, die Struktur von D°-Projekten und den Compiler. Somit fallen alle Aspekte, welche von Entwicklern bei der Erstellung von D°-Applikationen verwendet werden, in diesen Bereich.

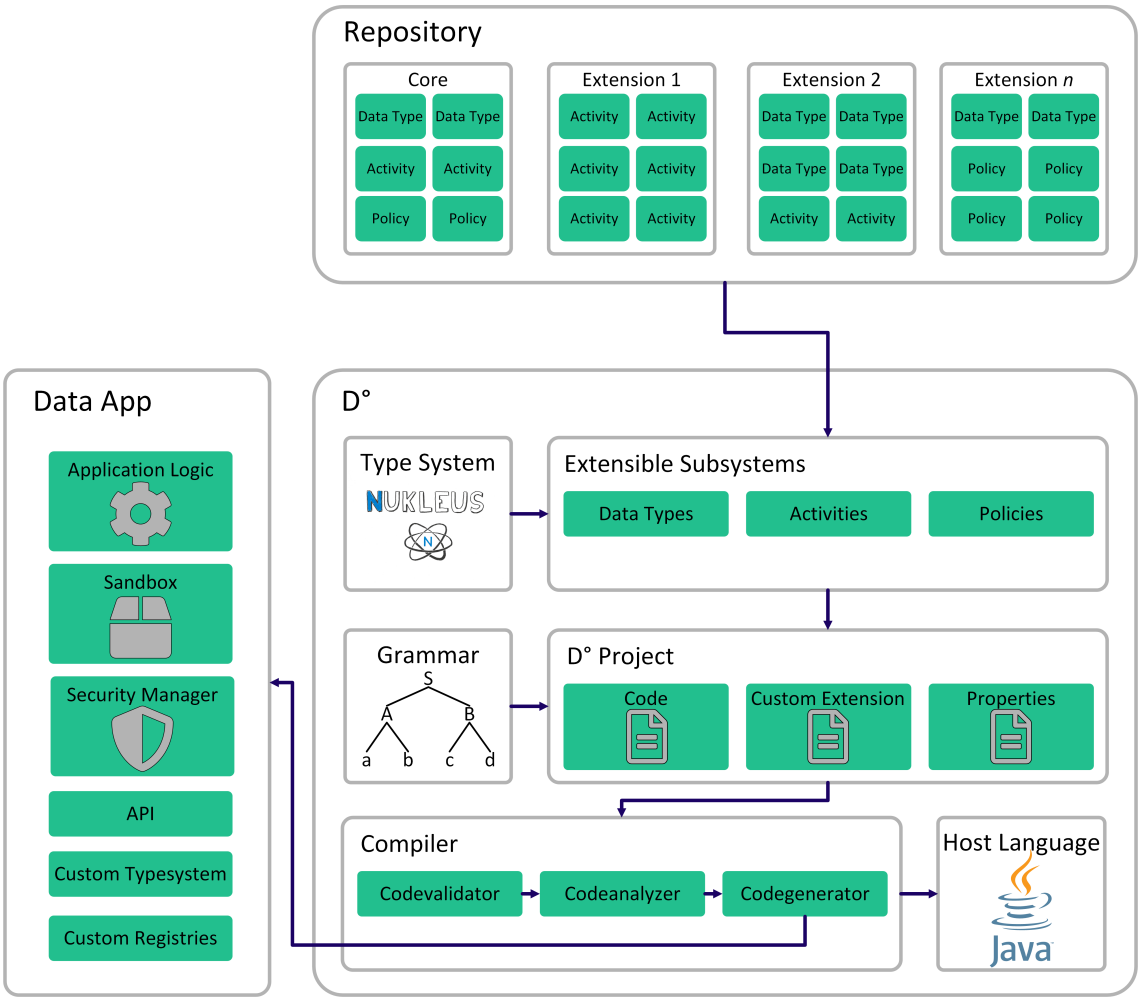

Abbildung 4.1: Schematische Darstellung, welche alle Komponenten von D° und D°-Applikationen, sowie die Beziehungen dieser Komponenten darstellt.

Entwicklungsaspekte

In diesem Abschnitt werden alle Aspekte von D° beschrieben, mit denen ein Entwickler während der Entwicklung interagieren kann oder muss.

D°-Grammatik Die Grammatik von D° definiert die Struktur von D°-Programmcode. Diese Struktur sieht für jede D°-Applikation zwei zentrale Elemente vor: Die Konfiguration der Applikation und die Applikationslogik.

Diese beiden Elemente einer D°-Applikation sind eng miteinander verbunden und befinden sich in der selben Datei. Der Bereich der Applikationslogik enthält dabei ausschließlich die Logik der Applikation, welche ausgeführt wird, wenn die Applikation aufgerufen wird. Informationen über die hierzu bereitgestellten Schnittstellen und unterschiedliche Ausführungsmodi (bspw. einmalige Ausführung und periodische Ausführung) befinden

sich in der Konfiguration. Diese Trennung erlaubt es das Verhalten von D°-Applikationen schnell anzupassen, ohne dabei Änderungen an der Applikationslogik vorzunehmen und hierdurch Fehler zu riskieren.

Neben den Einträgen, welche einen direkten Einfluss auf das Verhalten der erzeugten Applikation haben, enthält der Konfigurationsbereich weitere Daten. Diese umfassen beschreibende Werte (beispielsweise den Namen der Applikation), aber auch Daten, welche im Rahmen der Datennutzungskontrolle verwendet werden. So kann eine Liste von Tags angegeben werden, welche an die Applikation geheftet wird und im Rahmen der Überprüfung von Richtlinien verwendet werden kann. Mit den Tags kann beispielsweise beschrieben werden, für welchen Einsatzzweck (z.B. Risikoanalyse) eine Applikation verwendet wird, oder welche Aktionen auf den Eingabedaten ausgeführt werden (z.B. Persistierung).

Es ist erkennbar, dass eine sehr enge Bindung zwischen der Applikationslogik und der Konfiguration einer D°-Applikation vorliegt. Aus diesem Grund werden diese beiden Elemente auch physisch sehr nahe beieinander abgelegt, nämlich in einer einzigen Datei. Dabei werden die Konfiguration und die Applikationslogik nicht miteinander vermischt. Jedes der beiden Konzepte hat einen in sich geschlossenen Bereich innerhalb des D°-Codes.

Die Syntax der Applikationslogik orientiert sich an klassischen Programmiersprachen, um die Komplexität bei der Verwendung von D° gering zu halten. Die einzige Besonderheit ist, dass D° sowohl für einzelne Aktivitäten, als auch für eine ganze Applikation die Rückgabe von beliebig vielen Werten erlaubt. Die Werkzeuge, welche notwendig sind, um D°-Programmcode zu verarbeiten (Lexer & Parser) werden automatisch generiert.

In der Applikationslogik befinden sich keine direkt sichtbaren Informationen über Richtlinien, die in der Applikation verwendet werden sollen. Dies ist ein wichtiger Bestandteil bei der Umsetzung des Programmierparadigmas der policy-agnostischen Programmierung. Die Richtlinien sind mit den einzelnen Sprachelementen, welche innerhalb der Applikationslogik verwendet werden, verknüpft.

Erweiterbare Subsysteme Um mit D° Applikationen zu entwickeln, ist es notwendig, dass neben der Grammatik auch Sprachelemente zur Verfügung stehen, welche in der Applikation verwendet werden. Dabei umfassen diese Sprachelemente Datentypen, Aktivitäten und Richtlinien. Datentypen werden verwendet, um Dateninstanzen in Applikationen zu erzeugen und zu verwenden. Aktivitäten stellen die funktionalen Bausteine in D° dar. Im Kontext von D° sind diese Bausteine atomar, verwenden bei der Ausführung jedoch eine beliebige Menge Programmcode in der verwendeten Hostsprache. Die Richtlinien werden verwendet, um Regeln für die Datennutzungskontrolle abzubilden und die Überprüfungslogik in der Hostsprache zu definieren und für D° nutzbar zu machen.

Im Auslieferungszustand verfügt D° über keine integrierten Sprachelemente. Daraus folgt, dass D° im Auslieferungszustand nicht direkt zur Entwicklung von Applikationen verwendet werden kann. Es muss dem Compiler eine beliebige Menge von Modulen bereitgestellt werden, welche nutzbare Sprachelemente beinhalten. Module können eine beliebige Menge von allen Sprachelementen enthalten und verwendet werden, um die einzelnen Sprachelemente thematisch zu gruppieren. Dies ist vergleichbar mit Bibliotheken aus anderen Programmiersprachen. Es existiert ein core-Modul für D°, welches einige Datentypen, Akti-

vitäten und Richtlinien enthält und als Grundlage verwendet werden kann. Die vollständige Entkoppelung von verwendbaren Sprachelementen und dem Compiler erlaubt es einfacher Updates und Erweiterungen auszurollen, da nicht der gesamte Compiler ausgetauscht werden muss, sondern nur das entsprechende Modul.

Es ist für Anwender möglich neue Sprachelemente für D° zu definieren. Hierzu ist in jedem Fall eine Definition des jeweiligen Sprachelements notwendig. Diese Definition enthält alle Beschreibungen, welche notwendig sind, um das Element in D° zu verwenden. Zusätzlich können weitere optionale Informationen hinterlegt werden, welche für die Datennutzungskontrolle verwendet werden. Beispielsweise können Aktivitäten und deren Eingabeparameter mit Tags versehen werden.

Für Aktivitäten und Richtlinien ist neben der Definition noch die Erstellung einer Implementation in der Hostsprache notwendig. Dabei muss die Implementation nur wenige Regeln einhalten, um sicherzustellen dass D° die Verknüpfung zwischen Definition und Implementation findet. Die größte Einschränkung bei der Implementation ist, dass für Aktivitäten und Richtlinien festgelegte Schnittstellen existieren, welche eingehalten werden müssen.

D° unterscheidet bei Aktivitäten und Richtlinien zwischen Definitionen und Instanzen. Eine Definition enthält die vollständige Beschreibung des Sprachelements. Eine Implementation in der Hostsprache is ebenfalls Teil der Definition des Sprachelements. Diese Definitionen können nicht direkt in Applikationen verwendet werden, stattdessen müssen aus den Definitionen Instanzen erzeugt werden. Instanzen können als Sprachelemente innerhalb von Applikationen verwendet werden. Im Rahmen der Erzeugung von Instanzen ist es möglich Richtlinien mit Aktivitäten zu verknüpfen oder Parameter von Richtlinien mit konstanten Werten zu belegen. Auf diesem Weg werden Richtlinien in D°-Applikationen integriert, ohne den Programmcode mit Usage Control Mechanismen anreichern zu müssen. Darüber hinaus leistet diese Trennung einen großen Beitrag zur Wiederverwendbarkeit einzelner Sprachelemente. Beispielsweise kann die Definition einer Aktivität mehrmals instanziiert und dabei mit unterschiedlichen Richtlinieninstanzen verknüpft werden.

Projektstruktur Ein D°-Projekt, welches zur Applikationsentwicklung verwendet wird, besteht aus drei verschiedenen Komponenten. Die einzige Komponente, die verpflichtend vorhanden sein muss, ist der D°-Code. Dabei handelt es sich um eine einzelne Datei, welche die Applikationslogik und die Konfiguration für die Applikation enthält.

Zusätzlich können beliebig viele Module in dem Projekt abgelegt werden, um zusätzliche Sprachelemente in der Applikation verwenden zu können. Hierdurch können spezielle Erweiterungen für einzelne Applikationen bereitgestellt werden. Es ist unter Umständen notwendig zusätzliche Informationen für die Sprachelemente in den Modulen bereitzustellen, damit die Applikation nach der Übersetzung das gewünschte Verhalten aufweist. Beispielsweise kann ein Modul Aktivitäten beinhalten, welche Daten aus einer Datenbank holen und diese auf Datentypen abbilden, welche in der Applikation bekannt sind. Dabei ist es sinnvoll die Informationen, die notwendig sind um eine Verbindung zur Datenbank aufzubauen (z.B. Adresse und Anmeldeinformationen), nicht in dem Modul abzulegen, sondern diese

Daten zusätzlich bereitzustellen. Andernfalls müsste für jede Datenbankinstanz ein neues Modul angelegt werden.

Aus diesem Grund kann ein D°-Projekt Properties-Dateien beinhalten. Diese beinhalten beliebig viele Schlüssel-Wert Paare und werden vom Compiler während der Codegenerierung mit in die finale Applikation integriert. Die Vereinigung dieser drei Elemente bilden ein D° Projekt, welches als Eingabe für den Compiler verwendet wird.

Compiler Der D°-Compiler erzeugt in einem mehrstufigem Prozess eine ausführbare Applikation aus einem D°-Projekt. Zunächst werden alle verfügbaren Module, welche Sprachelemente enthalten, eingelesen. Dies umfasst neben den Modulen, welche im D°-Projekt abgelegt sind, auch global gespeicherte. Als Nächstes wird der D°-Code eingelesen und ein AST (Abstract Syntax Tree) erzeugt.

Im nächsten Schritt wird überprüft, ob alle Sprachelemente, welche in der Applikation verwendet werden, in den eingelesenen Modulen vorhanden sind. Diese verwendeten Sprachelemente werden gesammelt und während der Codegenerierung exportiert und in den generierten Code eingefügt. Hierdurch stehen in der finalen Applikation nicht sämtliche D°-Sprachelemente zur Verfügung, welche in den Modulen vorhanden sind. Ebenso wird der AST statisch analysiert, um die folgenden Punkte sicherzustellen:

- Für Aktivitäten werden korrekte Eingabeparameter verwendet.
- Rückgabewerte von Aktivitäten werden in passenden Variablen abgelegt.
- Variablen werden vor ihrer Verwendung initialisiert.
- Es finden nur typsichere Variablenzuweisungen statt.

Um die Anzahl der notwendigen Traversierung des D°-AST zu minimieren, wird parallel zu den statischen Analysen bereits ein AST in der Hostsprache generiert. Nach erfolgreichem Abschluss der statischen Analysen wird die eigentliche Codegenerierung angestoßen, wodurch Programmcode in der Hostsprache erzeugt wird. Dieser generierte Code wird anschließend in eine ausführbare Applikation übersetzt, welche die Ausgabe des D°-Compilers darstellt.

Typsystem Das von D° verwendete Typsystem trägt den Namen Nukleus. Nukleus ist für die Definition abstrakter Domänenmodelle entwickelt worden und hat eine zentrale Rolle in D°. Nukleus bietet die Möglichkeit zur Codegenerierung für definierte Datentypen und setzt auf String-basierte Serialisierung. Es unterstützt Mehrfachvererbung und automatisierte Validierung von Dateninstanzen, wodurch komplexe Strukturen und Beziehungen zwischen Datentypen abgebildet werden können. Zum einen wird es verwendet, um Datentypen und deren Instanzen innerhalb von D°-Applikationen zu erzeugen und zu verwalten. Zum anderen wird Nukleus im Compiler und in den generierten Applikationen verwendet, um Aktivitäten und Richtlinien zu verwalten. Trotz der zentralen Rolle von Nukleus in D° geht die vorliegende Arbeit nicht weiter auf Nukleus ein. Dies ist darin begründet, dass Nukleus nicht vom Autor dieser Arbeit entwickelt wurde.

Laufzeitaspekte

Die ausführbare Applikation, welche vom Compiler erzeugt wird, enthält neben der Applikationslogik und den umzusetzenden Richtlinien noch weitere Komponenten. Diese Komponenten sind dafür zuständig, dass die Richtlinien während der Ausführung der Applikationslogik an den korrekten Stellen ausgeführt werden. Die ausführbare Applikation ist in sich geschlossen und benötigt keinerlei externe Abhängigkeiten oder andere Artefakte, um ausgeführt zu werden. Somit ist es möglich die Applikation nach dem Übersetzungsvorgang, ohne weitere Vorbereitungen, in jeder Umgebung auszuführen, welche die Hostsprache unterstützt.

Die einzelnen Aktivitätsaufrufe der Applikationslogik werden an die Sandbox delegiert. Diese stellt sicher, dass die statischen Überprüfungspunkte der verknüpften Richtlinien an den korrekten Stellen ausgeführt werden, beispielsweise unmittelbar vor und nach dem Ausführen der Aktivitätslogik.

Der Security Manager stellt sicher, dass der dynamische Endpunkt von Richtlinien zu den richtigen Zeitpunkten ausgeführt wird. Dieser dynamische Endpunkt wird dazu verwendet bestimmte Operationen in der Hostsprache zu überwachen, beispielsweise Dateioperationen. Zu diesem Zweck werden wahlweise APIs bereitgestellt oder vorhandene APIs instrumentiert, sodass vor der Ausführung einer solchen Operation der Security Manager angefragt wird. Der Security Manager gibt den Richtlinien dann die Möglichkeit die Ausführung bei Richtlinienverstößen zu verhindern.

4.1.2 Das Remote Processing System

In diesem Abschnitt wird ein Überblick über das entwickelte Remote Processing System gegeben. Abbildung 4.2 enthält eine schematische Darstellung aller Komponenten, welche Teil des Remote Processings sind.

Die Knoten, welche über die Grenze des übergeordneten Remote Processing Blocks hinausgehen, repräsentieren die Endpunkte, welche von Applikationsbereitstellern zur Interaktion mit dem System verwendet werden. Die Pfeile, die von diesen Endpunktknoten ausgehen, zeigen auf, welche Komponenten des Remote Processing bei den Prozessen, welche durch den Aufruf des Endpunkts ausgelöst werden, verwendet werden. Dabei ist neben dem Endknoten jeder Knoten, der von dem Pfeil gekreuzt wird, an dem Prozess beteiligt.

Die Abbildung zeigt, dass die Bereitsteller von Applikationen ihre Applikation in Form von Sourcecode-Repositories beim Remote Processing System registrieren können. Das Remote Processing System klont das angegebene Repository, welches alle Ressourcen enthalten muss, die für die Übersetzung notwendig sind. Der Bereitsteller der Applikation kann eine Aktualisierung auslösen, wodurch die aktuellen Daten aus dem hinterlegten Repository bezogen werden. Dies ist beispielsweise dann relevant, wenn der Betreiber die Applikation im Rahmen des Audits zurückweist. Management Daten (bspw. welche Commits eines Applikations-Repositories verwendet wurden) können vom Remote Processing System in einem eigenem Repository hinterlegt werden, was die Migration des Remote Processing Systems zwischen unterschiedlichen Systemen vereinfacht.

Nach der Registrierung und nach Aktualisierungen kann durch den Systembetreiber ein Audit durchgeführt werden. Wird die Applikation nach dem Audit nicht zurückgewiesen,

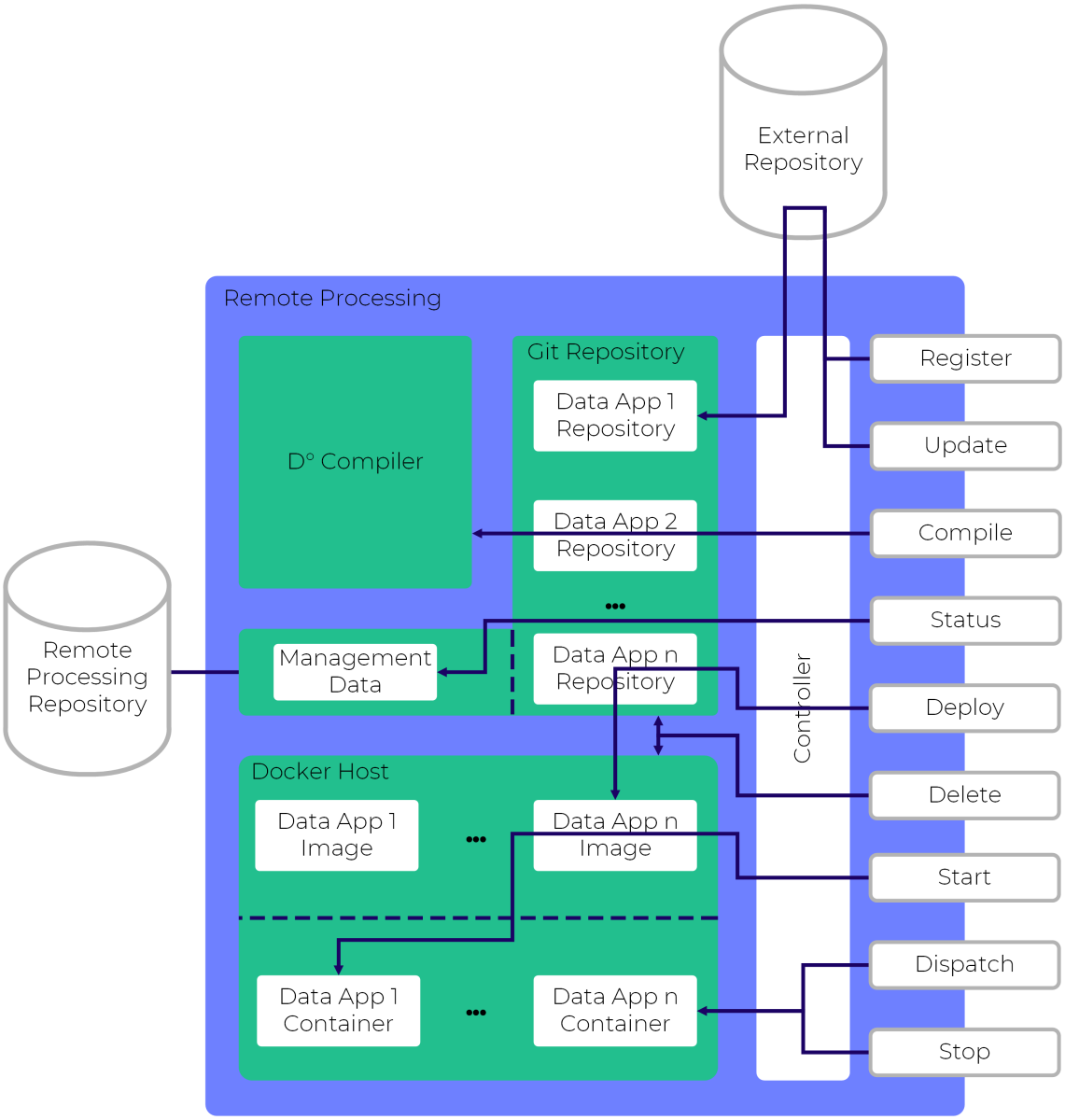

Abbildung 4.2: Schematische Darstellung, welche alle Komponenten des Remote Processing Systems, sowie die Beziehungen dieser Komponenten darstellt. Quelle: [Bru21a]

kann der Bereitsteller der Applikation die nächsten Aktionen auslösen. Sobald der Übersetzungsvorgang angestoßen wird, wird der komplette Inhalt des geklonten Repositories an den D°-Compiler übergeben, um eine ausführbare Applikation zu erzeugen.

Die Aktion, die im Nachgang der Übersetzung ausgeführt wird, erlaubt dem Bereitsteller das Erzeugen eines Applicationcontainer-Images. Dieses kann durch den Bereitsteller gestartet werden. Nach dem Start des Containers kann der Bereitsteller der Applikation Aufrufe an das Remote Processing System senden, welche an die Applikation weitergeleitet werden. Sobald die Applikation nicht mehr benötigt wird, kann der Bereitsteller den laufenden Container beenden.

4.1.3 Taxonomie für Usage Control Richtlinien

Geht es um die Bewertung der Mächtigkeit eines Usage Control Mechanismus, können zuvor einige grundlegende Aspekte betrachtet werden. In Abschnitt 2.2 wurde bereits aufgezeigt, wie sich Usage Control Mechanismen in unterschiedliche Kategorien unterteilen lassen. Aus diesem Grund wird nachfolgend zunächst betrachtet, welche grundlegenden Eigenschaften sich aus der Klassifizierung ableiten lassen. Anschließend wird betrachtet, welche Auswirkungen das auf die entwickelte Lösung hat. In diesem Zusammenhang wird auch die Mächtigkeit des entwickelten Usage Control Mechanismus näher betrachtet. Abschließend wird zur Veranschaulichung ein Praxisbeispiel demonstriert, welches aus den IDS kommt. In diesem Beispiel wird der Richtlinienkatalog der IDS betrachtet und hinsichtlich der Umsetzbarkeit in D° betrachtet.

Mächtigkeit der unterschiedlichen Klassen

Die Mächtigkeit eines Usage Control Mechanismus wird maßgeblich durch zwei Faktoren bestimmt:

1. Die Menge an Richtlinien, welche der Mechanismus technisch durchsetzen kann.
2. Die Ausdrucksstärke der Sprache, welche verwendet wird, um Richtlinien für den Mechanismus zu definieren.

Dabei ist die resultierende Mächtigkeit eines Usage Control Mechanismus die Schnittmenge dieser beiden Aspekte. Nur Richtlinien, welche definiert (und vom Mechanismus verstanden) und technisch umgesetzt werden können, sind verwendbar. Richtlinien, welche technisch umgesetzt, aber nicht mit der Policysprache definiert werden können, haben keinen Beitrag zu dem finalen Usage Control Mechanismus. Richtlinien, welche definiert, aber nicht technisch umgesetzt werden können, sind sogar negativ zu bewerten. Hierdurch wird die korrekte Anwendung des Usage Control Mechanismus verkompliziert, da es für den Anwender unter Umständen nicht intuitiv ersichtlich ist, welche definierbaren Richtlinien auch technisch umgesetzt werden können. Somit ist es wichtig bei der Entwicklung eines Usage Control Mechanismus eine angemessene Policysprache zu verwenden bzw. zu entwickeln.

Dabei ist die Frage nach der Mächtigkeit eines Usage Control Mechanismus individuell für jeden Mechanismus einzeln zu beantworten. Dennoch ist es möglich einige grundlegende Aussagen auf Basis der zuvor vorgestellten Klassifikation von Usage Control Mechanismen zu treffen. Dabei wird der Aspekt der Sprache zur Definition von Richtlinien nicht

betrachtet, da dieser für jeden Mechanismus individuell ist und sich nicht auf die Klassen verallgemeinern lässt. Stattdessen werden nachfolgend nur Betrachtungen hinsichtlich der technischen Umsetzbarkeit von Richtlinien gemacht.

Zu diesem Zweck ist es sinnvoll den Aufbau und mögliche Arten von Richtlinien zu betrachten. Eine Richtlinie, die von einem Usage Control Mechanismus umgesetzt werden soll, besteht aus einer beliebigen Menge von Regeln, welche in ihrer Kombination die Richtlinie ergeben. Im Kontext von D° werden diese Regeln als Constraints bezeichnet. Dabei sind diese Constraints atomare Aussagen, welche eingehalten werden müssen und nicht weiter zerlegt werden können. Die Verknüpfung einer beliebigen Menge dieser Constraints bildet eine Richtlinie, welche im Kontext von D° als Policy bezeichnet wird. Auch wenn Usage Control Mechanismen, bzw. deren verwendeten Sprachen zur Definition von Richtlinien, diesen Aufbau von Richtlinien nicht direkt abbilden, ist es möglich diese einzelnen Regeln in den definierten Richtlinien zu identifizieren.

Die einzelnen Regeln, aus denen die Richtlinien erzeugt werden, lassen sich in Kategorien einteilen. Dabei kann sich an den zentralen Elementen der UCON_{ABC} orientiert werden. Dies führt zu den drei Kategorien Authorization, Condition und Obligation. Abbildung 4.3 zeigt eine entsprechende (nicht erschöpfende) Taxonomie, welche diese drei Kategorien verwendet.

In die Kategorie Autorisation fallen Regeln, welche auch in traditionellen Access Control Systemen verwendet werden, um den Zugang zu Daten zu beschränken. Diese Art von Regeln lassen sich in allen drei Klassen von Usage Control Mechanismen gleich gut umsetzen. Sie erfordern keine Kenntnisse über interne Applikationsabläufe. Stattdessen werden bestimmte

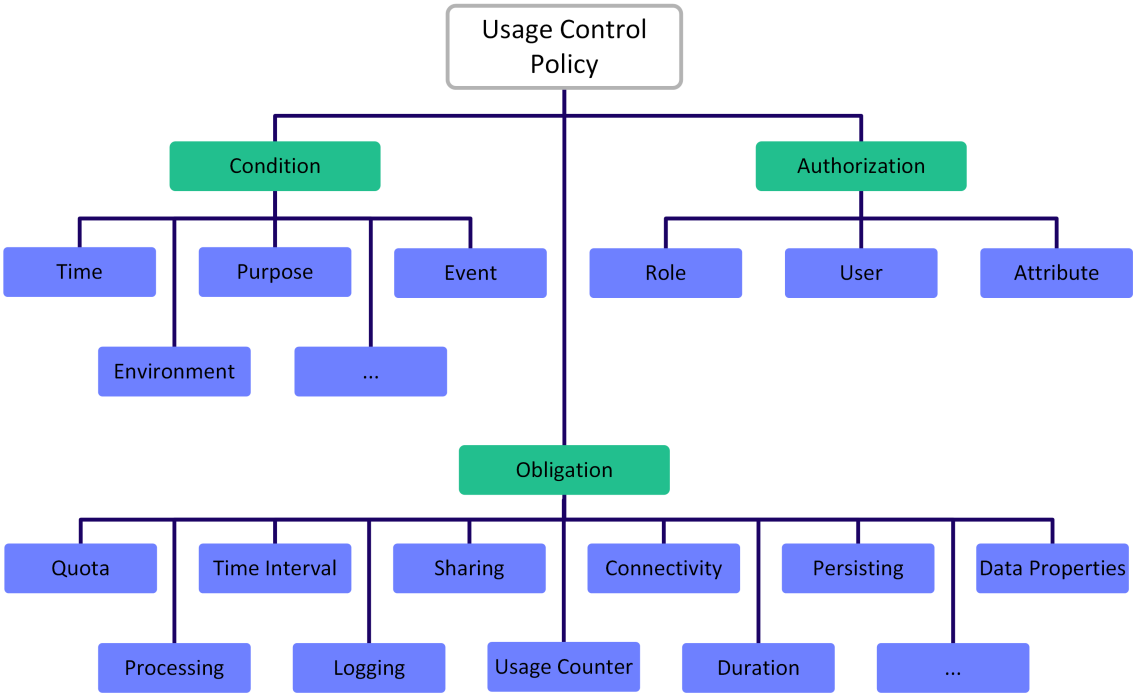

Abbildung 4.3: Taxonomie welche Usage Control Richtlinien in drei Kategorien klassifiziert.

Aspekte von Aufrufen (bspw. der aufrufende Nutzer oder dessen Rollen) mit erlaubten bzw. verbotenen Werten abgeglichen. Mithilfe der Regeln dieser Kategorie ist es möglich Applikationen bzw. einzelne Schnittstellen-Endpunkte nur für bestimmte Nutzer aufrufbar zu machen oder Aufrufe für bestimmte Rollen zu verbieten.

Ähnliches gilt für Regeln aus der Kategorie Condition. Zur Umsetzung dieser Regeln werden externe Datenquellen und Metadaten über die aufgerufene Applikation benötigt. Diese Informationen werden zur Überprüfung der Regeln verwendet. Dabei können Regeln definiert werden, welche die Nutzung von Daten bspw. nur für bestimmte Zwecke, zu bestimmten Zeitpunkten oder nach dem Auftreten (oder Ausbleiben) bestimmter Ereignisse erlaubt. Ebenso können Zugriffe abhängig von der Ausführungsumgebung erlaubt oder verboten werden, beispielsweise können Zugriffe bei zu hoher Systemlast verhindert werden. Auch diese Kategorie kann von allen drei Klassen für Usage Control Mechanismen gleich gut umgesetzt werden, da nur externe Daten bzw. allgemeine Metadaten zur Überprüfung benötigt werden. Dabei enthalten die Metadaten interne Informationen über die Applikation, beispielsweise zu welchem Zweck die einzelnen Eingabeparameter verwendet werden. Da diese Informationen im Allgemeinen jedoch statisch sind, können diese nach der Erstellung auch von externen Datenquellen bereitgestellt werden. Somit ergibt sich für diese Art von Regeln kein Vor- oder Nachteil durch die Verwendung der unterschiedlichen Usage Control Klassen, abgesehen von dem Aufwand zur Erstellung der notwendigen Metadaten.

Die umfangreichste Kategorie ist die der Obligations. Sie umfasst Regeln, welche Anforderungen definieren, die während der Ausführung durchgesetzt werden müssen. Mögliche Beispiele für solche Regeln sind:

- Schreibende Operationen dürfen ein definiertes Kontingent auf der Festplatte nicht überschreiten.
- Daten dürfen nur verschlüsselt persistiert werden.
- Die Daten dürfen nur mit anderen Applikationen geteilt werden, wenn sie aggregiert sind.
- Änderungen an den Daten müssen geloggt werden.
- Die Daten dürfen nur in einem festen Zeitintervall genutzt werden.

Die Kategorie der Obligations enthält eine Vielzahl von Regeln, welche häufig ein ausgiebiges Wissen über die internen Abläufe der jeweiligen Applikation erfordern. Ebenso finden sich in dieser Kategorie Regeln, welche erfordern, dass bei bestimmten Ereignissen zusätzliche Aktionen ausgelöst werden. Die Umsetzung dieser Regeln ist häufig einfacher, wenn sie mit Usage Control Mechanismen verwendet werden, welche direkt in die Applikation integriert sind.

Bei der direkten Integration von Usage Control Mechanismen in die Applikation ist die Applikation selbst eine White Box und die entsprechenden Überprüfungen können an den notwendigen Stellen integriert werden. Bei den anderen beiden Arten von Usage Control Mechanismen ist die Applikation als Black Box anzusehen. Daher muss das notwendige Wissen, für welche Klassen von Eingabedaten welche Überprüfungen notwendig sind, zunächst erlangt werden. Wie zuvor beschrieben, kann dieser Prozess eine erhebliche Komplexität

annehmen. Nachdem dieses Wissen gesammelt wurde, ist es notwendig entsprechende Regeln abzuleiten und im verwendeten Usage Control Mechanismus zu integrieren. Dies führt dazu, dass Teile der Applikationslogik im Usage Control Mechanismus repliziert werden, um eine passgenaue Umsetzung der definierten Regeln zu erreichen. Somit ist die Integration der Usage Control Mechanismen direkt in die Applikation für viele Regeln der Kategorie Obligations zumindest vom Standpunkt der Umsetzungskomplexität gegenüber den beiden anderen Arten von Usage Control Mechanismen vorzuziehen. Dennoch können diese Regeln in allen Arten von Usage Control Mechanismen verwendet werden, sofern das dazu notwendige Wissen erlangt werden kann.

Dies gilt jedoch nicht für alle Regeln dieser Kategorie. Wenn eine Regel erfordert, dass bei bestimmten Aktionen weitere Aktionen ausgeführt werden, sind nicht alle Regeln in allen drei Usage Control Klassen gut realisierbar. Zur Veranschaulichung wird das folgende Beispiel betrachtet: Alle Änderungen an den Daten müssen bei einem zentralen Dienst geloggt werden. Wie zuvor beschrieben, ist es häufig möglich zu identifizieren, für welche Klassen von Eingabedaten die Eingabedaten geändert werden und somit die Regel angewendet werden muss. Somit ist diese Regel mit allen drei Klassen von Usage Control Mechanismen umsetzbar. Ist eine enge zeitliche Koppelung zwischen der auslösenden Aktion und der Ausführung der zusätzlichen Aktion erforderlich, stoßen die Usage Control Mechanismen, welche die Applikation als Black Box betrachten, an ihre Grenze. Zur Veranschaulichung wird das zuvor genannte Beispiel leicht modifiziert: Alle Änderungen an den Daten müssen unmittelbar nach der Änderung bei einem zentralen Dienst geloggt werden.

Durch diese Änderung wird die Umsetzung für Black Box Ansätze unmöglich, da keine Kenntnis darüber erlangt werden kann, zu welchen genauen Zeitpunkten Änderungen an den Daten erfolgen. Diese Zeitpunkte werden durch eine Vielzahl von Parametern beeinflusst und lassen sich nicht exakt im Voraus bestimmen. Mögliche relevante Parameter, welche die Zeitpunkte beeinflussen, sind beispielsweise Systemlast, Größe der Eingabedaten und Antwortzeiten externer Services.

Ähnlich verhält es sich, wenn die Regel Aktionen erfordert, welche nicht gemeinsam mit der eigentlichen Datennutzung ausgeführt werden. Beispielsweise wenn durch eine Richtlinie die Speicherdauer beschränkt werden soll und nach einem gewissen Zeitraum sichergestellt werden muss, dass die Daten gelöscht worden sind. In diesen Fällen sind Ansätze, welche Usage Control direkt in der datenverarbeitenden Applikation umsetzen, ungeeignet. Der Grund hierfür ist, dass die Speicherung häufig außerhalb der eigentlichen Applikation erfolgt, beispielsweise in einer Datenbank. Der Wirkungsbereich der Usage Control Mechanismen ist auf die datenverarbeitende Applikation beschränkt und hat keine Kontrolle über die Datenbank. So kann durch die Usage Control Lösung in der Applikation nicht verhindert werden, dass eine andere Applikation Kopien der Daten anlegt, welche beim Löschvorgang nicht berücksichtigt werden.

Somit kann abschließend festgestellt werden, dass eine Vielzahl von Regeln in allen drei Kategorien von Usage Control Mechanismen verwendet werden kann. Dabei ist die Komplexität bei der Verwendung von Usage Control als separate Software um die Applikation oder der Integration in eine zentrale Komponente häufig höher als bei der direkten Integration in die eigentliche Applikation. Einschränkungen zur Umsetzbarkeit können auftreten, wenn

die verwendeten Regeln die Ausführung zusätzlicher Aktionen erfordern. Grundsätzlich ist es nicht möglich allgemeingültige Aussagen zur Umsetzbarkeit von Richtlinien in den drei Klassen von Usage Control Mechanismen zu treffen, da die unterschiedlichen Regeln frei miteinander kombiniert werden können und somit auf Einzelfallbasis festgestellt werden muss, ob eine Richtlinie in einem der Usage Control Mechanismen umgesetzt werden kann.

Folgen für den entwickelten Ansatz

Bei dem entwickelten Lösungsansatz handelt es sich um einen White Box Ansatz, bei dem die Usage Control Mechanismen direkt in die Applikation integriert werden. Wie bereits im vorherigen Abschnitt beschrieben, hat dies nur einen geringen Einfluss auf die Mächtigkeit des Usage Control Mechanismus. Durch das Vorgehen kann eine enge zeitliche Koppelung zwischen ausgeführten Aktionen und der Überprüfung relevanter Richtlinien erreicht werden. Nachfolgend wird untersucht, inwiefern der entwickelte Usage Control Mechanismus verwendet werden kann, um die Kategorien der zuvor vorgestellten Richtlinien-Taxonomie umzusetzen. Anschließend wird beschrieben, wie es sich mit der Umsetzbarkeit verhält, wenn mehrere Regeln in einer Richtlinie kombiniert werden.

Authorization Die Regeln, welche in die Kategorie Autorisierung fallen, sind alle sehr ähnlich hinsichtlich ihrer Umsetzung. Bei einem Aufruf werden Daten mitgeliefert, welche Informationen über den Aufrufer beinhalten. Beispiele für solche Daten sind der Benutzername des Aufrufers oder Rollen, über die der Aufrufer verfügt. In der Regel enthalten sind entsprechende Einschränkungen, welche Aktionen, abhängig von den zuvor genannten Daten, erlauben oder verbieten.

Zur Auswertung der Regeln müssen die Daten über den Aufrufer validiert werden. Nach der Validierung findet ein Abgleich mit den Verboten und Erlaubnissen der Regel statt. Das Ergebnis dieses Abgleichs ist gleichzeitig auch das Ergebnis der Regelauswertung.

Diese Kategorie von Regeln lässt sich innerhalb von D° vollständig umsetzen. Dabei wird der Endpunkt der Richtlinien-Schnittstelle, welcher für die Definition von Vorbedingungen verwendet wird, genutzt, um die Überprüfungslogik zu implementieren. Hierdurch wird sichergestellt, dass die Regel vor der Ausführung ausgewertet wird und die Ausführung im Falle eines Regelverstoßes verhindert wird.

Die Daten über den Aufrufer können direkt als Eingabeparameter der Applikation definiert werden. Es ist jedoch ebenso möglich andere Quellen für die notwendigen Daten zu verwenden. Beispielsweise kann eine D°-Applikation, welche eine HTTP-Schnittstelle bereitstellt, JWTs (JSON Web Tokens) verwenden. Eine Applikation, welche über die Kommandozeile verwendet wird, kann Umgebungsvariablen und Systemnutzer verwenden. Durch das erweiterbare Subsystem für Kontextinformationen, welches Teil von D° ist, können diese Daten, welche nicht in den Eingabeparametern enthalten sind, zur Überprüfung von Richtlinien verwendet werden.

Condition Bei den Regeln, welche in die Kategorie Condition fallen, verhält es sich nicht ganz so einheitlich wie bei den Authorization-Regeln. Die Regeln dieser Kategorie lassen sich in zwei Gruppen unterteilen.

Die erste Gruppe enthält Regeln, welche zusätzliche Informationen über die ausgeführte Applikation erfordern. Diese Regeln werden verwendet, um den Nutzungszweck bzw. die -art einzuschränken. Ein Beispiel für solche Regeln ist die Einschränkung, dass Daten nur für das Risikomanagement jedoch nicht für Marketingzwecke verwendet werden dürfen. Ein anderes Beispiel ist, dass Daten von einer Applikation nur verwendet werden dürfen, wenn diese die Daten anonymisiert. Diese Regeln benötigen zusätzliches Wissen über die ausführende Applikation, welches sich in vielen Fällen nicht aus dem Programmcode ablesen lässt; weder manuell noch automatisiert.

D° erlaubt unterschiedlichste Elemente mit Tags zu versehen. Zum einen können Aktivitäten und deren Instanzen, einzelne Parameter von Aktivitäten, sowie ganze Applikationen mit Tags ausgestattet werden, um ihren Einsatzzweck und ausgeführte Aktionen zu beschreiben. Darüber hinaus können individuelle Datensätze während der Laufzeit mit Tags versehen werden, welche beispielsweise den aktuellen Zustand des Datensatzes beschreiben. Dabei werden die Tags an Datensätzen von Aktivitäten vergeben und modifiziert.

Somit sind die Regeln, die in diese Gruppe fallen, prinzipiell innerhalb von D° nutzbar und können umgesetzt werden. Dies gilt aber nur, sofern die verfügbaren Mechanismen zur Verwendung von Tags ausreichen, um die notwendigen Informationen sinnvoll abzubilden. Somit lässt sich nicht allgemeingültig sagen, dass alle denkbaren Regeln dieser Gruppe unterstützt werden können.

Darüber hinaus existiert eine weitere Einschränkung bei der Umsetzung von Regeln aus dieser Gruppe. Die Vergabe von Tags an Applikationen und Aktivitäten erfolgt durch den Entwickler. Das selbe gilt für Tags, welche während der Ausführung von Aktivitäten an Datensätze geheftet werden. Somit besteht hier ein erhöhtes Risiko für menschliche Fehler und absichtliche Umgehungen der Regeln. Dementsprechend ist entsprechende Sorgfalt notwendig, welche gegebenenfalls von passenden Mechanismen (bspw. Codeaudits) unterstützt werden, um eine korrekte Regelumsetzung für diese Gruppe zu erreichen.

Dabei bezieht sich diese Problematik ausschließlich auf die Entwicklungszeit und die Bereitstellung der zusätzlich benötigten Daten. Zur Laufzeit ist eine korrekte Umsetzung durch die Codegenerierung sichergestellt.

Die zweite Gruppe von Regeln in der Kategorie enthält ebenfalls Regeln, welche zusätzliche Informationen zur Überprüfung benötigen. Der Unterschied zur ersten Gruppe ist dabei, dass die Regeln sich nicht auf die ausführende Applikation beziehen. Diese Gruppe von Regeln ermöglicht es die Ausführung auf Basis von unterschiedlichen Ereignissen zu limitieren bzw. zu erlauben. Einige Beispiele für solche Regeln sind:

- Die Ausführung darf nur vor bzw. nach einem Bestimmten Zeitpunkt erfolgen.
- Daten dürfen nur nach dem Leisten einer Zahlung verwendet werden.
- Die Ausführung darf nur auf einer Trusted Computing Platform erfolgen.

Diese Gruppe von Regeln lässt sich innerhalb von D° verwenden. Es muss lediglich sichergestellt werden, dass der Eintritt des Ereignisses, auf welches sich die Richtlinie bezieht, überprüft werden kann. Zu diesem Zweck kann von der Richtlinie sowohl das ausführende System, als auch externe Services angefragt werden. Somit sind alle Regeln dieser Gruppe innerhalb von D° umsetzbar, für die es eine entsprechende Informationsquelle

gibt, welche es erlaubt den Status der referenzierten Ereignisse zu überprüfen. Dabei müssen diese Informationsquellen möglichst sicher und zuverlässig sein, um eine korrekte Umsetzung der Regeln zu gewährleisten.

Zusammenfassend kann festgestellt werden, dass die Regeln dieser Kategorie in D° verwendet werden können. Dennoch existieren Risiken durch (absichtliche und unabsichtliche) Fehler bei der Vergabe von Tags und nicht verfügbare oder unzuverlässige Informationsquellen, welche während der Auswertung der Regeln verwendet werden.

Obligation Die dritte Kategorie der Obligations ist die umfangreichste. Dies bezieht sich sowohl auf die Anzahl der enthaltenen Regeln, als auch deren Vielfältigkeit. Nachfolgend werden die Regeln dieser Kategorie wieder in Gruppen unterteilt, welche nacheinander betrachtet werden.

Die erste Gruppe umfasst Regeln, welche die Ausführung bestimmter Aktionen nur dann erlaubt, wenn im Vorfeld andere Aktionen ausgeführt wurden bzw. ein bestimmter Zustand der verarbeiteten Daten hergestellt wurde. Einige Beispiele für diese Regeln sind:

- Daten dürfen nur dann an andere Applikationen übermittelt werden, wenn sie zuvor anonymisiert wurden.
- Eine Persistierung der Daten ist nur zulässig, wenn diese vorher verschlüsselt wurden.
- Daten, die nicht anonymisiert sind, dürfen nicht aggregiert werden.

Bei dieser Gruppe von Regeln ist eine gewisse Überschneidungen zu den Regeln aus der Kategorie Condition erkennbar. Der Unterschied zeichnet sich dadurch aus, dass die Regeln dieser Gruppe feingranularer sind und sich auf die Ausführung spezifischer Aktionen beziehen, wohingegen die Regeln der Kategorie Condition Regeln auf Applikationsebene enthalten. Zuvor wurde bereits beschrieben, dass D° es ermöglicht Datensätze mit Tags zu versehen. Dieser Mechanismus kann auch für diese Regeln verwendet werden, um den Zustand der Datensätze zu überprüfen.

Um sicherzustellen, dass die Regeln an den korrekten Stellen überprüft werden, ist es notwendig die Verwendung der Aktionen, auf welche sich die Regel bezieht, zur Laufzeit zu erkennen. Hierfür gibt es zwei Methoden. Zum einen kann der Security Manager verwendet werden, um Schnittstellen der Hostsprache zu überwachen. Auf diesem Wege können alle Aufrufe einer Schnittstelle automatisch zur Laufzeit entdeckt und entsprechende Richtlinienüberprüfungen ausgeführt werden. Zum anderen können Tags an Aktivitäten angebracht werden, welche die enthaltenen Aktionen beschreiben. Dies ist notwendig für die Fälle, in denen die entsprechende Schnittstelle nicht vom Security Manager überwacht wird, oder wenn es keine dedizierte Schnittstelle für die Operation gibt.

Für beide Komponenten, welche von den Regeln in dieser Gruppe verwendet werden, existieren Möglichkeiten zur Verwendung in D°. Somit kann diese Gruppe von Regeln innerhalb von D° verwendet werden.

Die nächste Gruppe enthält Regeln, die es notwendig machen Datennutzung auf eine definierte Art und Weise zu erfassen. Dies kann beispielsweise durch lokales Logging erfolgen. Ebenso kann eine entsprechende Nachricht an einem zentralen Logging-Service erforderlich sein, oder eine Mitteilung an einen Service, welcher die Datennutzung registriert.

Ein lokales Loggen der Datennutzung kann in D° sowohl innerhalb der Applikation, als auch einer zusätzlichen Software auf dem ausführenden System erfolgen. Auch ist es möglich externe Services über die Datennutzung zu informieren, sofern die notwendigen Serviceendpunkte bekannt und erreichbar sind. Somit kann diese Regelgruppe ohne Einschränkung in D° umgesetzt werden.

In der nächsten Gruppe befinden sich Regeln, welche Einschränkungen der Nutzungszeit erlauben. Bereits in der Kategorie Condition existieren Regeln, welche die Datennutzung vor bzw. nach einem bestimmten Zeitpunkt erlauben und verbieten können. Die Regeln dieser Gruppe gehen darüber hinaus. Sie erlauben es Zeitintervalle zu definieren, in denen die Datennutzung erlaubt ist. Ebenso können erlaubte Nutzungsdauern definiert werden.

Die Definition von erlaubten bzw. verbotenen Zeitintervallen zur Datennutzung ist in D° ohne Probleme möglich. Bei der Umsetzung gibt es allerdings eine Einschränkung, welche beachtet werden muss. Die Überprüfung der Regeln findet zu festen Zeitpunkten während der Ausführung der Applikation statt. Nur zu diesen Zeitpunkten ist es möglich festzustellen, ob ein erlaubtes Zeitintervall verlassen bzw. ein verbotenes betreten wurde. Dabei ist es nicht möglich die Zeit, die bis zur nächsten Überprüfung der Richtlinien verstreicht, zu bestimmen. Aus diesem Grund ist eine vollständige und sichere Einhaltung der definierten Zeitintervalle für D° nicht gegeben.

Diese Einschränkung besteht ebenfalls für Regeln, welche eine Nutzungsdauer definieren. Bei diesen Regeln muss darüber hinaus ein externer Service verfügbar sein, welcher es ermöglicht den Beginn des Nutzungszeitraums zu beziehen. Somit ist diese Gruppe von Regeln nur mit Einschränkungen in D° umsetzbar.

In der nächsten Gruppe befinden sich Regeln, welche eine mengenmäßige Beschränkung für Aktionen definieren. Beispiele für solche Beschränkungen sind, dass die Daten nur dreimal verwendet werden dürfen, oder dass eine definierte Quota nicht überschritten werden darf, wenn Daten auf die Festplatte geschrieben oder an externe Services gesendet werden.

Die rein zählenden Regeln können dabei sowohl über den Security Manager, als auch über die Tags an Aktivitäten realisiert werden. Sofern diese Regel ausschließlich innerhalb einer einzelnen D°-Applikation umzusetzen ist, sind keine weiteren Elemente zur Umsetzung notwendig. Handelt es sich jedoch um ein größeres Szenario, welches verschiedene Applikationen einsetzt, in welchem sichergestellt werden muss, dass die Daten nur dreimal verwendet werden, ist ein externer Service notwendig, welcher die hierzu notwendigen Informationen liefert.

Dies gilt ebenfalls für Regeln, die eine Quota vorschreiben. Dabei reicht für diese Regeln die Verwendung von Tags an Aktivitäten nicht aus. Für genaue Messungen der Nutzung der Quota müssen die Operationen, welche die entsprechenden Aktionen ausführen, direkt mit dem Security Manager überwacht werden. Dies liegt darin begründet, dass erst zum Zeitpunkt, wenn der Security Manager zur Ausführung einer Operation befragt wird, feststeht, welche genauen Eingabeparameter verwendet werden und wie hierdurch die Quota genutzt wird.

Dies beschränkt die Umsetzbarkeit von Regeln, welche eine Quota für bestimmte Aktionen definieren, in D° auf diejenigen ein, bei denen eine Schnittstellenüberwachung der

entsprechenden Aktion durch den Security Manager gewährleistet ist. Die zählenden Regeln unterliegen nicht dieser Einschränkung und sind in D° umsetzbar.

Komposition von Regeln Nachdem die unterschiedlichen Gruppen von Richtlinien betrachtet und hinsichtlich ihrer Umsetzbarkeit in D° bewertet wurden, muss ein weiterer Punkt betrachtet werden. Die Einträge der vorgestellten Regelgruppen können eigenständig als Richtlinien verwendet werden. Ebenso ist es möglich diese Regeln miteinander zu kombinieren, was zu beliebig komplexen Richtlinien führen kann. D° unterstützt diese Komposition von Regeln durch die beiden Konzepte Policy und Constraint.

Eine Constraint repräsentiert in D° eine einzelne Regel, welche direkt als Richtlinie verwendet werden kann. Demgegenüber umfasst die Policy beliebig viele Richtlinien, welche alle gleichzeitig umgesetzt werden müssen. Dabei handelt es sich bei der Policy selbst ebenfalls um eine Richtlinie, welche in anderen Policies verwendet werden kann. Die Policy führt eine Und-Verknüpfung der enthaltenen Richtlinien durch. Wie dabei eventuelle Konflikte bzw. Überschneidungen aufgelöst werden, hängt davon ab, welche Endpunkte der D°-Richtlinien Schnittstelle für die Überprüfung verwendet werden.

Die Endpunkte für Vor- und Nachbedingungen sorgen für einen Abbruch der Ausführung, sofern sie nicht erfüllt sind. Somit müssen alle Vor- und Nachbedingungen aller Richtlinien gleichzeitig erfüllt sein. Bei dem Endpunkt, welcher vom Security Manager aufgerufen wird, sobald eine überwachte Schnittstelle verwendet wird, verhält es sich anders. Zunächst wird von allen Richtlinien eine Entscheidung hinsichtlich der Gültigkeit der Ausführung eingeholt. Diese werden anschließend gemäß des entwickelten Greylistings ausgewertet, wobei eine Erlaubnis ein Verbot immer überdeckt [Bru20].

Die folgende Richtlinie, welche eine Verknüpfung von zwei Regeln enthält, lässt sich somit nicht abbilden, wenn die Überprüfung in den Vorbedingungen stattfindet: Daten dürfen persistiert werden, wenn es sich beim ausführenden System um eine Trusted Computing Plattform handelt, oder im Vorfeld eine Anonymisierung der Daten stattgefunden hat. Sind die Regeln so implementiert, dass eine Überprüfung im Rahmen der Anfrage des Security Managers geschieht, ist diese Kombination problemlos umsetzbar. Alternativ kann eine entsprechende, maßgeschneiderte Constraint entwickelt werden, welche genau diesen Fall abbildet. Diese Constraint kann dann auch die Vorbedingung zur Umsetzung verwenden.

Somit können durch die Verwendung von Constraints und Policies, sowie der passenden Verwendung der D°-Richtlinien Schnittstelle, komplexe Richtlinien zusammengesetzt und umgesetzt werden. Sollten diese Möglichkeiten nicht ausreichen, kann eine entsprechend maßgeschneiderte Constraint entwickelt werden, welche den individuellen Fall umsetzt. Diese Entscheidung wurde bewusst getroffen und hat keinen negativen Einfluss auf die Mächtigkeit des Usage Control Mechanismus von D°. Durch die eingeschränkten Möglichkeiten bei der Komposition von Regeln wird verhindert, dass die Komplexität in der Nutzung der Funktionalität übermäßig ansteigt, was zu Fehlern und unerwünschten Ergebnissen bei der Überprüfung von Richtlinien führen könnte.

Praxisbeispiel

Die Entwicklung von D° ging aus den IDS hervor und wurde maßgeblich von diesen motiviert. Dabei wurde auch die Entwicklung weiterer Usage Control Mechanismen im

Kontext der IDS vorangetrieben, wodurch der Anwender die Möglichkeit hat für den individuellen Anwendungsfall die passende Realisierung von Usage Control zu verwenden. Darüber hinaus können Anwender beliebige andere Lösungen für Usage Control verwenden, wenn diese besser geeignet sind. Zu diesem Zweck ist es notwendig, dass der Anwender die Eignung der individuellen Usage Control Mechanismen beurteilen kann.

Dabei muss bei dieser Entscheidung, neben der allgemeinen Funktionsweise der jeweiligen Lösung, die Mächtigkeit mit einbezogen werden. Andernfalls ist nicht sichergestellt, dass die Richtlinien, die in dem Szenario umgesetzt werden müssen, auch von der verwendeten Usage Control Lösung umgesetzt werden können. Wie in Abschnitt 4.1.3 bereits beschrieben, ist es notwendig eine Bewertung der Mächtigkeit individuell für jeden Ansatz durchzuführen.

Um eine Basis für Vergleiche der unterschiedlichen Usage Control Lösungen auf Basis ihrer Mächtigkeit zu liefern, wurde im Rahmen der IDS eine Menge von Richtlinien definiert, zu denen sich Lösungen positionieren können. Diese Richtlinien wurden dabei u.a. aus Anwendungsfällen mit Industrieunternehmen extrahiert. Dabei ist dieser Satz an Richtlinien nicht fest und kann im Laufe der Zeit erweitert oder modifiziert werden, um sich ändernde Anforderungen abzubilden.

Zum Zeitpunkt der Erstellung des vorliegenden Dokuments umfasst der in den IDS verwendete Satz 20 unterschiedliche Richtlinien [Eit21]. Dabei sind 11 dieser 20 Richtlinien offiziell in D° nutzbar. Bei den neun verbleibenden Richtlinien handelt es sich nicht ausschließlich um solche, welche mit D° nicht verwendet werden können.

Im Kontext der IDS werden D°-Applikationen als Bestandteil von Connectoren verwendet, welche Aufrufe entgegen nehmen und an die Applikationen weiterleiten. Deshalb ist es sinnvoll Richtlinien, welche auf Authentifizierung oder Systemeigenschaften basieren, bereits im Connector zu überprüfen und nicht erst in der Applikation. Aus diesem Grund sind die entsprechenden Richtlinien zwar in D° nutzbar, eine Umsetzung auf der Ebene des Connectors ist dabei jedoch vorzuziehen. Dies betrifft sechs der neun Richtlinien, deren Umsetzung am besten auf der Ebene des Connectors stattfindet. Somit kann D° – mit 17 von 20 – einen Anteil von 85% der Richtlinien, welche in den IDS als Grundlage vereinbart sind, umsetzen.

Nachfolgend werden die drei Richtlinien, welche nicht innerhalb von D° nutzbar sind, betrachtet. Die erste dieser drei Richtlinien erlaubt eine Datennutzung für einen bestimmten Zeitraum und fordert die anschließende Löschung der Daten vom ausführenden System. Somit fällt diese Richtlinie in die Kategorie Conditions. Um diese Richtlinie in D° verwenden zu können, wäre es notwendig aus dem Usage Control Mechanismus, welcher sich nur über die D°-Applikation erstreckt, eine Aktion im ausführenden System bzw. anderen Systemen auszulösen. Eine D°-Applikation hat keine Kenntnis über das System, in dem sie ausgeführt wird, und welche weiteren Applikationen in diesem System ausgeführt werden. Dieser Aspekt erschwert die Umsetzung der Regel in D° enorm. Dennoch könnte auf Einzelfallbasis eine entsprechende Constraint implementiert werden, welche maßgeschneidert auf die Ausführungsumgebung der jeweiligen Applikation ist und entsprechende Löschungen ermöglicht. Die Hauptproblematik, welche einer Umsetzung der Regel entgegensteht ist, dass die Überprüfung von Richtlinien innerhalb von D°-Applikationen zeitlich eng an die Ausführung der Applikationslogik gebunden ist. Eine Richtlinie, welche Aktionen erfordert, die in der Zukunft liegen, und somit nicht während der Ausführung der Applikationslogik

erfolgen, sind nicht in D° realisierbar. Das System, welches die D°-Applikation ausführt, ist besser geeignet, um diese Richtlinie umzusetzen.

Die nächste Richtlinie erfordert die Modifikation von Daten im Transit. Sie ist dazu gedacht, kritische Informationen aus Daten zu entfernen oder zu verfremden, bevor diese an Applikationen, welche die Daten verarbeiten, weitergegeben werden. Die Richtlinie kann ebenfalls verwendet werden, um beliebige andere Modifikationen an den Daten vorzunehmen, bevor diese an Applikationen übergeben werden. Dabei muss diese Modifikation während des Transports erfolgen und nicht erst in der verarbeitenden Applikation. Beispielsweise könnte ein Message Bus die erforderlichen Aktionen ausführen bevor eine Applikation aufgerufen wird. Da D° verwendet wird, um die datenverarbeitende Applikation zu entwickeln und die verwendeten Usage Control Mechanismen ein fester Bestandteil dieser Applikation sind, ist es nicht möglich diese Richtlinie in D° umzusetzen. Wird D° verwendet, um eine dem Message Bus ähnliche Applikation zu entwickeln, können die Aktionen, welche durch die Richtlinie erfordert werden, unter Umständen in eigenen Richtlinien umgesetzt werden. Dies liegt darin begründet, dass die D°-Applikation in diesem Fall selber den Transport der Daten vornimmt und die eigentliche Verarbeitung in anderen Applikationen stattfindet. Die Modifikation der Daten und die Weiterleitung an andere Applikationen ist dabei die Datenverarbeitung, welche von der Applikation geleistet wird. Diese empfangenden Applikationen können ebenfalls mit D° implementiert sein, sind jedoch eigenständig. Eine Komponente zur Weiterleitung von Nachrichten ist Bestandteil der IDS Connectoren, weshalb diese Richtlinie am besten direkt im Connector umgesetzt wird.

Die dritte Richtlinie adressiert Situationen, in denen Daten vom Konsumenten mit weiteren Parteien geteilt werden. In diesem Fall erfordert die Richtlinie, dass zusätzliche, vom Rechteinhaber definierte Richtlinien vom Konsumenten an die Daten geheftet werden und mit den zusätzlichen Parteien eine Vereinbarung getroffen wurde, welche die Einhaltung dieser Richtlinien sicherstellt. Erst nach dem Treffen dieser Vereinbarung ist es dem Datenkonsumenten erlaubt die Daten an die dritte Partei zu übergeben. Da D° über keine Möglichkeit verfügt, diese Vereinbarungen automatisiert auszuhandeln und abzuschließen, kann diese Richtlinie nicht in D° umgesetzt werden. Im Rahmen der IDS findet die Kommunikation zwischen den unterschiedlichen Teilnehmern über die Connectoren statt. Somit ist eine direkte Weiterleitung der Daten von einer Applikation, die in einem Connector läuft, an einen anderen Teilnehmer nicht das intendierte Vorgehen der IDS. Aus diesem Grund sollte diese Richtlinie im Connector verankert und umgesetzt werden. Sendet ein Connector Daten an einen anderen Connector, muss der versendende Connector sicherstellen, dass die notwendigen Richtlinien mit den Daten versandt werden und notwendige Vereinbarungen mit dem empfangenden Connector geschlossen wurden.

Nachfolgend findet eine kurze Betrachtung der Richtlinien, welche in D° nutzbar sind, statt. Es wird betrachtet, wie diese Richtlinien sich in die zuvor vorgestellte Taxonomie einordnen lassen. Dabei fällt nur eine einzelne Richtlinie in die Kategorie Authorization. Diese erlaubt es die Ausführung für bestimmte Rollen einzuschränken. Insgesamt sieben Richtlinien fallen in die Kategorie der Obligations. Diese umfassen, neben Richtlinien für Zeiträume, auch Quotas und Zustände, welche für Daten hergestellt werden müssen, bevor bestimmte Aktionen ausgeführt werden. Bei den verbleibenden 9 Richtlinien handelt es sich um Conditions. Diese stellen Anforderungen an die Ausführungsumgebung der

verarbeitenden Applikation und erfordern, dass bestimmte Ereignisse eingetreten oder externe Bedingungen erfüllt sind.

Als Nächstes wird die Umsetzungsquote von D° bei den IDS Richtlinien mit den anderen Usage Control Lösungen der IDS verglichen. LUCON erreicht eine Quote von 60% und MYDATA 100%, wodurch sich D° mit 85% in der Mitte platziert. Dabei sind diese Quoten nur bedingt aussagekräftig. Dies liegt darin begründet, dass die unterschiedlichen Usage Control Lösungen für unterschiedliche Einsatzgebiete entwickelt worden sind, was sich auch im technischen Aufbau der jeweiligen Lösung widerspiegelt. Ähnliches gilt für die Richtlinien, welche in den IDS verwendet werden.

Diese enthalten Richtlinien, welche am besten auf der Ebene der Applikationsausführung umgesetzt werden; beispielsweise dass die Nutzung von Daten bei einem zentralen Service geloggt werden müssen. Ebenfalls existieren Richtlinien, welche sinnvollerweise an einer anderen Stelle, beispielsweise auf dem Transportweg, umgesetzt werden. Ein Beispiel hierfür wäre die Modifikation von Daten, welche sich im Transit befinden. Um eine aussagekräftigere Umsetzungsquote zu erhalten, wird die Teilmenge der IDS Richtlinien betrachtet, für die eine Umsetzung auf der Ebene der Applikationsausführung sinnvoll ist. Dabei kann es für die unterschiedlichen Richtlinien durchaus mehrere Ebenen geben, an denen eine Umsetzung sinnvoll erfolgen kann.

Betrachtet man die drei Richtlinien, welche in den vorherigen Abschnitten betrachtet wurden und nicht in D° umgesetzt werden können, wird erkennbar, dass es sich dabei um Richtlinien handelt, welche nicht der Applikationsausführung zugeordnet sind. Somit sind diese Richtlinien nicht relevant für die Bewertung der Mächtigkeit des entwickelten Usage Control Mechanismus. Betrachtet man nur die relevanten Richtlinien, kommt man so zu einem Satz von 17 Richtlinien. Für diesen Satz erreichen sowohl D°, als auch MYDATA eine Umsetzungsquote von 100% und LUCON rund 53%.

Nachfolgend werden die 20 Richtlinien in tabellarischer Form dargestellt und aufgezeigt, ob diese in D° umsetzbar sind. Dabei sind umsetzbare Richtlinien mit einem Haken (✓) in der zweiten Spalte markiert. Richtlinien, die umgesetzt werden könnten, aber aus verschiedenen Gründen nicht umgesetzt werden, haben einen eingeklammerten Haken als Symbol. Die Richtlinien, die nicht in D° umgesetzt werden können, sind mit einem Kreuz (✗) gekennzeichnet. In der letzten Spalte der Tabelle finden sich für einige Richtlinien weiterführende Beschreibungen, welche die Richtlinie oder Informationen zur Umsetzung in D° erläutern. Dabei werden in der Tabelle die englischen Namen der Richtlinien verwendet, die auch im entsprechenden Dokument der IDS verwendet werden [Eit21].

	Name	**In D°**	**Anmerkung**
1	Allow the usage of data	✓	-
2	Perpetual Data Sale (Payment once)	✓	-

3	Data Rental (Payment frequently)	(✓)	Diese Richtlinie bildet ein klassisches Abonnement ab. Dabei muss während der Nutzung bzw. Verarbeitung der Daten fortlaufend überprüft werden, ob ein aktives Abonnement besteht. Da potentiell beliebig viele Applikationen gleichzeitig die Daten verwenden, entsteht nicht unerheblicher Overhead, wenn alle Applikationen fortlaufend den Status des Abonnements prüfen. Aus diesem Grund wäre eine Überprüfung der Richtlinie im übergeordnetem System, dem Connector, vorzuziehen. Der Connector ist dafür verantwortlich die Applikationen, welche die Daten verwenden, bei einem Richtlinienverstoß unterbrechen.
4	Role-restricted Data Usage	✓	-
5	Connector-restricted Data Usage	(✓)	Die Richtlinie könnte in D° umgesetzt werden. Im Kontext der International Data Spaces wird jede D°-Applikation innerhalb eines Connectors ausgeführt. Aus diesem Grund sollte der Connector überprüfen, ob eine Datennutzung erlaubt ist, und die Daten nicht erst an die Applikation weiterleiten, welche dann prüfen muss, ob eine Nutzung erlaubt ist. Die Richtlinie sollte zum frühestmöglichen Zeitpunkt überprüft werden.
6	System-restricted Data Usage	(✓)	Aus dem selben Grund wie die vorherige Richtlinie, sollte diese Richtlinie nicht in D° umgesetzt werden, auch wenn die theoretische Möglichkeit besteht.
7	Purpose-restricted Data Usage Policy	✓	-
8	Event-restricted Usage Policy	✓	-
9	Interval-restricted Data Usage	✓	-
10	Duration-restricted Data Usage	✗	Wie bereits zuvor beschrieben, kann diese Richtlinie nicht in D° umgesetzt werden. Sie erlaubt, dass Daten für eine bestimmte Zeit im System gespeichert und verwendet werden dürfen. Die Mechanismen zur Datennutzungskontrolle, welche in D° verwendet werden, werden nur während der Ausführung von D°-Applikationen angewandt.

			Somit überschreitet der von der Richtlinie erlaubte Nutzungszeitraum unter Umständen den Wirkungsbereich der D°-Applikation und somit der Mechanismen zur Datennutzungskontrolle. Aus diesem Grund ist eine Verwendung der Richtlinie in D° nicht möglich.
11	Location Restricted Policy	(✓)	Ebenso wie die Richtlinien 5 und 6 in dieser Tabelle, sollte diese Richtlinie durch den ausführenden Connector und nicht durch die Applikation umgesetzt werden, auch wenn dies möglich ist. Die geographische Position, an welcher sich das System befindet, auf dem die Applikation ausgeführt wird, ist kein originäres Attribut der Applikation. Das ausführende System kann diese Richtlinie bereits zu einem früheren Zeitpunkt umsetzen und so Overhead vermeiden.
12	Restricted Number of Usages	✓	-
13	Security Level Restricted Policy	(✓)	Im Kontext der International Data Spaces ist das Security Level ein Attribut, welches Connectoren beschreibt. Aus diesem Grund sollte der Connector Richtlinien auswerten und umsetzen, welche Anforderungen an den Security Level stellen. Eine möglichst frühe Umsetzung durch den Connector ist dabei der theoretischen Möglichkeit, die Richtlinie in D° umzusetzen, vorzuziehen,
14	Use data and delete it after	(✓)	Die Richtlinie, welche die Löschung der Daten nach ihrer Verwendung erfordert, kann auf Einzelfallbasis ggf. in D° umgesetzt werden. Eine D°-Applikation ist im Kontext der IDS nur ein einzelner Prozessschritt in einem Workflow eines Connectors. Aus diesem Grund müssen potentiell beliebig viele weitere Applikationen Aktionen ausführen, um diese Richtlinie umzusetzen. Diese Art von Richtlinien sollte auf der Ebene des Connectors umgesetzt werden, da dieser die Kontrolle und den Überblick über alle ausgeführten Applikationen besitzt.
15	Modify data (in transit)	✗	Diese Richtlinie kann in D° nicht umgesetzt werden. Der Grund hierfür ist, dass D°-Applikationen keinen Zugriff auf Daten im Transit haben.

			D°-Applikationen stellen einzelne Schritte in den Workflows, welche ein Connector umsetzen kann, bereit. Die Kommunikation zwischen den einzelnen Prozessschritten liegt dabei nicht in der Verantwortung der Applikation, sondern beim Connector, welcher diese Art von Richtlinien umsetzen muss.
16	Modify data (in rest)	✓	-
17	Local Logging	✓	-
18	Remote Notifications	✓	-
19	Attach Policy when Distribute to 3rd Party	✗	Diese Richtlinie erfordert es mit anderen Parteien Vereinbarungen darüber zu treffen, welche Richtlinien für die erhaltenen Daten eingehalten werden müssen. D° verfügt über keine Implementierung für ein Verfahren solche Vereinbarungen auszuhandeln, wie zum Beispiel die sog. Policy Negotiation, welche zwischen Connectoren verwendet wird. Aus diesem Grund kann diese Richtlinie nicht in D° umgesetzt werden.
20	Distribute only if encrypted	✓	-

4.2 Anteil der Einzelbeiträge an der Gesamtlösung

In diesen Abschnitt wird aufgezeigt, welchen Beitrag die einzelnen wissenschaftlichen Veröffentlichungen aus dem Kernbereich der Dissertation (s. Anhang) zur Gesamtlösung gehabt haben.

4.2.1 Eine Programmiersprache zur souveränen Datenverarbeitung

Der erste Beitrag zur vorliegenden Arbeit wurde auf der Konferenz D-A-CH Security 2018 veröffentlicht. Die Inhalte des Beitrages adressieren dabei primär die zweite Forschungsfrage der vorliegenden Arbeit. Es wird der Aufbau von D° als domänenspezifische Programmiersprache für datenverarbeitende Applikationen mit integrierten Usage Control Mechanismen beschrieben. Da der Beitrag neben dem Aufbau von D° auch den Aufbau von mit D° erzeugen Applikationen beschreibt, wird ein Gesamtüberblick geliefert. Dieser Überblick zeigt die Ziele von D° auf, ohne einzelne Aspekte im Detail zu behandeln. Der Beitrag adressiert in Teilen auch die vierte Forschungsfrage, da eine oberflächliche Ablaufbeschreibung des Remote Processings enthalten ist. Dabei wird der Remote Processing Prozess nur grob umrissen und der Ablauf beschrieben. Dieser Ablauf hat sich im Rahmen der weiteren Entwicklungen nur noch in Details geändert.

Demgegenüber hat sich der Aufbau von D° im Laufe der Entwicklungsarbeiten an manchen Stellen geändert. Beispielsweise wird in dem Beitrag eine eigene Laufzeitumgebung für D° vorgesehen, welche die Sandbox und ein Permission Center beinhaltet. Das Permission Center ist dabei der Vorläufer des Security Managers. Diese dedizierte Ausführungsumge-

bung für D° wurde im Laufe der Entwicklung aus D° entfernt. Stattdessen verfügt jede Applikation über die Elemente, welche neben der Applikationslogik notwendig sind, um die Applikation mit vollem Funktionsumfang auszuführen.

Ein Punkt, welcher in dem Beitrag beschrieben wurde, ist während der weiteren Entwicklungsarbeiten nicht weiter verfolgt worden: Die Überprüfung von Richtlinien zur Übersetzungszeit. Unter bestimmten Umständen ist es möglich Richtlinien für eine Applikation bereits zur Übersetzungszeit zu überprüfen. In diesen Fällen kann darauf verzichtet werden die Richtlinie zur Laufzeit zu überprüfen, falls die Richtlinie immer erfüllt ist. Dies resultiert in einem reduzierten Laufzeit-Overhead, welcher durch die integrierten Usage Control Mechanismen ansonsten nicht vermieden werden kann. Die Entwicklung der hierfür notwendigen statischen Codeanalysen wurde nicht weiter verfolgt. Stattdessen wurde der Fokus auf die Usage Control Mechanismen, welche während der Laufzeit angewendet werden, gelegt.

In dem Beitrag werden ebenfalls die erweiterbaren Systeme für Aktivitäten, Datentypen und Richtlinien vorgestellt, welche sich im weiteren Verlauf der Entwicklung nicht mehr geändert haben. Die Entscheidung, dass D° zunächst in eine Hostsprache übersetzt wird und nicht direkt in ein ausführbares Format, wurde ebenfalls in diesem Beitrag vorgestellt.

4.2.2 A Framework for Creating Policy-agnostic Programming Languages

Der nächste Beitrag wurde auf der DATA 2020 veröffentlicht. Der Fokus des Beitrags liegt dabei auf den Usage Control Mechanismen, welche in D° umgesetzt werden, und adressiert die zweite und die dritte Forschungsfrage. Dabei kann nicht klar festgelegt werden, welche Forschungsfrage primär durch den Beitrag adressiert wird. Dies liegt daran, dass sich die Antworten auf einzelne Aspekte der jeweiligen Forschungsfrage auch direkt auf die andere Forschungsfrage anwenden lassen.

Der Beitrag präsentiert den Aufbau und die Funktionsweise aller Konzepte und Elemente des Usage Control Mechanismus von D°. Darunter werden auch die Interaktionen und Abhängigkeiten zwischen den einzelnen Konzepten und Elementen aufgezeigt. Hierdurch ergibt sich eine vollständige Beschreibung des Usage Control Mechanismus, welcher von D° verwendet wird.

Es wird ausführlich aufgezeigt, wie D° das Programmierparadigma der policy-agnostischen Programmierung umsetzt. Zu diesem Zweck wird die Trennung zwischen Definitionen und Instanzen von Sprachelementen beschrieben. Außerdem wird erläutert, wie die einzelnen Sprachelemente vor dem Einsatz in D°-Applikationen miteinander kombiniert werden können, um Usage Control in Applikationen einzubetten. Darüber hinaus wird die Erweiterbarkeit von D° im Bereich der Aktivitäten, Datentypen und Richtlinien beschrieben. Es wird aufgezeigt, welche Schritte unternommen werden müssen, um in den einzelnen Bereichen neue Elemente zu erzeugen. Dabei wird sowohl auf die Erstellung von Definitionen innerhalb von D°, als auch die Entwicklung der verknüpften Implementation eingegangen.

Ebenfalls wird in dem Beitrag der Codegenerator, welcher Teil des D°-Compilers ist, betrachtet. Es wird dargestellt, wie die Transformation von D°-Code in Programmcode der verwendeten Hostsprache abläuft. Dabei liegt der Fokus der Darstellung darauf, wie sichergestellt wird, dass die Richtlinien, welche mit den einzelnen Sprachelementen verbunden sind, an allen notwendigen Stellen korrekt überprüft werden. Außerdem werden die

beiden zentralen Usage Control Konzepte von D°, die Sandbox und der Security Manager, beschrieben. Dies umfasst sowohl den Aufbau, als auch die Funktionalität.

Darüber hinaus enthält der Beitrag ausführliche Beschreibungen darüber, wie Richtlinien in D° aufgebaut sind. Dabei wird die Schnittstelle von Richtlinien beschrieben, welche die einzelnen Aspekte der Überprüfungslogik bereitstellen. Es wird beschrieben, zu welchen Zeitpunkten die einzelnen Endpunkte dieser Schnittstelle ausgewertet werden und für welche Zwecke sie verwendet werden können. Da D° die gleichzeitige Verwendung von Black- und Whitelisting ermöglicht, wird in dem Beitrag der strukturierte Prozess, welcher Greylisting heißt und zu diesem Zweck verwendet wird, vorgestellt. Greylisting wird innerhalb von D° verwendet, um Konflikte aufzulösen, welche bei der Auswertung der dynamischen Anteile von mehreren Richtlinien auftreten können.

Die Inhalte des Beitrags haben im weiteren Verlauf der Entwicklungen Bestand gehabt. In einem späteren Beitrag wurden noch Erweiterungen des Usage Control Systems vorgenommen. Dabei wurden die bestehenden Konzepte und Elemente jedoch nicht modifiziert, sondern lediglich erweitert.

4.2.3 Utilizing Remote Evaluation for Providing Data Sovereignty in Data-sharing Ecosystems

Der dritte Konferenzbeitrag wurde auf der HICSS 54 veröffentlicht. Die Inhalte dieses Beitrags adressieren vollständig die Beantwortung der vierten Forschungsfrage. In dem Beitrag wird das Remote Processing System spezifiziert.

Dabei werden verschiedene Aspekte des Remote Processing Systems innerhalb des Beitrags aufgegriffen. Zum einen findet eine ausführliche Beschreibung und Darstellung der Architektur statt. Diese umfasst neben den Komponenten, welche direkt in das System integriert, bzw. für das System entwickelt wurden, auch externe Systeme, welche das Remote Processing System während der Ausführung verwendet. Neben den einzelnen Komponenten des Remote Processing Systems werden ebenfalls die Interaktionen zwischen den einzelnen Komponenten und interne Prozesse beschrieben.

Der Beitrag beschreibt außerdem eine Zustandsmaschine, welche den Gesamtprozess, welcher vom Remote Processing System realisiert wird, darstellt. Dabei wird der beschriebene Prozess während der Ausführung gemeinsam vom Betreiber des Remote Processing Systems und dem Bereitsteller der Applikation durchlaufen und Zustandsübergange ausgelöst.

Das Remote Processing System wurde so entwickelt, dass es zu großen Teilen automatisiert betrieben werden kann. Das heißt, abseits der Aufrufe durch den Systembetreiber und den Bereitsteller der Applikation, welche Zustandsübergänge auslösen, soll es keine Notwendigkeit für manuelle Eingriffe bzw. Maßnahmen im Prozess geben. Die einzige Ausnahme, welche sich im Prozess befindet, ist die Möglichkeit für den Systembetreiber ein manuelles Audit des Applikationscodes durchzuführen. Im Kontext des automatisierten Ablaufs beschreibt der Beitrag, wie den D°-Applikationen auf eine strukturierte Art und Weise Zugriff auf die Daten gewährt werden kann.

Darüber hinaus werden in dem Beitrag die zusätzlichen Sicherheitsmechanismen beschrieben, welche die zu verarbeitenden Daten und die Systeme des Remote Processing Betreibers zusätzlich schützen sollen. Es wird aufgezeigt, wie die D°-Applikationen verpackt und ausgeführt werden. Dies schafft eine Isolation der ausgeführten Applikation vom

ausführenden System und ermöglicht zusätzliche Kontrollmechanismen. Beispielsweise die Einschränkung von verwendeten Ressourcen.

Außerdem wird beschrieben, wie das Remote Processing System die ausgeführten Applikationen von den Bereitstellern isoliert. Das Remote Processing System nimmt Aufrufe der Applikationsbereitsteller entgegen und leitet diese an die Applikation weiter.

Die Arbeiten am Remote Processing sind mit der Veröffentlichung abgeschlossen gewesen. Somit beschreibt der Beitrag den finalen Arbeitsstand des Remote Processing Systems. Auch im Nachgang fanden keine Erweiterungen oder Änderungen mehr statt.

4.2.4 A Policy-agnostic Programming Language for the International Data Spaces

Bei der letzten Veröffentlichung, welche einen Beitrag zum Kernbereich der vorliegenden Arbeit hat, handelt es sich um einen Artikel in der Buchreihe „Communications in Computer and Information Science“ von Springer. Es handelt sich bei dem Artikel um eine erweiterte Fassung des Beitrags auf der DATA 2020. Die Inhalte in dem Artikel adressieren die Forschungsfragen zwei und drei.

In dem Artikel wird das Usage Control System von D° erweitert. Es wird an verschiedenen Stellen von D° die Möglichkeit hinzugefügt Tags zu verwenden. Diese Tags beschreiben entweder Aktionen, die ausgeführt werden sollen, oder den aktuellen Zustand von Datensätzen. Durch diese Tags vergrößert sich die Menge an umsetzbaren Richtlinien, was die Mächtigkeit des Usage Control Mechanismus insgesamt steigert.

Darüber hinaus wird ein Mechanismus präsentiert, welcher es erlaubt externe Identifier für Daten innerhalb von D° zu verwenden. Dies ist eine wichtige Funktionalität, wenn es darum geht D° in Umgebungen einzusetzen, in denen mehrere Instanzen von Usage Control Mechanismen verwendet werden, und es notwendig ist einzelne Datensätze innerhalb des Szenarios eindeutig zu identifizieren. Dabei muss es sich bei den verschiedenen Usage Control Mechanismen nicht zwingend um unterschiedliche Lösung handeln. Die Möglichkeit zur eindeutigen Identifizierung über die Grenze einer einzelnen D°-Applikation ist auch in einem mehrschrittigen Prozess, welcher nur aus D°-Applikationen besteht, von Relevanz.

Durch diese Erweiterung wird es möglich Richtlinien zu verwenden, welche einen größeren Geltungsbereich als einzelne D°-Applikationen haben. Hierzu werden die notwendigen Informationen zur Überprüfung von Richtlinien in einem zentralen Dienst verwaltet. Richtlinien können diese Daten unter Verwendung des eindeutigen Identifiers abfragen und zur Richtlinienüberprüfung verwenden. Dies ist zum Beispiel notwendig, wenn die Häufigkeit bestimmter Aktionen durch Richtlinien limitiert werden soll. Ein mögliches Beispiel für eine solche Richtline wäre, dass Daten nur einmalig ausgedruckt werden dürfen.

Außerdem wird in dem Artikel ein Gesamtüberblick über den finalen Entwicklungsstand von D° gegeben. Dabei wird auch beschrieben, wie der D°-Compiler aus der Konfiguration, welche Teil des Programmcodes ist, ableitet, welche Schnittstellen die erzeugte Applikation bereitstellen soll.

Der Artikel enthält des Weiteren Beschreibungen, wie D° in den International Data Spaces (IDS), welche ein wichtiges Einsatzgebiet für D° sind, eingesetzt werden kann. Dabei werden die Richtlinien, welche in den IDS verwendet werden, in Gruppen unterteilt und hinsichtlich ihrer Umsetzbarkeit in D° beurteilt.

Die Konzepte und Mechanismen, welche in dem Artikel beschrieben werden, repräsentieren dabei den finalen Stand von D°. Im Nachgang fanden keine weiteren Arbeiten statt, die Änderungen bewirkt haben.

KAPITEL 5

Demonstration

In diesem Kapitel wird ein Demonstrator präsentiert, welcher D° zur Integration von Usage Control Mechanismen verwendet. Es wird beschrieben, wie D° sich in das Szenario einfügt und sichergestellt wird, dass die Richtlinien, welche in dem System umgesetzt werden sollen, durch D° durchgesetzt werden.

Dabei werden zunächst die Rollen, welche im Szenario relevant sind, beschrieben. Anschließend wird ein allgemeiner Überblick über das Szenario gegeben. Außerdem werden Anforderungen definiert, welche im Bereich der Datennutzungskontrolle umgesetzt werden müssen. Darauf folgt eine Beschreibung der Webapplikation, welche zur Nutzerinteraktion verwendet wird. Im Anschluss konzentrieren sich die Beschreibungen auf die Anwendung von D° im Demonstrator und die Umsetzung der Datennutzungskontrolle durch D°. Des Weiteren wird eine der eingesetzten D°-Applikationen detaillierter betrachtet. Anhand von Beispielen wird aufgezeigt, wie Sprachelemente für D° entwickelt und verwendet werden können. Ebenfalls wird erläutert, wie durch den Compiler sichergestellt wird, dass Applikationslogik und Datennutzungskontrolle in der ausführbaren Applikation fest miteinander verwoben sind. Abschließend wird ein Beispiel präsentiert, welches darstellt, wie sich verwendeten Richtlinien zur Laufzeit bemerkbar machen.

5.1 Darstellung des Szenarios

Der Demonstrator bildet ein Szenario ab, welches den Kauf von 3D Modellen, sowie den anschließenden Druck auf 3D Druckern von Serviceanbietern umfasst. Dabei gibt es drei Rollen, die in dem Szenario von Relevanz sind.

5.1.1 Modellersteller

Die erste Rolle innerhalb des Szenarios ist der Modellersteller. Inhaber dieser Rolle erzeugen 3D Modelle mit der Absicht diese zu verkaufen beziehungsweise zu verteilen. Dabei hat der Modellersteller unterschiedliche Möglichkeiten die 3D Modelle zu erzeugen. Neben Software für die 3D Modellierung (z.B. CAD) können 3D Scanner verwendet werden, um 3D Modelle von realen Objekten zu erzeugen.

Der Verkauf der 3D Modelle kann dabei über einen Webshop erfolgen. Der Modellersteller ist für die von ihm erzeugten Modelle der Rechteinhaber. Unter Umständen möchte der Modellersteller einschränken, wie bzw. in welchem Umfang die Käufer die gekauften Modelle verwenden können. Beispielsweise möchte der Modellersteller unter Umständen sicherstellen, dass das 3D Modell nach dem Kauf nur einmalig gedruckt wird.

5.1.2 Kunde

Die verfügbaren 3D Modelle, welche beispielsweise im Webshop zum Verkauf stehen, werden von Kunden bezogen. Dabei möchten die Kunden die gekauften 3D Modelle mit einem 3D Drucker ausdrucken, verfügen allerdings über keinen eigenen 3D Drucker. Diese Kunden sind die zweite Rolle in dem Szenario.

Ebenso wie Modellersteller, können Kunden ein Interesse daran haben Einschränkungen für die Nutzung von bezogenen 3D Modellen zu definieren. Handelt es sich bei einem 3D Modell um ein mechanisches Bauteil, welches bestimmte Eigenschaften erfüllen muss, möchte der Kunde ggf. Einschränkungen definieren, welche Druckeigenschaften oder verwendete Materialien betreffen.

5.1.3 Betreiber

Die dritte Rolle im vorgestellten Szenario ist der Betreiber. Der Betreiber ist die einzige Rolle, welche über 3D Drucker verfügt. Dabei ist das Ausdrucken von 3D Modellen die Serviceleistung, welche der Betreiber für die Kunden zur Verfügung stellt.

Der Betreiber muss sicherstellen, dass er die von Modellersteller und Kunden definierten Richtlinien, welche an 3D Modellen hängen, einhält. Um seinen Service zu erbringen, verwendet der Betreiber unterschiedliche Softwareprodukte, unter anderem zur Ansteuerung der 3D Drucker und zur Interaktion mit Kunden. Dabei setzt der Betreiber mit D° entwickelte Software, um die korrekte Umsetzung der Richtlinien sicherzustellen.

5.1.4 Gesamtüberblick

Abbildung 5.1 enthält eine schematische Darstellung des Demonstrators. Sie führt zudem die drei beschriebenen Rollen und die von ihnen verwendeten Komponenten auf. Ebenso sind die Interaktionen zwischen den einzelnen Komponenten und die Weitergabe der durchzusetzenden Richtlinien innerhalb des Demonstrators von System zu System erkennbar.

Die Abbildung zeigt, dass D° nur bei dem 3D Druck Service eingesetzt wird, welcher die zentrale Komponente des Betreibers darstellt. Somit ist der 3D Druck Service alleine für die Umsetzung der Richtlinien verantwortlich; die anderen Komponenten im Demonstrator sind nicht an der Umsetzung beteiligt. Ebenso kann der Abbildung entnommen werden, dass der Betreiber Digital Twins für die verwendeten 3D Drucker einsetzt. Diese enthalten Informationen über den aktuellen Zustand der eingesetzten 3D Drucker und werden sowohl für die Ablaufsteuerung, als auch die Umsetzung von Richtlinien verwendet. Es ist erkennbar, dass zur Steuerung der 3D Drucker spezielle Software verwendet wird. Diese 3D Druck Server werden vom 3D Druck Service verwendet, beispielsweise um Druckaufträge zu starten. Ebenfalls ist erkennbar, dass der Betreiber einen Policy Information Point (PIP) einsetzt. Der PIP liefert Informationen darüber, wie oft jedes (dem Betreiber bekannte) 3D Modell bereits gedruckt wurde.

Die Abbildung zeigt das 3D Druck Portal, welches vom Kunden verwendet wird, um 3D Modelle zu verwalten, Druckaufträge zu starten, Richtlinien zu betrachten und zu erstellen. Das 3D Druck Portal wird in der Abbildung getrennt vom Policy Editor dargestellt. In der Umsetzung ist der Policy Editor ein Bestandteil des 3D Druck Portals. Die getrennte Darstellung in der Abbildung liegt darin begründet, dass der Fluss von Richtlinien im

Demonstrator so besser dargestellt werden kann. Darüber hinaus ist der Policy Editor eine zentrale Funktionalität des 3D Druck Portals, welche so besser hervorgehoben werden kann.

Aus dem selben Grund wird der Fluss von 3D Modellen und Richtlinien in der Abbildung getrennt voneinander dargestellt. In der Implementation des Demonstrators werden ausschließlich Dateiarchive verwendet, welche sowohl das 3D Modell, als auch die Richtlinien, die für das Modell gelten, enthalten.

5.1.5 Anforderungen an die Datennutzungskontrolle

In diesem Abschnitt findet eine Auflistung der Anforderung an die Datennutzungskontrolle im Szenario statt. Hierdurch wird die Möglichkeit geschaffen zu bewerten, ob die Anforderungen durch die Verwendung von D° erfüllt worden sind. Ebenfalls kann angegeben werden, welche Anforderungen einen Einfluss auf unterschiedliche Aspekte der Umsetzung haben.

Anforderung 1 – Im Demonstrator können Richtlinien definiert und umgesetzt werden, welche die verwendeten Druckeinstellungen, Druckmaterialien und die maximale Anzahl von Drucken für 3D Modelle einschränken.

Anforderung 2 – Der Modellersteller kann Richtlinien für 3D Modelle definieren, welche gemeinsam mit dem Modell an die Kunden ausgeliefert werden.

Anforderung 3 – Der Kunde kann zusätzliche Richtlinien für verwendete 3D Modelle definieren, solange diese nicht den Richtlinien des Modellerstellers widersprechen.

Anforderung 4 – Der Betreiber druckt 3D Modelle nur dann aus, wenn die anhängigen Richtlinien erfüllt sind.

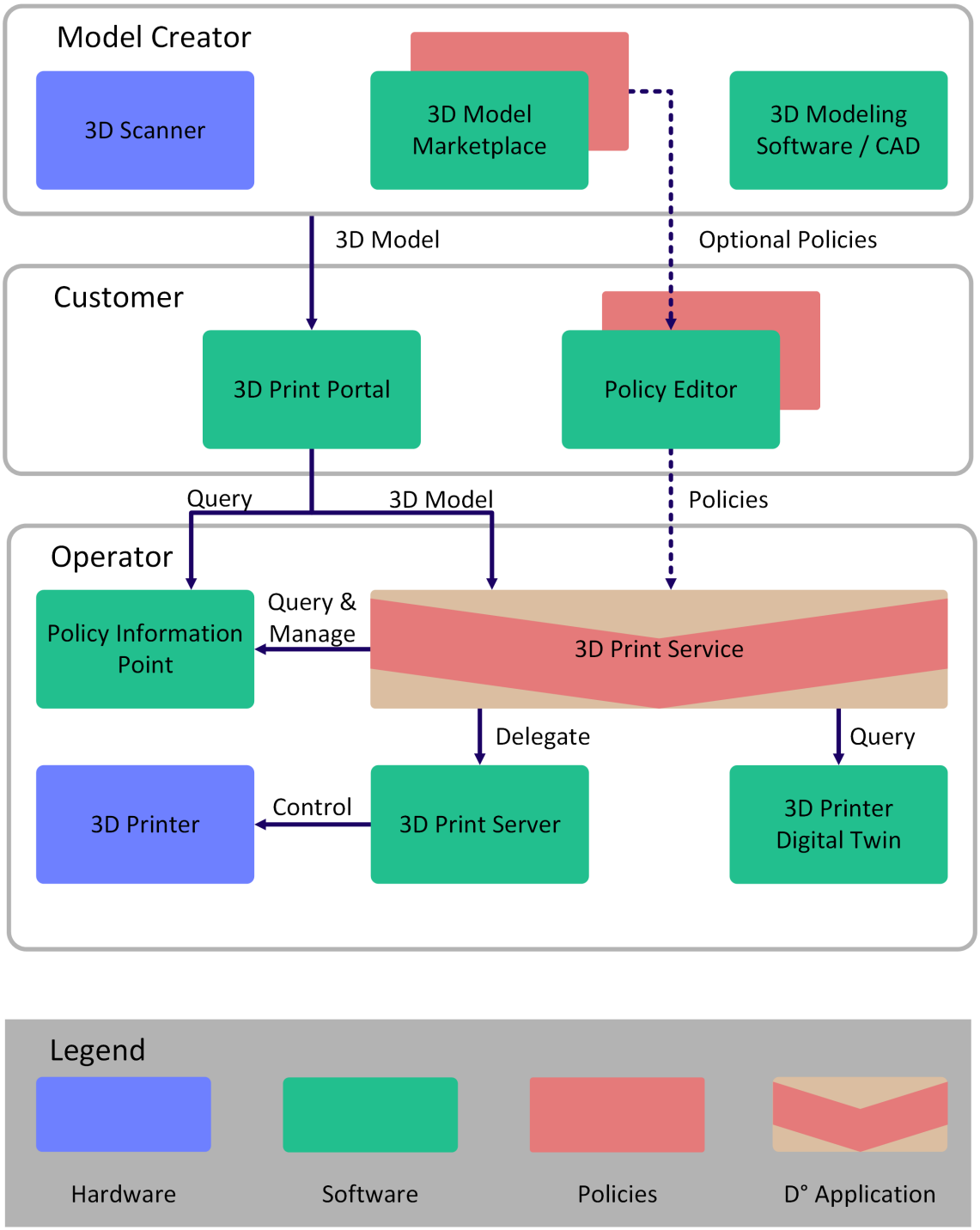

Abbildung 5.1: Schematische Darstellung, welche alle Komponenten des Demonstrators, sowie die Interaktionen dieser Komponenten darstellt. Angelehnt an: [Bru21a]

5.2 Beschreibung des 3D Druck Portals

Im Rahmen des Demonstrators verbindet das 3D Druck Portal den Kunden mit dem Betreiber. Das Portal wird durch den Kunden verwendet, um Anfragen an den 3D Druck Service zu senden und 3D Modelle zu verwalten. Bei dem 3D Druck Portal handelt es sich um eine Webanwendung, welche über ein Vue.js Frontend und ein Spring Backend verfügt. Dabei wurde das Frontend als Single-page-Webanwendung umgesetzt.

Das Portal bietet dem Kunden verschiedene Funktionen. Gekaufte 3D Modelle (inklusive der anhängigen Richtlinien) können im Portal hinterlegt werden. Dies ist notwendig, um Druckaufträge aus 3D Modellen zu erstellen, und somit ein erforderlicher Schritt in dem Prozess, welcher zum Drucken von 3D Modellen durchlaufen werden muss. Sobald ein 3D Modell geöffnet wird, wird es dem Nutzer präsentiert. Dies umfasst zum einen eine interaktive Darstellung des eigentlichen 3D Modells und zum anderen einen Policy Editor. Dieser Editor gibt lesenden Zugriff auf die vom Rechteinhaber erstellten Richtlinien. Außerdem ermöglicht der Editor die Definition von zusätzlichen Richtlinien, welche nicht im Konflikt mit den Richtlinien des Rechteinhabers stehen. Hierdurch werden Anforderungen 1 und 3 aus Abschnitt 5.1.5 adressiert.

Sofern ein Projekt ausgewählt wurde, wird dem Anwender darüber hinaus angezeigt, welche 3D Drucker und Druckmaterialien beim Betreiber zur Verfügung stehen und welche Druckeinstellungen und Slicer vom Betreiber unterstützt werden. Der Kunde kann aus diesen angezeigten Elementen eine Auswahl treffen, um beim Betreiber die Erstellung eines Druckauftrags anzufragen.

Außerdem werden im Portal die erstellten Druckaufträge angezeigt. Dabei werden diverse Informationen über den jeweiligen Druckauftrag angezeigt. Sofern der Auftrag noch nicht in eine Warteschlange eingereiht wurde, wird dem Anwender eine Möglichkeit geboten den Druck in die Warteschlange einzureihen.

Abbildung 5.2 zeigt einen Screenshot der Benutzeroberfläche des 3D Druck Portals, direkt nach dem Aufruf der Weboberfläche. Im linken Bereich ist die Liste der im Portal hinterlegten 3D Modelle. Auf der rechten Seite befindet sich die Liste mit den erstellten Druckaufträgen. Der mittlere Bereich zeigt nur eine Meldung darüber an, dass bisher kein Projekt geöffnet wurde. Am oberen Rand ist über die gesamte Breite eine Applikationsleiste platziert, welche Informationen über das aktuell geöffnete Projekt beinhaltet und den Zugriff auf die Einstellungen ermöglicht. In diesen Einstellungen können die Endpunkte der verwendeten Systeme des Betreibers hinterlegt und kleinere Anpassungen an der Anzeige vorgenommen werden.

In Abbildung 5.3 ist ein weiter Screenshot der Benutzeroberfläche zu sehen. Die Abbildung zeigt den Zustand der Oberfläche nach dem Öffnen eines Projekts. Im rechten Bereich ist weiterhin die Liste der Druckaufträge zu sehen. Auf der linken Seite ist die Liste der verfügbaren Projekte verschwunden. Stattdessen befindet sich dort nun eine Anzeige, welche aus mehreren Seiten besteht, und dem Anwender ermöglicht die Parameter (z.B. Druckeinstellungen) zu wählen, welche verwendet werden, wenn aus dem geöffneten 3D Modell ein Druckauftrag erzeugt wird. Der rote Knopf im rechten Bereich der Applikationsleiste erlaubt es das aktuell geöffnete Projekt wieder zu schließen. Wird diese Aktion ausgeführt, kehrt die Oberfläche in den Zustand, welcher in Abbildung 5.2 gezeigt ist, zurück. Im

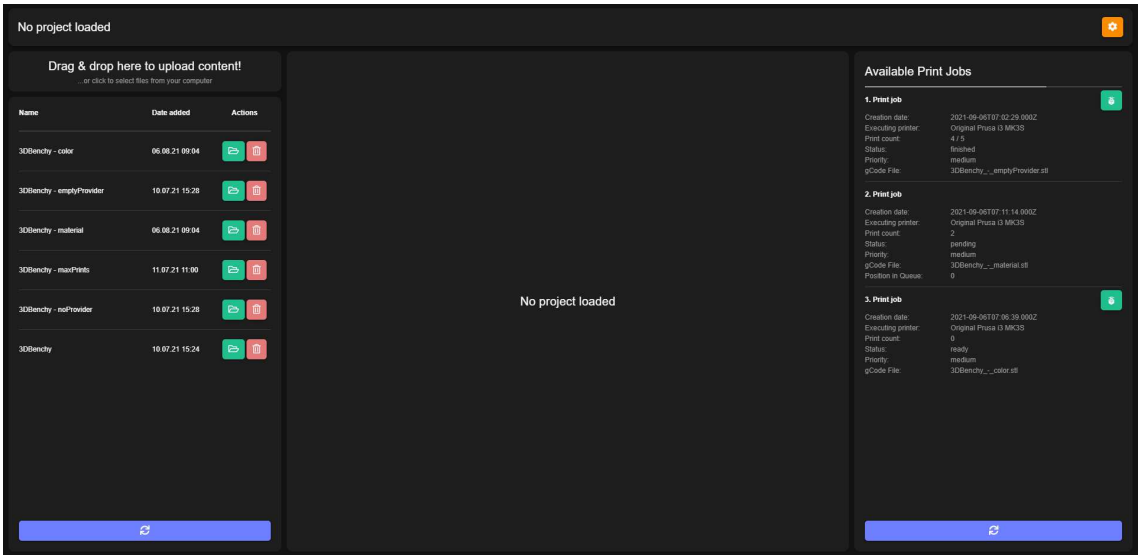

Abbildung 5.2: Screenshot der Benutzeroberfläche des 3D Druck Portals direkt nach dem Aufruf und ohne ein geöffnetes Projekt.

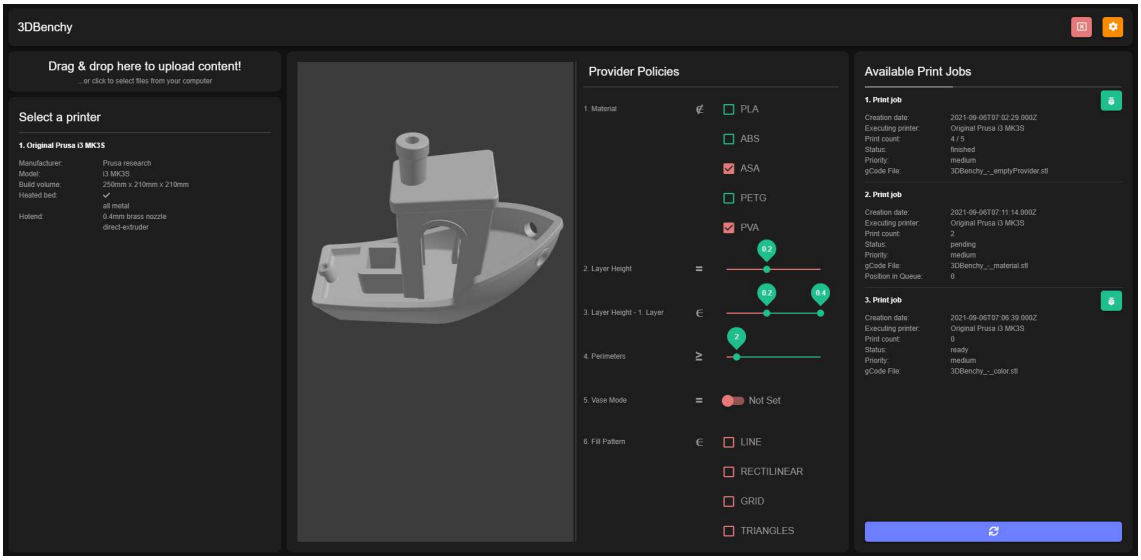

Abbildung 5.3: Screenshot der Benutzeroberfläche des 3D Druck Portals nachdem ein Projekt geöffnet wurde.

mittleren Bereich der Oberfläche ist nun eine interaktive Darstellung des 3D Modells und der Policy Editor zu sehen. Sind alle notwendigen Parameter ausgewählt worden, können diese gemeinsam mit dem 3D Modell und den Policies, welche im Policy Editor angezeigt werden, an den Betreiber übermittelt werden, um einen neuen Druckauftrag zu erzeugen. Dieser taucht anschließend in der Liste auf der rechten Seite auf.

Um den Anwender über die Ergebnisse von ausgelösten Aktionen (– beispielsweise das Einreihen eines Druckauftrags in die Warteschlange –) zu informieren, werden Benachrichtigungen verwendet. Diese Benachrichtigungen sind abhängig von ihrem Typ eingefärbt. Allgemeine Informationen (bspw. Öffnen eines 3D Modells) werden in blau angezeigt. Erfolgreich ausgeführte Aktionen (bspw. Erstellen eines Druckauftrags) werden grün und fehlgeschlagene rot eingefärbt.

In den nachfolgenden Abschnitten werden die einzelnen Komponenten der Oberfläche näher beschrieben.

5.2.1 Projektauswahl

Sofern in der Benutzeroberfläche des 3D Druck Portals kein 3D Modell geöffnet wurde, befindet sich im linken Bereich eine Anzeige, welche eine Liste aller hinterlegten 3D Modelle enthält. Dabei wird für jedes 3D Modell der Name und der Zeitpunkt, zu dem das Modell dem 3D Druck Portal hinzugefügt wurde, angezeigt. Darüber hinaus werden für jedes 3D Modell zwei Buttons angezeigt. Diese ermöglichen es das Modell zu öffnen oder aus dem 3D Druck Portal zu entfernen. Über einen weiteren Button, welcher am unteren Ende der Anzeige platziert ist, kann ein Neuladen der Liste ausgelöst werden. Die Daten werden dabei aus dem Backend des 3D Druck Portals geladen.

Neue 3D Modelle können dem Portal auf zwei Wegen hinzugefügt werden. Über der Listenanzeige befindet sich ein weiteres Feld. Werden Dateien im korrekten Format per Drag'n'Drop in dem Feld abgelegt, werden sie dem Portal hinzugefügt. Alternativ kann ein Dateiexplorer verwendet werden, um Dateien zum Hinzufügen auszuwählen. Der Explorer wird durch einen Klick auf das Feld geöffnet.

Abbildung 5.4 zeigt einen Screenshot, welcher den Bereich der Benutzeroberfläche zeigt, der die Liste der hinterlegten 3D Modelle darstellt.

5.2.2 Auswahl der Parameter für Druckaufträge

Nachdem ein 3D Modell geöffnet wurde, verschwindet die Liste der verfügbaren 3D Modelle. An dieser Stelle werden daraufhin nacheinander verschiedene Listen von Objekten angezeigt, welche im 3D Druck Service des Betreibers hinterlegt sind. Aus jeder dieser Liste muss ein Element ausgewählt werden, um für das geöffnete 3D Modell einen Druckauftrag erzeugen zu können.

Bevor dem Anwender Objekte zur Auswahl angezeigt werden, sendet das 3D Druck Portal eine Anfrage an den 3D Druck Service des Betreibers. Der Service sammelt alle relevanten Daten und übermittelt diese an das Portal. Nacheinander muss der Anwender dann jeweils ein Element aus den folgenden Bereichen auswählen: 3D Drucker, Filament, Druckeinstellungen, Slicer. Dabei kann der Anwender jederzeit zur vorherigen Liste zurückkehren, um seine Auswahl zu ändern.

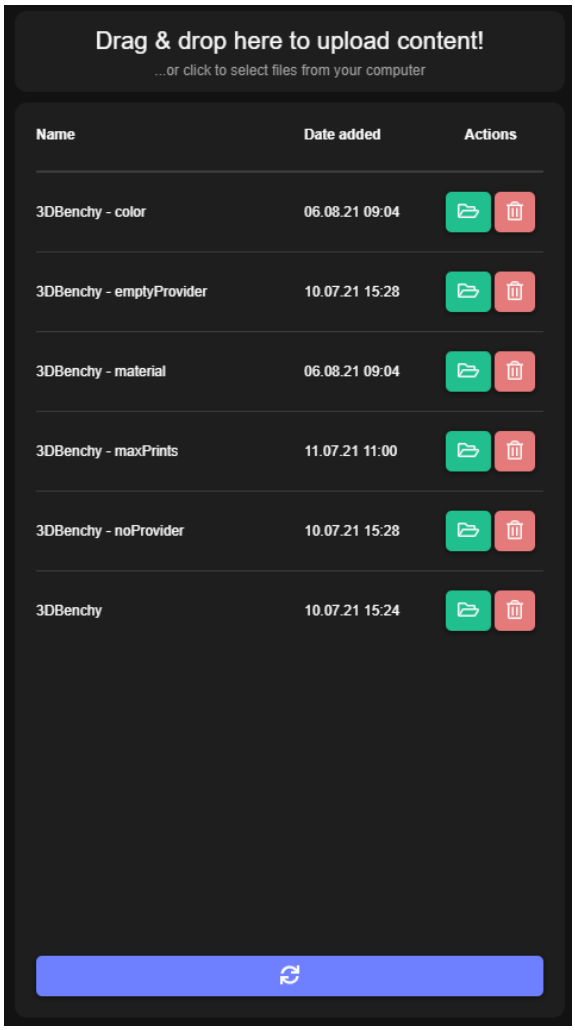

Abbildung 5.4: Screenshot, welcher die Liste der 3D Modelle, welche im 3D Druck Portal hinterlegt sind, zeigt. Die Anzeige ist im linken Bereich der Benutzeroberfläche sichtbar, solange kein Projekt geöffnet ist.

Sobald der Anwender ein Objekt aus allen Kategorien ausgewählt hat, wird eine Zusammenfassung gezeigt, welche alle ausgewählten Objekte präsentiert. In der Anzeige der Zusammenfassung befindet sich darüber hinaus ein Button, welcher es erlaubt alle Daten an den 3D Druck Service zu senden, um einen neuen Druckauftrag anzulegen. Die dabei übermittelten Daten enthalten neben den zuvor ausgewählten Objekten das 3D Modell, sowie die anhängigen Richtlinien. Über das Ergebnis der Erstellung des Druckauftrags wird der Anwender über die zuvor beschriebenen Benachrichtigungen im rechten Bereich der Benutzeroberfläche informiert.

Abbildung 5.5 zeigt die zuvor beschriebenen Seiten, welche von der Anzeige verwendet werden. Dabei entspricht die Reihenfolge der Seiten in Leserichtung der Reihenfolge, in welcher die Seiten in der Benutzeroberfläche angezeigt werden. Die Abbildung zeigt, dass

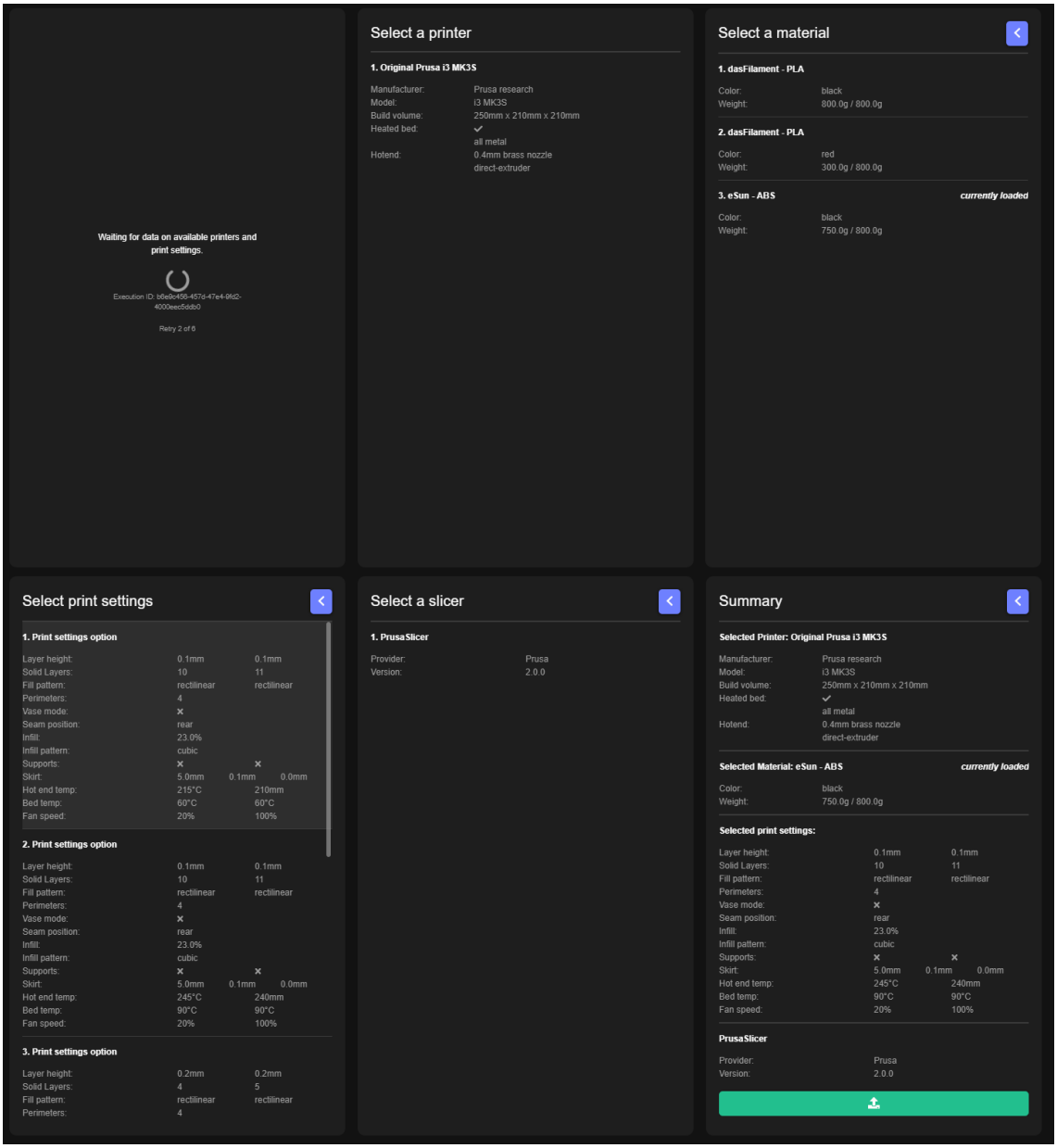

Abbildung 5.5: Screenshots, welche die verschiedenen Seiten zeigen, über die der Nutzer des 3D Druck Portals Objekte für die Erstellung von Druckaufträgen auswählt. Die Screenshots sind in Leserichtung in der Reihenfolge angeordnet, in welcher sie auch in der Applikation verwendet werden.

die Seiten, welche Objekte zur Auswahl präsentieren, detaillierte Informationen zu den vorhandenen Objekten anzeigt. Darüber hinaus ist der Abbildung zu entnehmen, dass diese Informationen ebenfalls in der Zusammenfassung präsentiert werden.

5.2.3 Modellbetrachter

Sobald im 3D Druck Portal durch den Kunden ein 3D Modell geöffnet wird, erscheint im mittleren Bereich der Nutzeroberfläche auf der linken Seite eine interaktive Darstellung des 3D Modells. Der Anwender kann das angezeigte 3D Modell durch Klicken und Bewegen der Maus rotieren, verschieben, vergrößern und verkleinern.

Diese Anzeige erlaubt es dem Anwender sicherzustellen, dass er das gewünschte 3D Modell geöffnet hat, bevor er einen entsprechenden Druckauftrag erstellt. Darüber hinaus hat die Anzeige keinen weiteren Beitrag zur Funktionalität des 3D Druck Portals.

Abbildung 5.6 zeigt einen Screenshot, welche die Anzeige für 3D Modelle enthält. Auf der rechten Seite ist eine vertikale Trennlinie erkennbar, welche verwendet werden kann, um die Anzeige innerhalb des übergeordneten Containers zu vergrößern bzw. zu verkleinern. Der übergeordnete Container nimmt den gesamten mittleren Bereich der Benutzeroberfläche ein und enthält neben der Anzeige für 3D Modelle auch den Policy Editor.

5.2.4 Policy Editor

Sofern ein 3D Modell im 3D Druck Portal geöffnet ist, wird neben dem Modellbetrachter der Policy Editor angezeigt. Dabei ist der Policy Editor in zwei Segmente unterteilt.

Das erste (obere) Segment zeigt die Richtlinien an, welche durch den Rechteinhaber des geöffneten 3D Modells definiert wurden und mit dem Modell verknüpft sind. Dieses

Abbildung 5.6: Screenshot, welcher die interaktive Anzeige für 3D Modelle zeigt. Die Anzeige ist im mittleren Bereich der Benutzeroberfläche sichtbar, wenn ein 3D Modell geöffnet ist.

Segment bietet nur einen lesenden Zugriff auf die angezeigten Richtlinien. Hierdurch wird sichergestellt, dass die Richtlinien des Modellerstellers nicht modifiziert werden.

Das zweite Segment erlaubt es dem Kunden zusätzliche Richtlinien zu definieren. Dabei können nur Richtlinien zu Attributen definiert werden, welche nicht bereits durch die Richtlinien des Rechteinhabers abgedeckt werden. Hierdurch werden Konflikte zwischen den einzelnen Richtlinien verhindert und die Anforderungen 1 und 3 aus Abschnitt 5.1.5 adressiert. Gleichzeitig entfällt durch dieses Vorgehen die Möglichkeit Richtlinien des Rechteinhabers weiter zu verschärfen.

Der Nutzer des 3D Druck Portals kann für jedes Attribut einzeln festlegen, ob eine Richtlinie verwendet werden soll oder nicht. Wird die Richtlinie für ein einzelnes Attribut aktiviert, kann der Nutzer auswählen, welche Art von Richtlinie verwendet werden soll (– beispielsweise, dass ein gewisser Schwellwert nicht über- oder unterschritten werden darf –). Außerdem kann der Nutzer die individuellen Parameter der Richtlinie (– beispielsweise den Schwellwert –) festlegen.

Für jede Richtlinie wird angezeigt, auf welches Attribut Bezug genommen wird (bspw. Materialfarbe). Ebenfalls wird angezeigt, welche Art von Richtlinie vorliegt. Beispiele für mögliche Arten sind, dass eine Schwelle nicht überschritten wird oder der Wert in einem gewissen Intervall liegt. Darüber hinaus werden für jede Richtlinie die erlaubten und verbotenen Werte, bzw. Schwellwerte, oder Intervallgrenzen angezeigt. Dabei sind erlaubte Werte in grün markiert und verbotene in rot. Während die Richtlinien des Rechteerstellers nur lesend dargestellt werden, können die Richtlinien im Bereich des Kunden de-/aktiviert, sowie die Art der Richtlinie und relevante Parameter angepasst werden.

Abbildung 5.7 zeigt jeweils einen Screenshot für die unterschiedlichen Policy Editor Segmente. Auf der linken Seite befindet sich das Segment, welches zur lesenden Anzeige der Richtlinien des Modellerstellers verwendet wird. Die zuvor beschriebenen Elemente der Anzeige können in den drei Spalten identifiziert werden. Die erste Spalte enthält den Namen des betroffenen Attributs inkl. eines Tooltips, welcher sichtbar wird, wenn die Maus über den Namen fährt. In Spalte zwei findet sich ein Symbol, welches anzeigt, um welche Art von Richtlinie es sich handelt. Die dritte Spalte enthält dann die konkreten Werte der Richtline.

Auf der rechten Seite der Abbildung ist das Segment zur Anzeige und Bearbeitung von Richtlinien des Kunden zu sehen. Der Aufbau ist identisch zu der Anzeige für Richtlinien des Modellerstellers. Allerdings findet sich vor den drei bekannten Spalten eine zusätzliche Spalte. Diese enthält eine Checkbox, welche es erlaubt die Regeln für jedes Attribut einzeln zu aktivieren oder deaktivieren. Die Art der Richtlinie, sowie ihre konkreten Parameter können für aktivierte Richtlinien verändert werden.

5.2.5 Liste der Druckaufträge

Auf der rechten Seite der Benutzeroberfläche wird durchgehend eine Liste der erstellten Druckaufträge gezeigt. Dabei werden in der Liste diverse Informationen für jeden Druckauftrag bereitgestellt. Für jeden Druckauftrag wird angezeigt, wann er erstellt wurde und welche 3D Modelle verwendet wurden, um den Auftrag zu erstellen. Hierdurch können die einzelnen Druckaufträge leicht unterschieden und den unterschiedlichen 3D Modellen

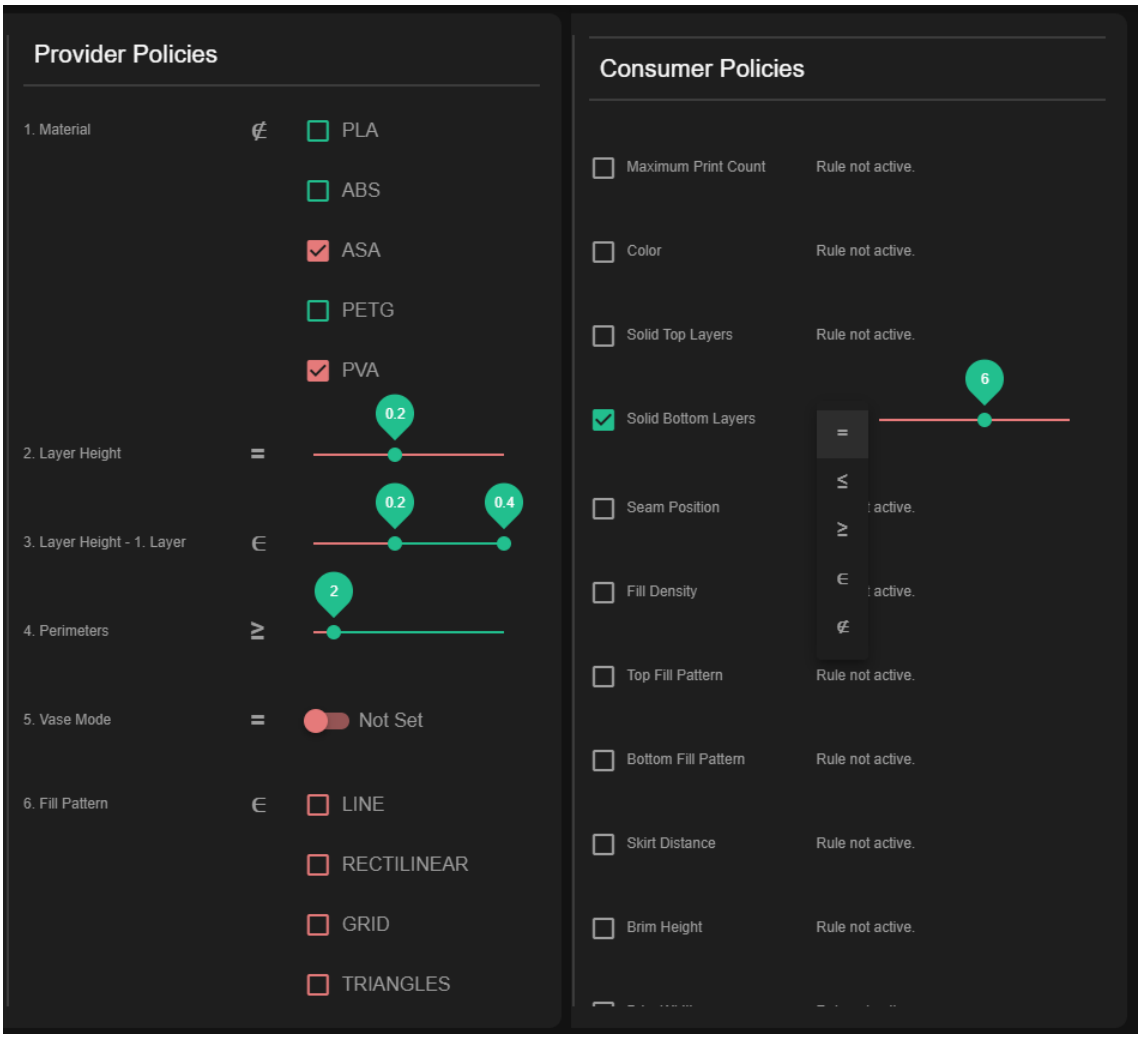

Abbildung 5.7: Screenshot, welcher auf der linken Seite die Anzeige zur lesenden Präsentation von Richtlinien des Modellerstellers zeigt. Auf der rechten Seite befindet sich die Anzeige zur Präsentation und Bearbeitung von Richtlinien des Nutzers. Die Anzeige ist im mittleren Bereich der Benutzeroberfläche sichtbar, wenn ein 3D Modell geöffnet ist.

zugeordnet werden. Außerdem wird für jeden Druckauftrag angezeigt, welcher Drucker den Auftrag ausführen soll und welche Priorität der Auftrag hat.

Ebenfalls wird darüber informiert, wie oft jeder Druckauftrag bereits gedruckt wurde. Die Anzahl wird dabei aus dem PIP des Betreibers bezogen. Falls in den Richtlinien des 3D Modells, welche zur Erstellung des Druckauftrags verwendet wurden, eine maximale Anzahl an Drucken definiert ist, wird diese ebenfalls angezeigt.

Darüber hinaus wird für jeden Druckauftrag angezeigt, welcher Status dem Druckauftrag beim Betreiber zugewiesen ist. Für Aufträge, welche in der Warteschlange eingereiht, aber noch nicht gestartet sind (– pending –) wird ebenfalls die Position in der Warteschlange angezeigt. Druckaufträge, welche entweder fertig gedruckt oder neu erstellt wurden (–

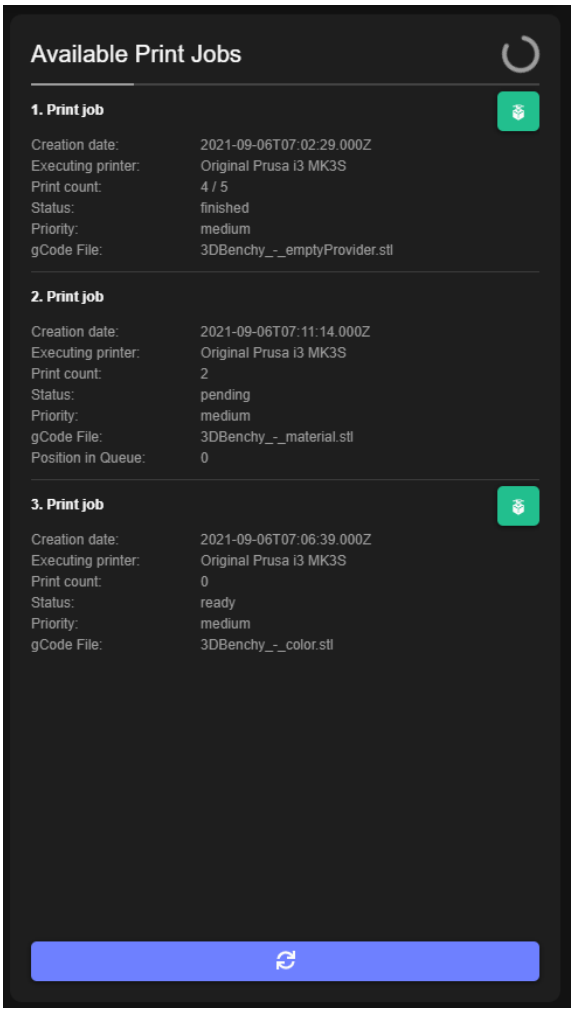

Abbildung 5.8: Screenshot, welcher die Anzeige für erstellte Druckaufträge zeigt. Die Anzeige ist im rechten Bereich der Benutzeroberfläche sichtbar.

finished und ready –), verfügen über eine Schaltfläche, welche es erlaubt den Auftrag beim Betreiber in die Warteschlange einzureihen.

Der Anwender hat die Möglichkeit die Liste der Druckaufträge über eine Schaltfläche im unteren Bereich der Anzeige erneut vom Betreiber zu laden. Außerdem aktualisiert sich die Anzeige periodisch selbstständig. Dabei wird dem Anwender angezeigt, ob grade ein Ladevorgang ausgeführt wird, und durch einen Fortschrittsbalken angedeutet, wie lange es bis zum nächsten Aktualisieren dauert. Der Grund für die automatische Aktualisierung ist, dass Druckaufträge auch ohne Interaktion des Anwenders ihren Status beim Betreiber ändern können, beispielsweise indem Druckaufträge gestartet oder abgeschlossen werden. Damit der Anwender zeitnah über diese Änderungen informiert wird, aktualisiert sich die Anzeige automatisch.

Abbildung 5.8 zeigt einen Screenshot, welcher die Anzeige für bekannte Druckaufträge enthält. Die dargestellte Liste enthält drei Druckaufträge, von denen zwei Stück durch einen Druck auf die grünen Schaltflächen in die Warteschlange eingereiht werden können. Der verbleibende Druckauftrag ist bereits eingereiht. Der erste Druckauftrag in der Liste hat in den anhängigen Richtlinien festgelegt, dass das 3D Modell, welches die Basis für den Auftrag bildet, nur fünfmal gedruckt werden darf. Es ist ebenso erkennbar, dass der Auftrag bereits viermal gedruckt wurde. In der oberen rechten Ecke ist ein (rotierendes) Symbol sichtbar, welches anzeigt, dass aktuell eine Aktualisierung durchgeführt wird. Unterhalb der Titelzeile befindet sich eine Fortschrittsanzeige, welche ein Indikator dafür ist, wann die nächste Aktualisierung ausgeführt wird.

5.3 Beschreibung des 3D Druck Services

Der 3D Druck Service, welcher vom Betreiber eingesetzt wird, ist mit D° implementiert worden. Dabei hat der 3D Druck Service verschiedene Aufgaben, welche umgesetzt werden müssen:

- Initialisierung des Systems (– dies umfasst unter anderem das initiale Anlegen der Digital Twins für 3D Drucker und das Anlegen von verfügbaren Materialien –).
- Bereitstellung von Daten über verfügbare 3D Drucker, Materialien, bekannter Druckeinstellungen und unterstützter Slicer.
- Entgegennahme von 3D Modellen mit verknüpften Richtlinien vom Kunden und Erstellung von Dateien, welche die 3D Drucker verwenden können (sog. Slicing).
- Bereitstellung von Daten über bekannte Druckaufträge.
- Starten von Druckaufträgen und das Einreihen selbiger in Warteschlangen.
- Verwaltung der Warteschlange von Druckjobs inkl. Zuordnung von Aufträgen an 3D Drucker, inklusive der Kommunikation mit den 3D Druck Servern, welche die eigentlichen 3D Drucker kontrollieren.

Zur Bereitstellung der Funktionen verwendet der 3D Druck Service für alle Funktionen, mit denen der Kunde interagiert, eine HTTP-Schnittstelle. Die Initialisierung des Systems und die Verwaltung der Warteschlange erfordern keine Interaktion mit dem Nutzer und bieten aus diesem Grund eine Kommandozeilen-Schnittstelle an. Da jede D°-Applikation eine Schnittstelle mit einem einzelnen Endpunkt, welcher Logik ausführt, verfügt, wurde der 3D Druck Service mit sechs einzelnen D°-Applikationen umgesetzt. Dabei wird jeder Punkt aus der vorherigen Liste durch eine eigene Applikation abgebildet.

Eine mögliche Alternative wäre die Entwicklung einer einzelnen Applikation, welche die unterschiedlichen Funktionalitäten über bedingte Verzweigungen in der Applikationslogik bereitstellt. Dies würde darin resultieren, dass der Endpunkt der Applikation eine komplizierte Schnittstelle aufweist, da alle Eingabeparameter, welche für die jeweiligen Funktionalitäten notwendig sind, von der Applikation entgegengenommen werden. Aus diesem Grund wurden die unterschiedlichen Funktionalitäten in jeweils eigenständigen Applikationen entwickelt.

Wie bereits in Abbildung 5.1 aufgezeigt, wird D° nur für den 3D Druck Service des Betreibers verwendet. Bei den anderen Komponenten wird auf existierende Software oder andere Programmiersprachen zur Implementation zurückgegriffen.

Abbildung 5.9 enthält eine detaillierte Darstellung, welche die einzelnen D°-Applikationen im 3D Druck Service zeigt. Des Weiteren enthält die Abbildung detaillierte Informationen darüber, welche Softwareprodukte beziehungsweise Technologien verwendet werden, um die anderen Komponenten des Demonstrators zu realisieren.

Die Abbildung zeigt außerdem eine Softwarekomponente, welche in Abbildung 5.1 nicht enthalten ist. Der Slicer wird verwendet, um aus 3D Modellen gCode zu erzeugen. Bei gCode handelt es sich um eine Liste von Befehlen, welche von einem 3D Drucker während des Drucks interpretiert und ausgeführt wird [Kra00]. Da der Slicer nur von einer einzelnen D°-Applikation verwendet wird und keine Interaktion mit anderen Komponenten hat, ist er in der vorherigen Abbildung nicht enthalten. Auf eine Darstellung des Kunden wurde in der Abbildung verzichtet, da jede beliebige Shopsoftware, welche den Verkauf digitaler Güter unterstützt, für den Demonstrator verwendet werden kann. Darüber hinaus liegt keine direkte Kopplung des Shops zu den anderen Komponenten des Demonstrators vor.

Ein weiterer Aspekt, welcher durch die Abbildung aufgezeigt wird ist, dass nur drei der sechs D°-Applikationen über integrierte Mechanismen zur Datennutzungskontrolle verfügen. Der Grund hierfür ist, dass die D°-Applikationen, welche nur Daten ausliefern (GetData & GetJobs), im beschriebenen Szenario keine Richtlinien umsetzen müssen. Die Init-Applikation, welche das System initial mit Daten über verfügbare Drucker, Filamente, etc. befüllt, muss ebenfalls keine Richtlinien umsetzen und verfügt dementsprechend über keine Mechanismen zur Datennutzungskontrolle.

Um sicherzustellen, dass D° über alle notwendigen Elemente verfügt, welche zur Umsetzung des Demonstrators notwendig sind, wurde ein entsprechendes Modul mit Sprachelementen entwickelt. Dieses Modul umfasst 99 Aktivitäten, 47 Typdefinitionen und 3 Richtlinien. Die Aktivitäten umfassen beispielsweise Funktionen zum Slicen eines 3D Modells, Hinzufügen eines neuen Filaments und das Updaten der 3D Drucker Warteschlangen. Beispiele für enthaltene Datentypen sind 3D Drucker, Druckmaterialien und Druckobjekte. Die definierten Richtlinien erfüllen die folgenden Aufgaben und setzen damit Anforderung 1 aus Abschnitt 5.1.5 um.

1. Validierung von Druckeinstellungen.
2. Einhaltung von Materialeinschränkungen.
3. Überprüfen, dass die maximale Anzahl an Drucken nicht überschritten wird.

Dabei wurden nicht alle 99 Aktivitäten in den sechs entwickelten D°-Applikationen verwendet. Das entwickelte Sprachmodul enthält Aktivitäten, welche CRUD-Operationen für diverse Typdefinitionen bereitstellen. Da diese im Demonstrator nicht verwendet werden, finden nur wenige der verfügbaren Aktivitäten Anwendung im Demonstrator.

In den sechs Applikationen werden insgesamt 27 verschiedene Aktivitäten verwendet. Unter den verwendeten Aktivitäten befinden sich solche zum Laden und Speichern von verschiedenen Datentypen in der verwendeten Datenbank und zur Erzeugung von Ausgaben auf der Konsole. Darüber hinaus werden Aktivitäten zur Interaktion mit Druckaufträgen

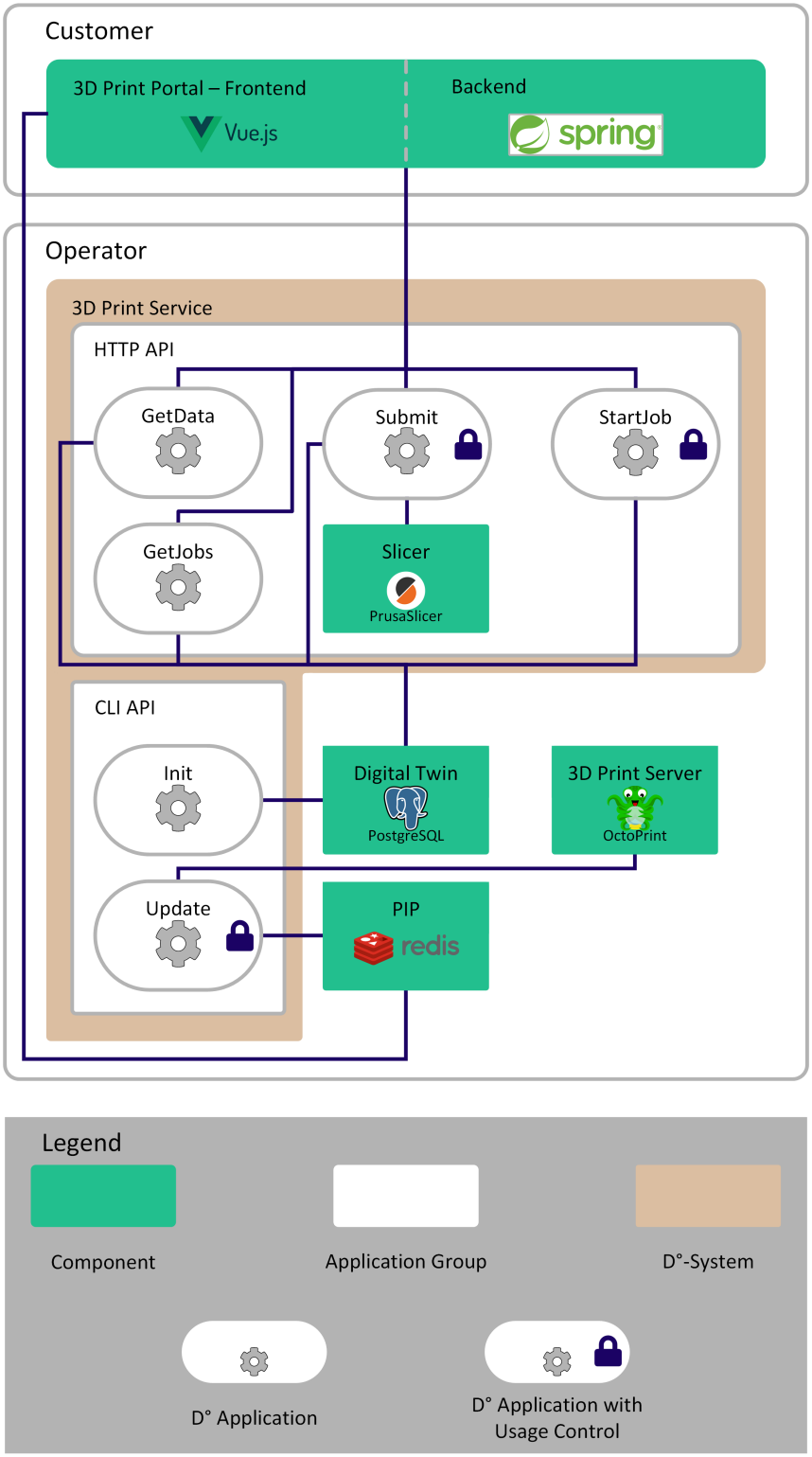

Abbildung 5.9: Detaillierte Darstellung der einzelnen Komponenten und verwendeten Technologien, welche in den Systemen des Kunden und des Betreibers verwendet werden.

verwendet. Dies umfasst das Erstellen, Einreihen in Warteschlangen und das Starten von Druckaufträgen. Außerdem wird eine Aktivität verwendet, welche die Druckjobs in der Warteschlange durchgeht und überprüft, ob der nächste Auftrag gestartet werden kann. Drei der verwendeten Aktivitäten sind mit den zuvor genannten Richtlinien verknüpft, welche zur Laufzeit überprüft werden, wodurch Anforderung 4 aus Abschnitt 5.1.5 umgesetzt wird.

Neben diesen Elementen, welche für den Entwickler von D°-Applikationen sichtbar und nutzbar sind, sind noch weitere Komponenten in dem Modul enthalten. Ein komplettes Domänenmodell, welches alle notwendigen Datentypen enthält, ist Teil des Moduls. Durch die Verwendung eines ORM-Mappers kann dieses Domänenmodell in der verwendeten PostgreSQL-Datenbank abgelegt werden. Dabei enthält die Datenbank unter anderem die verfügbaren 3D Drucker und deren digitale Zwillinge, geplante Druckjobs und verfügbare Druckmaterialien. Ebenfalls ist ein Redis-Client Teil des Moduls. Redis wird im Rahmen des Demonstrators als PIP verwendet. Der PIP hat die Aufgabe persistente Daten, welche zur Überprüfung von Richtlinien verwendet werden, abzuspeichern und auszuliefern.

Außerdem bietet das Modul Funktionen, welche eine Transformation zwischen dem Domänenmodell und den von D° verwendeten Datentypen erlaubt. Die Transformation in die andere Richtung ist ebenfalls eine Funktionalität des Moduls.

Bei den Richtlinien des Moduls ist ein zusätzlicher Aspekt zu beachten. Da die Richtlinien für jedes 3D Modell individuell sind, ist es nicht sinnvoll feste Richtlinien zu definieren und mit den verwendeten Aktivitäten zu verknüpfen. Solch feste Richtlinien würden es erfordern, dass für jedes 3D Modell ein neuer Satz an Applikationen gebaut wird, welche exakt die Richtlinien des 3D Modells enthalten. Dies ist eine direkte Folge aus der Art und Weise, wie D° Datennutzungskontrolle in Applikationen integriert [Bru20].

Aus diesem Grund wurde hier ein anderes Vorgehen gewählt, welches es erlaubt das konkrete Verhalten der Richtlinien zur Laufzeit zu beeinflussen. Zu diesem Zweck wurde ein besonderer Datentyp definiert und im Modul hinterlegt. Dieser Datentyp erlaubt es Regeln abzubilden, welche sich auf die Eigenschaften eines Druckauftrags beziehen. Beispielsweise kann festgelegt werden, dass ein bestimmtes Material verwendet oder eine gewisse Wandstärke nicht unterschritten wird. Ergänzend dazu wurden die Richtlinien so definiert, dass der zuvor genannte Datentyp als Eingabe verwendet wird und die enthaltenen Regeln zur Laufzeit ausgewertet werden. Hierdurch ist es ausreichend eine Instanz des 3D Druck Services zu betreiben, welche für alle 3D Modelle verwendet werden kann. Da die 3D Modelle und die verknüpften Richtlinien ohnehin gemeinsam mit den 3D Druck Service übertragen werden, ist hier keine Änderung notwendig, um dieses Vorgehen abzubilden.

Der Datentyp, welcher zur Definition von Richtlinien verwendet wird, enthält ein JSON-Objekt. Abbildung 5.10 zeigt ein Beispiel JSON-Objekt, welches eine komplette und valide Instanz des zuvor beschriebenen Datentypen enthält. Das Beispiel beinhaltet fünf Regeln, welche durch den Rechteinhaber definiert wurden, jedoch keine zusätzlichen Regeln des Kunden. Dabei beziehen sich alle fünf Regeln auf Attribute, welche sich in den Druckeinstellungen befinden. Das Objekt besteht aus drei zentralen Komponenten.

```
{
    "objectId" : "20c9d5d4-403e-4808-8011-bf0dc8c2df32",
    "provider" : {
        "layerHeight": {
            "op"    : "EQ",
            "value" : 0.2
        },
        "firstLayerHeight": {
            "op"    : "BTWN",
            "lower" : 0.2,
            "upper" : 0.4
        },
        "perimeters": {
            "op"    : "GE",
            "value" : 2
        },
        "bedTemperature": {
            "op"    : "LE",
            "value" : 65
        },
        "fillPattern": {
            "op"    : "IN",
            "value" : [
                { "value" : "HONEYCOMB" },
                { "value" : "GYROID" },
                { "value" : "_3DHONEYCOMB" }
            ]
        },
    },
    "consumer" : {

    }
}
```

Abbildung 5.10: Beispiel für Richtliniendefinitionen welche sich auf ein 3D Modell im Demonstrator bezieht. Das Beispiel enthält fünf verschiedene Regeln des Rechteinhabers und ein leeres Objekt, in welches die zusätzlichen Regeln des Kunden eingefügt werden können. Die `objectId` wird zur eindeutigen Identifikation des 3D Modells im Demonstrator verwendet.

- Ein eindeutiger Bezeichner für das 3D Modell, auf welches sich die Richtlinien beziehen. Dieser findet sich im Beispiel in Zeile 2 und erlaubt die eindeutige Identifikation des 3D Modells im gesamten Demonstrator.
- Ein Objekt, welches die Regeln beinhaltet, die vom Modellersteller definiert wurden. In der Abbildung findet sich dieses Objekt in den Zeilen 3 bis 29.
- Ein Objekt, welches die Regeln beinhaltet, die der Kunde dem 3D Modell hinzugefügt hat. Im Beispiel ist dieses Objekt leer und kann in den Zeilen 30 bis 32 gefunden werden.

Die Regeln, welche in den beiden Objekten für den Modellersteller und den Kunden enthalten sind, besitzen die folgende Struktur.

- Unter dem Namen des Attributs, auf welchen sich die Regel bezieht, werden weitere Attribute abgelegt.
- Das Attribut `op` enthält eine Zeichenkette und bestimmt, um welche Art von Regel es sich handelt. Dabei ist die Menge von erlaubten Zeichenketten abhängig vom Typen des Attributs, für das die Regel definiert wird. Für numerische Attribute können Regeln definiert werden, welche erfordern, dass der Wert gleich, kleiner-gleich oder größer-gleich einem definierten Wert ist. Darüber hinaus kann definiert werden, dass der Wert in einem geschlossenem Intervall enthalten ist bzw. nicht in einem offenem Intervall vorkommt. Für aufzählende Attribute kann definiert werden, dass sie in einer definierten Menge enthalten sein müssen bzw. nicht darin vorkommen dürfen. Für boolesche Attribute kann festgelegt werden, welchen Wert sie einnehmen müssen.
- Die Regeln enthalten außerdem die Parameter, welche erlaubte bzw. verbotene Werte spezifizieren. Für Regeln, welche ein Intervall definieren, werden hierzu die Attribute `lower` und `upper` verwendet. Für alle anderen Regeln wird das Attribut `value` verwendet. Dabei kann dieses Attribut abhängig von der Regel entweder einen numerischen Wert oder eine Liste von Objekten, welche jeweils aus einer Zeichenkette bestehen, enthalten.

Wie zuvor beschrieben, muss der 3D Druck Service verschiedene Funktionalitäten bieten. Dies umfasst sowohl komplexere, als auch einfache Funktionalitäten. Ein Beispiel für eine komplexere Funktionalität ist die Entgegennahme eines 3D Modells, welches mit bestimmten Druckeinstellungen gesliced und anschließend in der Datenbank hinterlegt wird. Einfache Funktionalitäten sind beispielsweise das Hinzufügen und Entfernen von Druckmaterialien, sowie das Abfragen aller bekannten Druckaufträge. Dies zeigt sich auch in der Komplexität der unterschiedlichen Aktivitäten, welche diese Funktionalitäten in Form einer Java-Implementation bereitstellen.

5.4 Umsetzung von Datennutzungskontrolle

In diesem Abschnitt wird konkret aufgezeigt, wie die Datennutzungskontrolle innerhalb des Demonstrators umgesetzt wurde. Dabei wird zum einen betrachtet, wie die Richtlinien, welche fester Bestandteil der einzelnen Applikationen des 3D Druck Services sind, umgesetzt

werden. Zum anderen wird aufgezeigt, wie die Regeln, welche mit den individuellen 3D Modellen verknüpft sind, von den jeweiligen Applikationen eingehalten werden.

5.4.1 Integration in Applikationen

Für die Umsetzung der Datennutzungskontrolle sind im vorgestellten Szenario zwei Komponenten notwendig. Zum einen werden D°-Richtlinien mit Aktivitäten verknüpft und somit in die Applikationen integriert. Diese Richtlinien sollen zur Laufzeit für unterschiedliche 3D Modelle, welche über verschiedene verknüpfte Regelsätze verfügen, verwendet werden können. Zu diesem Zweck erwarten die Richtlinien den besagten Regelsatz, welcher ein Datentyp im entwickelten Modul ist, als Eingabe und werten die enthaltenen Regeln zur Laufzeit aus. Dieser spezielle Datentyp, welcher verwendet wird, um die individuellen Regeln zu definieren, ist die zweite wichtige Komponente für die Umsetzung der Datennutzungskontrolle im Demonstrator.

Die Integration der Richtlinien in die Applikation geschieht im Rahmen der Codegenerierung, welche vom D°-Compiler durchgeführt wird. Im D°-Programmcode finden sich keine Hinweise auf eine Verwendung der Richtlinie. Dieser verwendet Aktivitätsinstanzen, um die Applikationslogik abzubilden. Diese Aktivitätsinstanzen haben verweise auf Richtlinien, welche im Rahmen der Aktivitätsausführung umgesetzt werden müssen.

Der D°-Compiler löst diese Verweise auf und erzeugt entsprechenden Programmcode in der Hostsprache. Dabei wird im generierten Programmcode jeder Aktivitätsaufruf von der Sandbox ausgeführt, welche Bestandteil jeder D°-Applikation ist. Bevor die Sandbox die eigentliche Logik der aufgerufenen Aktivität ausführt, wird ein Endpunkt aller Richtlinien, welche mit der Aktivität verknüpft sind, aufgerufen. Sofern hier keine Verletzungen der Richtlinien festgestellt werden, wird die eigentliche Aktivität ausgeführt. Im Anschluss wird ein weiterer Endpunkt der verknüpften Richtlinien aufgerufen. Somit ist durch die Sandbox sichergestellt, dass Vor- und Nachbedingungen von Richtlinien korrekt ausgeführt werden. Hiermit sind zwei der drei Endpunkte von D°-Richtlinien abgedeckt.

Der Aufruf des dritten Endpunkts der Richtlinien wird nicht durch die Sandbox sichergestellt. Dieser wird stattdessen vom Security Manager aufgerufen, wenn geschützte Schnittstellen (bspw. Dateioperationen) der Hostsprache verwendet werden. Wird eine solche geschützte Schnittstelle verwendet, wird vor der eigentlichen Ausführung der Operation der Security Manager aufgerufen. Der Security Manager fragt bei der Sandbox die Aktivität an, welche sich aktuell in Ausführung befindet. Hierdurch können die verknüpften Richtlinien identifiziert werden. Von diesen identifizierten Richtlinien wird daraufhin der verbleibende Endpunkt der Richtlinien-Schnittstelle aufgerufen.

Abbildung 5.11 enthält eine schematische Darstellung der zuvor beschriebenen Codegenerierung, mit einem Fokus auf die Usage Control Mechanismen. Die Abbildung zeigt deutlich, wie durch die Sandbox zwei der drei Richtlinien-Endpunkte aufgerufen werden. Ebenso ist der Security Manager sichtbar, welcher für die Aufrufe des dritten Endpunktes verantwortlich ist. Durch dieses Vorgehen ist sichergestellt, dass die Richtlinien, welche in D°-Applikationen eingesetzt werden, zu den notwendigen Zeitpunkten ausgeführt werden.

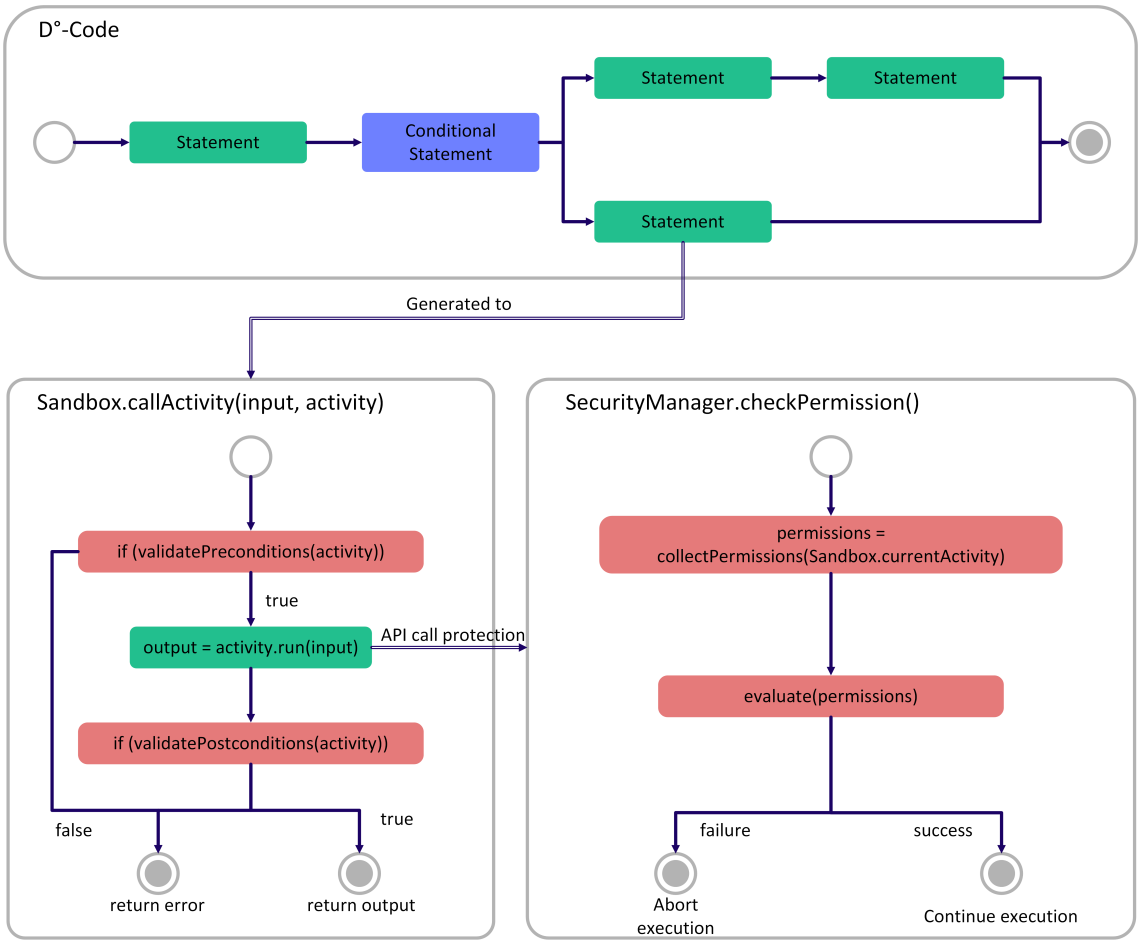

Abbildung 5.11: Schematische Darstellung, welche aufzeigt, welcher Code vom D°-Compiler für Aktivitätsaufrufe erzeugt wird und wie dabei Usage Control Mechanismen integriert werden. Quelle: [Bru20]

5.4.2 Umsetzung der Regeln

Die Codegenerierung stellt die korrekte Umsetzung der Richtlinie innerhalb der Applikation sicher. Für eine vollständige Umsetzung der Datennutzungskontrolle muss ebenfalls sichergestellt werden, dass die Regeln, welche mit den 3D Modellen verknüpft sind, korrekt an die Richtlinie übergeben werden.

Dies wird bereits während der Erzeugung der Aktivitätsinstanzen sichergestellt. Die Aktivitäten, welche die Richtlinie zur Regelumsetzung umsetzen müssen, werden mit einem zusätzlichen Eingabeparameter versehen. Dieser Eingabeparameter enthält die Regeln, welche für die verarbeiteten 3D Modelle gelten. Bei der Erzeugung der Aktivitätsinstanz besteht die Möglichkeit eine Abbildung von Eingabeparametern der Aktivität auf Eingabeparameter der Richtlinie zu definieren.

Diese Abbildung der Eingabeparameter wird während der Laufzeit durch die Sandbox aufgelöst. Da die Sandbox eine Mehrheit der Endpunkte von Richtlinien aufruft, verfügt

sie auch über eine Funktionalität, welche es erlaubt die notwendigen Eingabeparameter für die Richtlinienausführung aus unterschiedlichen Quellen zu sammeln. Somit wird durch die Codegenerierung in Kombination mit den Usage Control Konzepten, welche in die finale Applikation integriert werden, und die Verknüpfung von Richtlinien mit Sprachelementen sichergestellt, dass die Datennutzungskontrolle wie geplant funktioniert.

5.4.3 Vollständiges Beispiel: D°-Applikation zum erstellen von Druckaufträgen

Für einen besseren Überblick, wie Datennutzungskontrolle in D° im Allgemeinen und im Demonstrator im Speziellen, umgesetzt wird, wird nachfolgend eine der D°-Applikationen, welche eine der zuvor beschriebenen Richtlinien umsetzt, genauer betrachtet. Dabei werden zunächst die eigentliche Applikation und die verwendeten Sprachelemente betrachtet. Daraufhin wird die Richtlinie und deren Verknüpfung mit der Applikationslogik aufgezeigt. Abschließend wird dargestellt, wie der D°-Compiler die einzelnen Elemente miteinander kombiniert, um eine ausführbare Applikation zu erzeugen.

Die Beispielapplikation Als Beispiel wird die Applikation ausgewählt, welche vom Kunden 3D Modelle und eine Auswahl von Druckparametern entgegennimmt und aus diesen Daten einen Druckauftrag erzeugt. Dieser Prozess inkludiert das sog. Slicen des 3D Modells, bei dem eine Menge von Instruktionen erzeugt wird, welche während einer späteren Ausführung des Druckjobs von dem ausgewähltem 3D Drucker ausgeführt werden.

Wie bereits in Abschnitt 4.1.1 beschrieben, besteht jede D°-Applikation aus zwei Komponenten: einer Konfiguration für die Applikation und der eigentlichen Applikationslogik. Der Konfigurationsbereich wird während der Übersetzung der Applikation vom D°-Compiler eingelesen. Die enthaltenen Metadaten werden anschließend an dafür vorgesehenen Stellen in die erzeugte Applikation eingefügt und können innerhalb der Applikation verwendet werden (bspw. im Kontext der Richtlinienauswertung). Die Daten im Konfigurationsbereich, welche das Verhalten der Applikation bestimmen, haben einen direkten Einfluss auf den Übersetzungsvorgang. Dabei wird nicht die Applikationslogik verändert, sondern nur das Aufrufverhalten der Applikation bestimmt [Bru21b].

```
1 configuration
2     - namespace : "de_fhg_isst_oe270.degree.demonstrator.print3d"
3     - name : "submit"
4     - version : "0.0.1-1-SNAPSHOT"
5     - tags: "SHOWCASE"
6     - execution: "single"
7     - port : "1101"
8     - url: "process"
```

Abbildung 5.12: Konfiguration einer D°-Applikation, welche vom Betreiber als Teil des 3D Druck Services verwendet wird. Die Applikation nimmt Kundendaten entgegen und erzeugt Druckaufträge, welche nachfolgend ausgeführt werden können.

Abbildung 5.12 zeigt den Konfigurationsbereich der Applikation zur Erstellung von Druckaufträgen. Die erste Zeile leitet den Konfigurationsbereich ein und ist ein von der D°-Grammatik verlangter Bestandteil der Konfiguration. Die Zeilen 2 bis 5 beinhalten dabei Metadaten der Applikation. In den Zeilen 6 bis 8 findet sich die Einstellung des Verhaltens der Applikation. Dabei wird in dem Beispiel eine Applikation gebaut, welche über eine HTTP-Schnittstelle unter dem Pfad `/process` unter Port 1101 erreichbar ist. Ebenfalls wird festgelegt, dass die Applikationslogik bei einem Aufruf nur einmalig ausgeführt wird.

Die vorgestellte Applikation erhält verschiedene Eingabeparameter. Neben dem eigentlichen 3D Modell und anhängiger Richtlinien werden die eindeutigen Bezeichner relevanter Elemente an die Applikation übergeben. Dabei sind Bezeichner für den Satz an Druckeinstellungen, welche verwendet werden sollen, den Slicer, mit welchem das 3D Modell verarbeitet werden soll, das verwendete Druckermodell und die konkrete Druckerinstanz in den Eingabeparametern enthalten. Unter Verwendung dieser Eingabeparameter führt die Applikationslogik die folgenden Schritte aus:

- Laden der angegebenen Druckeinstellungen aus der Datenbank.
- Laden der Repräsentation des 3D Drucker Modells, welches zum Druck des Modells verwendet werden soll, aus der Datenbank.
- Verknüpfung der geladenen Druckeinstellungen und des 3D Druckers mit dem an die Applikation übergebenem 3D Modell.
- Erzeugen und Speichern von sog. gCode mit dem spezifiziertem Slicer und Druckeinstellungen, welcher vom ausführenden 3D Drucker ausgeführt werden kann.
- Laden des frisch erstellten gCodes, der Warteschlange und der ausführenden 3D Drucker Instanz aus der Datenbank.
- Erstellen eines neuen Druckauftrags mit den zuvor erstellten bzw. geladenen Attributen und den übergebenen Richtlinien.
- Speichern des soeben erstellten Druckauftrags in der Datenbank.
- Rückgabe einer Erfolgsmeldung und des eindeutigen Bezeichners des erstellten Druckauftrags.

Abbildung 5.13 zeigt den D°-Code, welcher die zuvor beschriebene Applikationslogik der Applikation zeigt. Dabei wurden einige Stellen aus Gründen der Übersichtlichkeit gekürzt. In den Zeilen 15, 21 und 32 wurden Fehlerprüfungen und -behandlungen gekürzt, da diese, bis auf andere Statusnachrichten, identisch zu dem Block in den Zeilen 7 bis 12 sind. Ebenfalls wurde in Zeile 23 das Laden von drei Elementen aus der Datenbank entfernt, da dies analog zu dem Laden der Druckeinstellungen mit anschließender Fehlerbehandlung in den Zeilen 6 bis 12 ist.

Zeile 1 des Beispiels enthält das Schlüsselwort `code`, welches von der D°-Grammatik gefordert wird, um die Applikationslogik einzuleiten. In Zeile 2 befindet sich eine Definition aller Eingabeparameter. Diese umfasst sowohl Namen der Parameter, als auch deren Typen. In den Zeilen 3 bis 4 und 34 bis 35 erzeugen Statusnachrichten, welche auf der Konsole

```
code
    [printModel = $demonstrator_3d_printer.PrintModel(), printSettingsId =
↪  $Integer, slicerId = $Integer(), printerTypeId = $Integer(), printerId
↪  = $Integer(), policies = $Text()] -> begin
        statusMsg = $Text(@write["Registering new 3D model."]);
        LogToConsole[statusMsg];

        [success, loadedPrintSettings] = demonstrator_3d_printer.
↪ getPrintSettingsById[printSettingsId];
        if (success == "false") begin
            statusMsg = "An error occurred while loading print settings.";
            LogToConsole[statusMsg];
            error = $Error(@write["An error occurred while loading print
↪ settings."]);
            return [error];
        end

        printModel.recommendedPrintSettings = loadedPrintSettings;
        [success, printModelId] = demonstrator_3d_printer.addPrintModel[
↪ printModel];
        if (success == "false") begin /* [...] */ end

        [success, gCodeId, msg] = demonstrator_3d_printer.slice[
↪ printModelId, printSettingsId, slicerId, printerTypeId, policies];
        if (success == "false") begin /* [...] */ end

        /* Get created gCode, the queue, and the executing printer. */

        printJob = $demonstrator_3d_printer.PrintJob();
        printJob.gCodes += gCode;
        printJob.queue = queue;
        printJob.executingPrinter = executingPrinter;
        printJob.policies = policies;

        [success, printJobId] = demonstrator_3d_printer.addPrintJob[
↪ printJob];
        if (success == "false") begin /* [...] */ end

        statusMsg = "Finished registration of 3D model.";
        LogToConsole[statusMsg];
        return[success, printJobId];
    end
```

Abbildung 5.13: Applikationscode einer D°-Applikation, welche vom Betreiber als Teil des 3D Druck Services verwendet wird. Die Applikation nimmt Kundendaten entgegen und erzeugt Druckaufträge, welche nachfolgend ausgeführt werden können.

ausgegeben werden und dem Betreiber somit Informationen über den aktuellen Zustand der Applikation geben.

Die Zeilen 6 bis 12 sind dafür verantwortlich die in den Eingabeparametern spezifizierten Druckeinstellungen aus der Datenbank zu laden und auf Fehler beim Laden zu prüfen. Dabei findet sich in Zeile 6 der Aufruf der Aktivität `demonstrator_3d_printer.getPrintSettingsById`. `demonstrator_3d_printer` ist das Präfix, welches für alle Sprachelemente (Aktivitäten, Datentypen und Richtlinien) verwendet wird, die Teil des Sprachmoduls sind, das für den beschriebenen Demonstrator entwickelt wurde. Die Aktivität nimmt einen Eingabeparameter entgegen, welcher den eindeutigen Bezeichner der zu ladenden Druckeinstellungen enthält, und erzeugt zwei Rückgabewerte: einen Indikator, ob die Aktivität erfolgreich ausgeführt wurde, und die geladenen Druckeinstellungen. Hier zeigt sich direkt eine Besonderheit von D°: Sowohl Aktivitäten, als auch D°-Applikationen können beliebig viele Rückgabewerte erzeugen. Die Zeilen 7 bis 12 beinhalten die Fehlerbehandlung für die Aktivität. Falls der von der Aktivität zurückgegebene Indikator einen Fehler anzeigt, wird eine Statusnachricht auf der Konsole ausgegeben und auch an den Aufrufer der Applikation zurückgegeben.

In Zeile 14 werden die geladenen Druckeinstellungen mit dem an die Applikation übergebenem 3D Modell verknüpft. In Zeile 15 wird das 3D Modell in der Datenbank des Betreibers hinterlegt. Die anschließende Fehlerprüfung und -behandlung findet sich in Zeile 16, wurde in der Abbildung aus Gründen der Übersichtlichkeit jedoch gekürzt. Die Aktivität, welche das 3D Modell in der Datenbank speichert, gibt den eindeutigen Bezeichner des erzeugten Elements zurück.

In Zeile 18 wird das Slicing des 3D Modells ausgelöst. Bei der Ausführung dieser Aktivität wird der spezifizierte Slicer aufgerufen, welcher aus dem 3D Modell, den Druckeinstellungen und dem ausführenden Drucker gCode erzeugt. Dieser erzeugte gCode wird in der Datenbank abgelegt und der eindeutige Bezeichner als Rückgabewert zurückgegeben.

Anschließend werden diverse Elemente aus der Datenbank geladen. Da der Code hierfür analog zu den Zeilen 6 bis 12 ist, wurde die Stelle in der Abbildung gekürzt, um Redundanzen zu vermeiden. Die angesprochene Kürzung ist in Zeile 21 angedeutet.

Anschließend wird in den Zeilen 23 bis 27 ein Druckauftrag erzeugt. Diesem Druckauftrag werden die geladenen bzw. an die Applikation übergebenen Daten zugewiesen. Danach wird der fertige Druckauftrag in der Datenbank gespeichert. Der entsprechende Aktivitätsaufruf findet sich in Zeile 29. Abschließend gibt die Applikation in Zeile 34 einen Indikator, ob die Ausführung der Applikation fehlerfrei durchlaufen ist, und den Bezeichner des angelegten Druckauftrags zurück.

Neben der Ausführung der Applikationslogik hat die Applikation eine weitere Aufgabe: Die Überprüfung und Durchsetzung einer der drei Richtlinien, welche im Szenario verwendet werden. Die Applikation überführt 3D Modelle in gCode und verwendet hierfür Druckeinstellungen, welche maßgeblich Einfluss auf den erzeugten gCode haben. Somit ist es am sinnvollsten zu überprüfen, ob die Regeln, welche Druckeinstellungen einschränken, eingehalten werden, bevor diese Druckeinstellungen im Rahmen des Slicing verwendet werden, um gCode zu erzeugen. So kann sichergestellt werden, dass kein gCode erzeugt wird, welcher auf Druckeinstellungen basiert, die den definierten Regeln widersprechen.

Definition & Implementation von Aktivitäten Betrachtet man den Aufruf der Aktivität in Abbildung 5.13 Zeile 18, welche das Slicing durchführt, findet sich kein Hinweis auf eine Überprüfung einer entsprechenden Richtlinie. Auch in den vorherigen Zeilen findet sich keine entsprechende Anweisung. Dies ist das Ergebnis der Umsetzung der policy-agnostischen Programmierung in D°. Der Programmcode der Applikation fokussiert sich einzig auf die Abbildung der Applikationslogik. Die Integration von Richtlinien in die Applikation findet an anderer Stelle statt. Somit wird die Applikationslogik nicht durch die Mechanismen zur Datennutzungskontrolle aufgebläht, was die Lesbarkeit verbessert. Darüber hinaus benötigen die Entwickler von Applikationen kein Wissen über die korrekte Nutzung der Richtlinien.

Um nachzuvollziehen, an welchen Stellen Richtlinien in der Applikation integriert sind, muss genauer betrachtet werden, wie die einzelnen Sprachelemente definiert sind und verwendet werden. Der grundlegende Baustein dafür ist die Definition einer Aktivität, die nicht nur dessen Namen festlegt. Auch beschreibt sie die Ein- und Ausgabeparameter, welche die Signatur definieren, mit der die Aktivität in Applikationen verwendet werden kann. Dabei enthält die Definition der Aktivität keine Logik, welche die gewünschte Funktionalität umsetzt. Stattdessen enthält die Definition einen Hinweis darauf, wie die Logik der Aktivität implementiert wurde.

Abbildung 5.14 zeigt die Definition der Aktivität, die zum Slicen der 3D Modelle verwendet wird. Alle Definitionen von Eingabeparametern, außer der ersten, wurden gekürzt, um die Lesbarkeit zu verbessern und Redundanzen zu vermeiden. Die Zeilen 3 bis 4 definieren den Namen der Aktivität, welcher verwendet wird, wenn andere Sprachelemente diese Definition referenzieren.

Dabei enthalten die Zeilen 15 bis 28 die Beschreibungen der Rückgabewerte der Aktivität. Jeder Rückgabewert wird durch seinen Namen und des verwendeten Datentyps definiert. Die gezeigte Aktivität verfügt über drei Rückgabewerte:

1. Einen booleschen Wert, der anzeigt, ob die Ausführung erfolgreich war.
2. Den eindeutigen Bezeichner des gCode Objekts, welches von der Aktivität erzeugt und abgespeichert wurde.
3. Einen Text, welcher die vom Slicer erzeugten Statusnachrichten enthält.

In den Zeilen 5 bis 14 findet sich die Definition der Eingabeparameter der Aktivität. Eingabeparameter werden ebenfalls über einen Namen und den verwendeten Datentypen definiert. Darüber hinaus können Eingabeparameter mit Tags versehen werden. Mit diesen Tags kann auf eine maschinenlesbare Weise beschrieben werden, wie die einzelnen Parameter verwendet werden. Diese Tags beeinflussen nicht das Verhalten der Aktivität, können aber während der Auswertung von Richtlinien verwendet werden. Aktivitätsdefinitionen können aus dem selben Grund mit Tags versehen werden; das gezeigte Beispiel macht von dieser Funktionalität keinen Gebrauch.

In den Zeilen 29 und 30 wird angegeben, wie die Logik der Aktivität implementiert ist. In diesem Fall handelt es sich um eine Java-Implementierung. Die Information wird später vom Compiler und auch von der erzeugten Applikation verwendet, um eine entsprechend

```yaml
demonstrator_3d_printer.sliceDef:
  degree.Activity@SliceDef:
    name:
      Identifier: "demonstrator_3d_printer.sliceDef"
    inputParameters:
      degree.TaggedParameter:
        - name:
            Identifier: "printModelId"
          type:
            Type: "core.Integer"
          tags:
            degree.ActivityInputTag:
              - "DISTRIBUTE"
        # [...]
    outputParameters:
      degree.Parameter:
        - name:
            Identifier: "success"
          type:
            Type: "Boolean"
        - name:
            Identifier: "gCodeId"
          type:
            Type: "Integer"
        - name:
            Identifier: "msg"
          type:
            Type: "Text"
    executionContainer:
      degree.ExecutionContainer: "java"
```

Abbildung 5.14: Definition der Aktivität, welche zum Slicen von 3D Modellen verwendet wird. Die Definition liegt im `yaml`-Format vor und definiert den Namen der Aktivität, Ein- und Ausgabeparameter und den Ausführungskontext.

```
1  @ActivityAnnotation(qualifiedName = "demonstrator_3d_printer.sliceDef")
2  public class Slice extends BaseActivity {
3
4      // [...]
5
6      @NotNull
7      @Override
8      public OutputScope run(@NotNull InputScope inputScope) {
9          // [...]
10     }
11
12 }
```

Abbildung 5.15: Struktur der Implementation, welche die Logik der Aktivität zum Slicen von 3D Modellen enthält. Die eigentliche Logik wurde entfernt.

annotierte Java-Klasse zu finden. Diese Klasse wird aufgerufen, wenn die Aktivität innerhalb der Applikation verwendet wird.

Abbildung 5.15 zeigt die Struktur der Implementation für die betrachtete Aktivität. Dabei wurde der eigentliche Programmcode, welcher die Logik enthält, entfernt, da dieser in diesem Beispiel nicht von Relevanz ist. In Zeile 1 findet sich die Annotation, welche dafür verantwortlich ist, dass der Compiler und die erzeugte Applikation die Klasse auffinden können. Dabei verfügt die Annotation über einen Parameter, welcher angibt, für welche Aktivitätsdefinition die Klasse eine Implementation liefert.

Um sicherzustellen, dass die D°-Applikation die Implementation der Aktivität aufrufen kann, gibt es eine vorgeschriebene Schnittstelle, welche implementiert werden muss. Dies spiegelt sich in dem gezeigten Beispiel in Zeile 2 wider, in der angegeben ist, dass die Klasse `BaseActivity` durch die definierte Klasse erweitert wird. Die eigentliche Implementation der Schnittstelle findet sich in den Zeilen 6 bis 10.

Definition von Richtlinien Nachdem aufgezeigt wurde, wie Aktivitäten definiert und implementiert werden, wird betrachtet, wie Richtlinien definiert werden. Dabei wird für Richtlinien auf eine genauere Betrachtung der Implementation verzichtet, da das Vorgehen analog zu dem bei Aktivitäten ist: Die Implementation einer Richtlinie wird über eine spezielle Annotation gefunden und mit der Definition verknüpft. Darüber hinaus muss die Implementierung eine vorgegebene Schnittstelle umsetzen, um von der Applikation später korrekt verwendet werden zu können.

Die Definition einer Richtlinie zeigt zu einem gewissen Grad Überschneidungen zur Definition einer Aktivität. Dies zeigt sich auch in Abbildung 5.16, welche die Definition der Richtlinie enthält, die zur Überprüfung von Druckeinstellungen verwendet wird. Dabei liegt die Definition im `yaml`-Format vor. In den Zeilen 3 und 4 wird der Name der Richtlinie festgelegt. Die Beschreibung der Eingabeparameter findet sich in den Zeilen 6 bis 14. Analog zu den Eingabeparametern von Aktivitäten, verfügt jeder Parameter über einen Namen und einen Typen. Anders als bei Aktivitäten, können weder die Parameter einer Richtlinie,

```yaml
demonstrator_3d_printer.ValidatePrintSetting:
  degree.Constraint@ValidatePrintSetting:
    name:
      Identifier: "demonstrator_3d_printer.validatePrintSetting"
    attribute:
      degree.Parameter:
        - name:
            Identifier: "policy"
          type:
            Type: "Text"
        - name:
            Identifier: "printSettingsId"
          type:
            Type: "Integer"
```

Abbildung 5.16: Definition der Richtlinie, welche zum Überprüfen und Einschränken von Druckeinstellungen verwendet wird. Die Definition liegt im `yaml`-Format vor und definiert Eingabeparameter der Richtlinie.

noch die Richtlinie selbst mit Tags versehen werden. Ebenso entfällt für Richtlinien die Notwendigkeit zu spezifizieren, auf welche Weise die Implementation bereitgestellt wird. Der Grund hierfür ist, dass alle Richtlinien in der verwendeten Hostsprache – im vorliegenden Fall handelt es sich dabei um Java – implementiert werden.

Instanziierung von Richtlinien & Aktivitäten Nachdem sowohl die Aktivität, als auch die Richtlinie definiert wurden, ist es notwendig diese beiden Elemente miteinander zu verknüpfen, um sie in Applikationen verwenden zu können. Dies geschieht in D° über Instanzen, welche aus den Definitionen der Elemente erzeugt werden. Die zuvor betrachteten Definitionen können nicht direkt in Applikationen verwendet werden.

```yaml
  demonstrator_3d_printer.ValidatePrintSettingsPreSlice:
    degree.ConstraintInstance@ValidatePrintSettingPreSlice:
      name:
        Identifier: "demonstrator_3d_printer.validatePrintSettingPreSlice"
      definition:
        degree.ConstraintReference: "degree.
↪ Constraint@ValidatePrintSetting"

```

Abbildung 5.17: Definition der Richtlinieninstanz, welche zum Überprüfen und Einschränken von Druckeinstellungen verwendet wird. Die Definition liegt im `yaml`-Format vor und definiert die zugrundeliegende Richtliniendefinition.

Abbildung 5.17 zeigt die Instanziierung der zuvor beschriebenen Richtlinie im `yaml`-Format. Im vorliegenden Beispiel wird bei der Instanziierung nur ein Name vergeben und die zugrundeliegende Definition referenziert. Grundsätzlich ist es möglich bei der Erzeugung von Richtlinieninstanzen die Eingabeparameter der Richtlinie mit konstanten Werten zu belegen. Dies ist nützlich, wenn einzelne Parameter sich zur Laufzeit nicht ändern können, wird im vorliegenden Beispiel aber nicht verwendet. Im gezeigten Beispiel werden keine neuen bzw. zusätzlichen Daten in die Richtlinieninstanz eingefügt. Dennoch ist dieser Schritt notwendig, um die Richtlinie mit Aktivitäten verknüpfen und somit in Applikationen verwenden zu können.

Bei Aktivitäten ist der Schritt der Erzeugung einer Instanz für den Anwender unter Umständen optional. Bei der Erzeugung von Aktivitätsinstanzen stehen verschiedene Funktionalitäten zur Verfügung. Beispielsweise die Verknüpfung der Instanz mit Instanzen von Richtlinien. Wird für den konkreten Anwendungsfall keine dieser Funktionalitäten benötigt, kann auf die Erzeugung einer Aktivitätsinstanz verzichtet werden. Der Entwickler verwendet die Definition dann direkt im D°-Code. Der Compiler erkennt dies im Rahmen des Übersetzungsvorgangs und erzeugt eine sogenannte Schatteninstanz. Diese Schatteninstanz wird dann anstelle der Definition aufgerufen und in die erzeugte Applikation integriert. Da im gezeigten Beispiel die Aktivität mit einer Richtlinie verknüpft werden soll, kann auf die Erzeugung einer Aktivitätsinstanz nicht verzichtet werden.

Abbildung 5.18 zeigt die Beschreibung der Aktivitätsinstanz, welche für die zuvor beschriebene Aktivität erzeugt wird, im `yaml`-Format. In den Zeilen 3 und 4 wird der Instanz ein Name zugewiesen. Die Referenz der zugrundeliegenden Aktivitätsdefinition findet sich in den Zeilen 5 und 6. Die Verknüpfung mit der zuvor gezeigten Richtlinieninstanz findet sich in den Zeilen 7 bis 12. Dabei kann eine Aktivität mit beliebig vielen Richtlinien verknüpft werden. Diese Verknüpfungen werden als Liste angegeben. Im gezeigten Beispiel findet nur eine Verknüpfung mit einer einzelnen Richtlinie statt. Der Eintrag für diese einzelne Verknüpfung findet sich in den Zeilen 9 bis 12. Dabei wird in den Zeile 11 und 12 angegeben, welche Richtlinieninstanz mit der Aktivitätsinstanz verknüpft werden soll. In den Zeilen 9 und 10 wird der verknüpften Richtlinieninstanz ein Name zugewiesen, welcher eine eindeutige Identifikation der Richtlinieninstanz innerhalb der Aktivitätsinstanz erlaubt.

Da die verwendete Richtlinie für den Entwickler nicht im Applikationscode sichtbar ist, hat dieser auch keine Möglichkeit sicherzustellen, dass die Eingabeparameter der Richtlinie mit den notwendigen Werten belegt wird. Um dies sicherzustellen, ist es notwendig bei der Instanziierung der Aktivität eine Abbildung anzugeben, welche die Eingabeparameter der Aktivität mit den noch nicht belegten Eingabeparametern der Richtlinie verbindet. Dies findet sich im gezeigten Beispiel in den Zeilen 13 bis 24. Diese Abbildung besteht aus einer Liste von Einträgen, welche jeweils einen Eingabeparameter der Aktivität auf eine Menge von Eingabeparametern der verknüpften Richtlinien abbildet. Die gezeigte Abbildung enthält zwei solcher Einträge, welche in den Zeilen 15 bis 19 und 20 bis 24 zu sehen sind. Jeder Eintrag verfügt über einen Schlüssel, welcher den zu nutzenden Eingabeparameter der Aktivität (vgl. Zeilen 15 bis 16 und 20 bis 21) benennt. Ebenso enthält jeder Eintrag die Eingabeparameter der Richtlinien, auf welche der Eingabeparameter der Aktivität abgebildet werden soll (vgl. Zeilen 18 bis 19 und 23 bis 24). Dabei werden bei der Angabe der

```
demonstrator_3d_printer.slice:
  degree.ActivityInstance:
    name:
      Identifier: "demonstrator_3d_printer.slice"
    definition:
      degree.ActivityReference: "degree.Activity@SliceDef"
    policies:
      degree.MappedPolicyInstanceMap:
        - key:
            Text: "validatePrintSettings"
          value:
            degree.ConstraintOrPolicyInstanceReference: "degree.
↪ ConstraintInstance@ValidatePrintSettingPreSlice"
    parameterMappings:
      degree.ParameterMappingsMap:
        - key:
            Text: "printSettingsId"
          value:
            Text:
              - "validatePrintSettings.printSettingsId"
        - key:
            Text: "policies"
          value:
            Text:
              - "validatePrintSettings.policy"
```

Abbildung 5.18: Definition der Aktivitätsinstanz, welche zum Slicen von 3D Modellen verwendet wird. Die Definition liegt im `yaml`-Format vor und definiert die zugrundeliegende Aktivitätsdefinition, verknüpfte Richtlinien und eine Abbildung von Aktivitätsparametern auf Parameter der Richtlinie.

Eingabeparameter der Richtlinie die zuvor vergebenen Namen der verknüpften Richtlinien verwendet.

Der D°-Compiler überprüft während des Übersetzungsvorgangs, ob es sich bei allen verwendeten Richtlinieninstanzen um sog. geschlossene Richtlinien handelt. Bei einer geschlossenen Richtlinie sind alle Eingabeparameter mit einem konstanten Wert belegt, oder mit einer Quelle, bspw. dem Eingabeparameter der Aktivität, mit welcher die Richtlinie verknüpft ist, verbunden. Ist mindestens ein Eingabeparameter der Richtlinie nicht mit einem Wert belegt, handelt es sich um eine offene Richtlinie. Findet der Compiler in einer Applikation offene Richtlinien, wird der Übersetzungsvorgang mit einer Fehlermeldung abgebrochen.

Der Vorteil der Trennung von Definitionen und Instanzen ist, dass spezifisch für jeden Anwendungsfall Ausprägungen der Sprachelemente erzeugt und verwendet werden

können. Hierdurch werden Redundanzen vermieden. Beispielsweise können mehrere Instanzen von einer Aktivitätsinstanz erzeugt und mit jeweils unterschiedlichen Richtlinien verknüpft werden. Auch können mehrere Instanzen einer Richtlinie erzeugt werden, wobei die Eingabeparameter mit unterschiedlichen Konstanten verknüpft werden können.

Abbildung 5.19 zeigt eine schematische Darstellung, welche die vorherigen Beschreibungen komprimiert darstellt. Es ist erkennbar, wie Instanzen von Richtlinien und Aktivitäten erstellt werden. Es ist auch erkennbar, wie dabei die Verknüpfung von Richtlinien und Aktivitäten bei der Erstellung von Aktivitätsinstanzen erreicht wird. Außerdem zeigt die Darstellung, wie die erzeugten Aktivitätsinstanzen in Applikationen verwendet werden können. Dabei wird noch einmal dargestellt, dass die Richtlinieninstanzen, welche mit den Instanzen der Aktivität verknüpft sind, nicht in der späteren Applikation sichtbar sind.

In diesem Beispiel wurde gezeigt, wie die Richtlinie zur Überprüfung und Limitierung von Druckeinstellungen im vorgestellten Demonstrator umgesetzt ist. Für die beiden anderen Richtlinien, welche im Szenario umgesetzt werden, ist das Vorgehen identisch.

Die Richtlinie, welche die maximale Anzahl von Drucken für ein 3D Modell überprüfen und limitieren kann, wird in der Applikation zum Starten von Druckaufträgen umgesetzt. Dabei ist die Richtlinie an die Aktivität gehängt, welche einen erstellten Druckauftrag in die aktive Druckwarteschlange einhängt, indem der Status des Auftrags auf `pending` gesetzt wird.

Die dritte Richtlinie hat die Aufgabe sicherzustellen, dass Druckaufträge nur ausgeführt werden, wenn der ausführende Drucker das korrekte Material zum Druck verwendet. Dabei können Einschränkungen hinsichtlich Farbe und Material definiert werden. Diese Richtlinie wird mit der Aktivität verknüpft, welche den Zustand der Drucker überprüft und Druckaufträge startet. Die Aktivität wird in der Update-Applikation verwendet, welche automatisch periodisch ausgeführt wird.

Somit werden zwei der drei verwendeten Richtlinien in Applikationen verwendet, welche durch den Kunden aufgerufen werden. Kommt es zu Richtlinienverstößen, wird die Ausführung abgebrochen und dem Aufrufer eine entsprechende Fehlermeldung zurückgegeben. Dies erlaubt es dem Aufrufer die gewählten Druckparameter anzupassen und die Applikation erneut aufzurufen.

Die dritte Richtlinie wird in einer Applikation ausgeführt, welche periodisch ausgeführt wird, aber keine Verbindung zum Kunden hat. Wird ein Verstoß erkannt, wird die aktuelle Iteration abgebrochen und eine Fehlermeldung auf der Konsole ausgegeben. Der Betreiber kann auf diese Fehlermeldung reagieren, indem er das vom 3D Drucker verwendete Material ändert und so ein Durchlaufen der nächsten Iteration ermöglicht.

Somit werden Richtlinienverstöße immer an die Entität gemeldet, welche die Möglichkeit besitzt die Verstöße durch ihre Handlungen zu beseitigen. Dies veranschaulicht, wie Richtlinien in D° zielgerichtet an relevanten Stellen umgesetzt werden und Fehlermeldungen an die relevanten Stellen ausgegeben werden.

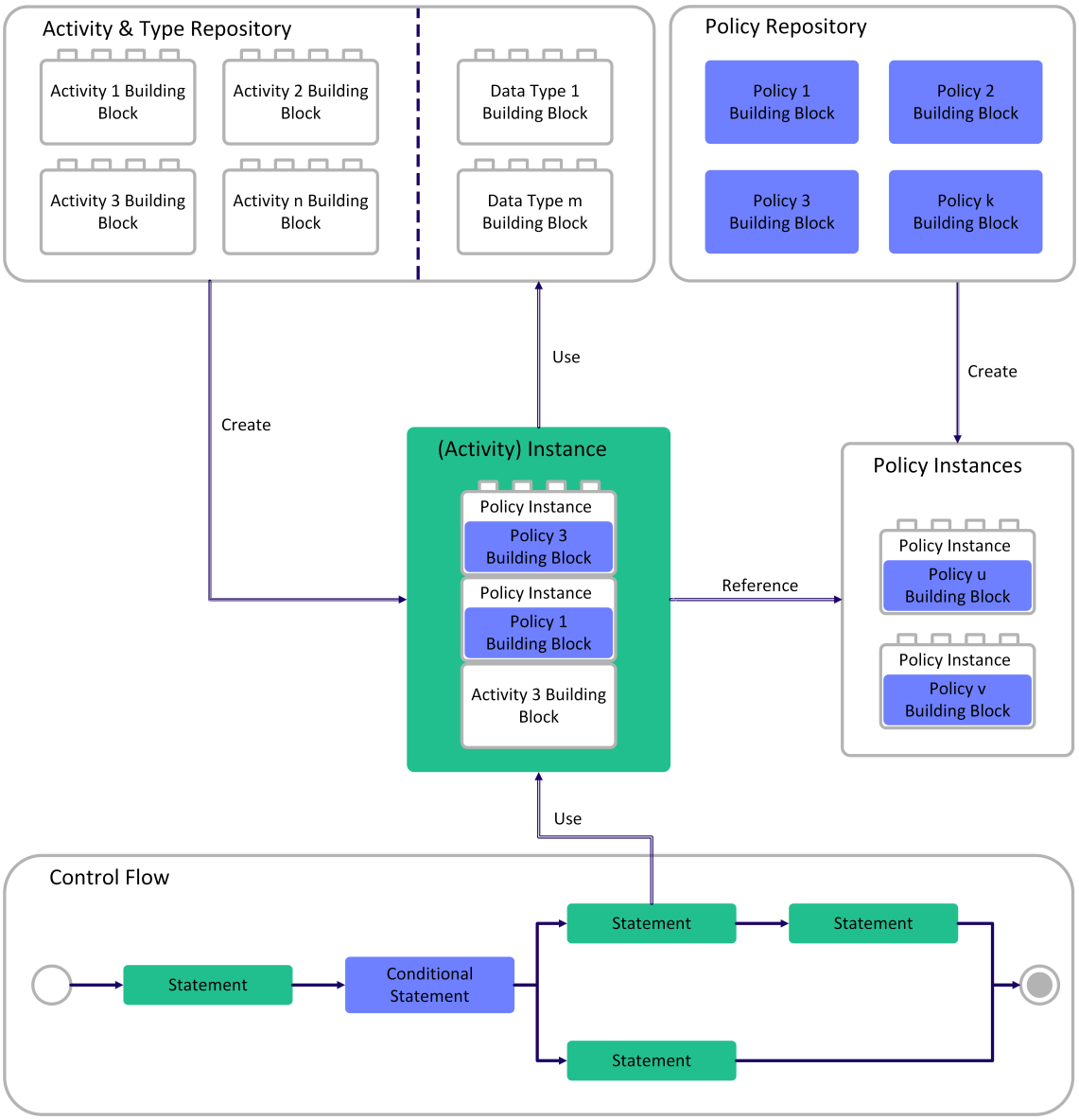

Abbildung 5.19: Schematische Darstellung, wie Definitionen von Aktivitäten und Richtlinien instanziiert, miteinander kombiniert und anschließend in Applikationen verwendet werden. Quelle: [Bru20]

5.5 Effekt der Richtlinien zur Laufzeit

In diesem Abschnitt wird ein Beispiel präsentiert, welches die Auswirkungen der Richtlinien zur Laufzeit darstellt. Zu diesem Zweck wird die D°-Applikation, welche zum Starten von Druckaufträgen verwendet wird, betrachtet. Die Applikation hat als einzigen Eingabeparameter den eindeutigen Bezeichner eines Druckauftrags. Dabei kann die Applikation für jeden Druckauftrag verwendet werden, welcher mit der D°-Applikation, welche in Abschnitt 5.4.3 vorgestellt wurde, verwendet werden.

Die Applikation lädt den Druckauftrag mit dem übergebenem eindeutigen Bezeichner und hängt diesen in die Warteschlange der zu druckenden Druckaufträge ein. Wie in Abschnitt 5.3 beschrieben, sorgt eine andere D°-Applikation dafür, dass periodisch überprüft wird, ob Druckaufträge in der Warteschlange an den 3D Drucker übergeben und ausgeführt werden können.

Aus den Anforderungen 1 und 4 in Abschnitt 5.1.5 folgt, dass der Betreiber Richtlinien einhalten muss, welche einschränken, wie oft ein 3D Modell ausgedruckt werden kann. Diese Richtlinie wird in der vorgestellten D°-Applikation zum Starten von Druckaufträgen umgesetzt. Die Aktivität, welche den Druckauftrag in der Warteschlange einfügt, ist mit einer Richtlinie verknüpft. Diese fragt beim PIP an, wie oft das 3D Modell bereits gedruckt wurde. Hierfür wird der eindeutige Bezeichner des 3D Modells verwendet. Dabei befindet sich die Logik der Richtlinie in der Vorbedingung und wird somit vor der Aktivität ausgeführt.

In diesem Beispiel wird ein 3D Modell betrachtet, dessen Richtlinie besagt, dass das Modell nur viermal gedruckt werden darf. Abbildung 5.20 zeigt einen Screenshot, welcher einen Teil des 3D Druck Portals zeigt. Der Ausschnitt zeigt die Liste von Druckaufträgen und enthält einen einzelnen Eintrag. Es ist erkennbar, dass der Druckauftrag bereits dreimal ausgeführt wurde. Auch die maximale Anzahl an Drucken kann der Oberfläche entnommen werden. Darüber hinaus zeigt die Abbildung, dass der Status des Druckauftrags `pending` ist. Dies bedeutet der Druckauftrag wurde in die Warteschlange eingereiht und wartet auf die Zuteilung zu einem 3D Drucker. Ebenfalls kann der Abbildung entnommen werden, dass der Druckauftrag aktuell Platz 0 in der Warteschlange hat, d.h. es ist der Druckauftrag der als nächstes ausgeführt wird. Im unteren rechten Bereich des Screenshots findet sich eine Benachrichtigung, welche über den erfolgreichen Start des Druckauftrags informiert. Diese Benachrichtigungen werden vom 3D Druck Portal verwendet, um die Ergebnisse von Aktionen, die im Backend oder beim 3D Druck Service des Betreibers ausgeführt werden, anzuzeigen.

Sobald der Druckauftrag einem 3D Drucker zur Ausführung zugeordnet wurde, wechselt der Auftrag in den Status `running`. Die D°-Applikation, welche die Warteschlange aktualisiert, überprüft bei jeder Aktualisierung den Status des 3D Druckers. Sobald erkannt wird, dass der 3D Drucker den Druck abgeschlossen hat, wird der Status des Druckauftrags auf `finished` gesetzt. Darüber hinaus wird im PIP der Zähler für durchgeführte Drucke für das gedruckte Objekt inkrementiert. Der aktualisierte Zustand des Druckauftrags wird dem Kunden in der Liste der Druckaufträge im 3D Druck Portal angezeigt.

Versucht der Nutzer des 3D Druck Portals den Druckauftrag erneut zu starten, wird die selbe D°-Applikation aufgerufen. Dabei wird die Applikationslogik bis zu der Aktivität

Abbildung 5.20: Ein Screenshot des 3D Druck Portals, welcher die Notification nach erfolgreichem Starten eines Druckauftrags zeigt.

ausgeführt, welche den Druckauftrag in die Warteschlange einhängt. An dieser Stelle wird die Richtlinie zur Beschränkung der Anzahl von Drucken ausgeführt. Es wird festgestellt, dass die maximale Anzahl an Drucken für den gegebenen Druckauftrag bereits erreicht ist. Aufgrund dessen, wird die Ausführung der Applikationslogik abgebrochen und dem Aufrufer eine Fehlermeldung mit einem Hinweis auf den Richtlinienverstoß zurückgegeben.

Abbildung 5.21 zeigt einen Screenshot der Liste von Druckaufträgen aus dem 3D Druck Portal. Der einzige enthaltene Druckauftrag ist derselbe wie in Abbildung 5.20. Der Unterschied ist, dass der Druckauftrag jetzt bereits viermal gedruckt wurde und damit das in der Richtlinie definierte Maximum von Drucken erreicht hat. Im unteren Bereich des Screenshots ist die Benachrichtigung zu sehen, die im Portal angezeigt wird, wenn der Nutzer versucht diesen Druckauftrag erneut zu starten. Die Benachrichtigung enthält

die Fehlermeldung, welche den Nutzer darüber informiert, dass eine Richtlinienverletzung vorliegt.

Abbildung 5.21: Ein Screenshot des 3D Druck Portals, welcher die Notification nach einem fehlgeschlagenen Starten eines Druckauftrags zeigt. Dabei ist das Starten aufgrund eines Richtlinienverstoßes fehlgeschlagen.

KAPITEL 6

Diskussion

In diesem Kapitel findet eine Bewertung der erzielten Ergebnisse statt. Dabei wird ein Fokus auf die Erfüllung der ursprünglich in Abschnitt 3.3 definierten Forschungsfragen gelegt.

6.1 Kategorien von Usage Control Richtlinien

Forschungsfrage 1 – Welche Arten von Usage Control Richtlinien gibt es und was zeichnet die unterschiedlichen Gruppen aus?

Es ist möglich beliebige Usage Control Richtlinien zu definieren. Ob und wie diese technisch umsetzbar sind, ist nicht immer sofort ersichtlich. Auch ist es nicht ohne Weiteres möglich unterschiedliche Usage Control Lösungen hinsichtlich ihrer Mächtigkeit miteinander zu vergleichen.

Das Praxisbeispiel in Abschnitt 4.1.3 zeigt, wie die vorgestellte Taxonomie für Usage Control Richtlinien verwendet werden kann, um Richtlinien zu kategorisieren. Der Aufbau der Taxonomie orientiert sich am UCON_{ABC} Modell, welches eine grundlegende Arbeit im Bereich Usage Control ist. Hierdurch ist sichergestellt, dass sich die Taxonomie verwenden lässt, um entsprechende Arbeiten zu analysieren und zu bewerten.

Durch die Taxonomie wird es möglich Richtlinien in Kategorien einzuteilen und zu vergleichen. Dabei können Vergleiche sowohl in der Art wie die Richtlinien umgesetzt werden, als auch in der Komplexität der Umsetzung durchgeführt werden. Ebenfalls ermöglicht die Taxonomie den Vergleich von verschiedenen Usage Control Mechanismen. Hierzu wird betrachtet, welche Kategorien von Richtlinien von den jeweiligen Lösungen umgesetzt werden können.

Grundsätzlich kann die vorgestellte Taxonomie nicht für alle Vergleiche genutzt werden. Aufgrund der beliebigen Komplexität von Usage Control Richtlinien ist eine eindeutige Einordnung in die Taxonomie nicht immer möglich, oder es muss zunächst eine Zerlegung der Richtlinien in einzelne Teilaussagen erfolgen. Daraus folgt, dass unter Umständen im Einzelfall eine Betrachtung notwendig ist, bei der die Taxonomie ggf. nur als Grundlage verwendet werden kann.

Zusammenfassend kann festgestellt werden, dass die Taxonomie eine gute Grundlage sein kann, um Richtlinien und Usage Control Mechanismen miteinander zu vergleichen. Insbesondere in komplexen Anwendungsfällen reichen die Möglichkeiten der Taxonomie jedoch nicht für aussagekräftige Vergleiche aus und können allenfalls als Grundlage für weitere Untersuchungen verwendet werden.

6.2 Usage Control als fester Bestandteil im Programmiersystem

Forschungsfrage 2 – Wie können erweiterbare Usage Control Mechanismen als fester Bestandteil in ein Programmiersystem integriert werden?

Das zuvor präsentierte Praxisbeispiel zeigt, dass D° verwendet werden kann, um eine Vielzahl von unterschiedlichen Richtlinien umzusetzen. Darüber hinaus zeigt der Demonstrator, welcher in Kapitel 5 vorgestellt wurde, dass sowohl D° im Allgemeinen, als auch die implementierten Mechanismen zur Datennutzungskontrolle flexibel genug sind, um durch Erweiterungen in verschiedensten Domänen verwendet zu werden. Außerdem zeigen der Demonstrator, sowie die Arbeiten Bruckner u. a. [Bru20] und Bruckner u. a. [Bru21b], wie die Datennutzungskontrolle während der Übersetzung zu einem festen Bestandteil einer D°-Applikation wird.

Durch die entwickelte Richtlinien-API, welche von allen Richtlinien innerhalb von D° umgesetzt werden muss, können zu verschiedenen Zeitpunkten der Applikationsausführung unterschiedliche Richtlinien ausgeführt werden [Bru20; Bru21b]. In Abschnitt 4.1.3 wurde bereits aufgezeigt, dass durch D° ein großer Anteil der Richtlinien, welche der Taxonomie aus Abschnitt 4.1.3 entspringen, umgesetzt werden können. Neben Richtlinien, die sich direkt auf die verwendeten Daten bzw. zusätzliche externe Daten oder den Applikationsstatus beziehen, können auch solche in D° umgesetzt werden, die sich auf Metadaten beziehen. Dabei sind mögliche Quellen für Metadaten beispielsweise Tags, die mit Aktivitäten verknüpft sind, welche in der ausführenden Applikation verwendet werden, oder ein zentraler PIP. Durch die Möglichkeit der Nutzung von zentralen PIPs kann die Datennutzungskontrolle von D° gleichzeitig und kooperativ mit anderen Lösungen zur Datennutzungskontrolle verwendet werden.

Für D°-Richtlinien wird die eigentliche Richtlinienlogik durch eine Implementation in der Hostsprache bereitgestellt. Jede Logik, die sich in der Hostsprache abbilden lässt, kann auch in einer D°-Richtlinie verwendet werden. Für die vorgestellte Implementation von D°wird die universelle Programmiersprache Java verwendet. Aus diesem Grund ist die Mächtigkeit der Datennutzungskontrolle sehr hoch. Es gibt jedoch verschiedene Aspekte, welche die Mächtigkeit einschränken.

Zum einen ist es nicht möglich Richtlinien umzusetzen, welche nicht durch den Funktionsumfang der verwendeten Hostsprache abgebildet werden. Beispielsweise ist es durch die Verwendung von Java als Hostsprache nicht möglich direkt auf die Inhalte von CPU Registern zuzugreifen bzw. diese zu verändern. Ebenso kann nicht direkt mit der im System verwendeten Hardware interagiert werden. Für diese Einschränkung existiert ein Workaround. Innerhalb der Implementation von Richtlinien können spezielle Applikationen, welche in anderen Programmiersprachen implementiert wurden, ausgeführt werden. Dieses Vorgehen sorgt allerdings für eine erhebliche Komplexität und ist fehleranfällig.

Die nächste Einschränkung ergibt sich durch die Art und Weise wie Richtlinien in D° ausgewertet werden. Die Auswertung von Richtlinien findet zu diskreten Zeitpunkten statt, welche fest mit der Ausführung der Applikationslogik verbunden sind. Kontinuierliche Überprüfungen können mit D° nicht vorgenommen werden. Ebenso ist es nicht möglich Richtlinienüberprüfungen losgelöst von der Applikationslogik zu beliebigen Zeitpunkten durchzuführen.

Somit ist es nicht möglich mit D° Richtlinien umzusetzen, welche Überprüfungen abseits der diskreten Zeitpunkte, die für die Überprüfung von Richtlinien vorgesehen sind, erfolgen. Ein Beispiel für eine solche Richtlinie kann im gezeigten Praxisbeispiel gefunden werden: Die Nutzung von Daten wird für einen gewissen Zeitraum ermöglicht und die Daten müssen anschließend gelöscht werden. D° kann diesen Zeitraum nicht kontinuierlich überwachen. Es ist nur möglich zu überprüfen, ob der erlaubte Zeitraum verlassen wurde, wenn die Richtlinien in der Applikation ausgeführt werden, d.h. während der Ausführung der Applikationslogik.

Zusammenfassend kann festgestellt werden, dass D° über flexible und erweiterbare Mechanismen zur Datennutzungskontrolle verfügt. Außerdem kann festgestellt werden, dass diese Mechanismen während des Übersetzungsvorgangs in die erzeugte Applikation integriert werden. Somit ist Datennutzungskontrolle sowohl auf der Ebene der Programmiersprache, als auch auf der Ebene der ausführbaren Applikationen ein fester Bestandteil von D°. Somit ist die zweite Forschungsfrage durch die Ergebnisse in diesem Dokument und anhängige Arbeiten beantwortet.

6.3 Kapselung der Komplexität von Usage Control

Forschungsfrage 3 – Wie kann die Komplexität der korrekten Anwendung von Usage Control Mechanismen gekapselt werden, damit sie nicht Aufgabe des Applikationsentwicklers ist?

Aufgrund der Integration von Usage Control Mechanismen existiert eine Komplexität in D°, welche mit der korrekten Anwendung dieser Mechanismen verbunden ist. Ein Ziel ist es diese Komplexität zu kapseln und so den Entwickler, welcher Anwendungen mit D° entwickelt, zu entlasten. Zu diesem Zweck setzt D° das Programmierparadigma der policy-agnostischen Programmierung um. Dieses basiert auf der Trennung von Applikationslogik und Mechanismen zur Datennutzungskontrolle während der Entwicklungszeit und einer anschließenden Verknüpfung in einer späteren Phase des Entwicklungszyklus. Bei D° ist diese spätere Phase die Übersetzung in eine ausführbare Applikation. Ziel ist es, dass die Entwickler sich während der Entwicklung von Anwendungen nicht um die Verwendung von Datennutzungskontrolle in der Applikation beschäftigen müssen. Dies kann vor- bzw. nachgelagert geschehen und unter Umständen auch durch andere Entwickler durchgeführt werden, welche die notwendige Expertise im Hinblick auf die korrekte Verwendung von Datennutzungskontrolle haben.

Durch das Konzept von Richtlinien findet in D° die Definition von Richtlinien, sowie die Bereitstellung von Implementationen, getrennt von der Applikationslogik statt. Hierdurch wird eine effektive Kapselung der Komplexität bei der Definition und Implementation von Richtlinien erreicht. Dies entlastet den Entwickler, welcher Richtlinien für D° entwickelt, insofern, als dass Richtlinien ohne Kenntnis und Berücksichtigung von Aktivitäten bzw. Applikationen erfolgen können.

Einen zusätzlichen Beitrag zur Kapselung der Komplexität wird durch die Möglichkeiten zur Komposition von Aktivitäten und Richtlinien geleistet. D° ermöglicht keine direkte Verwendung von Richtlinien im Programmcode von Applikationen. Stattdessen müssen alle Richtlinien, die verwendet werden sollen, mit den entsprechenden Aktivitäten verknüpft

werden. Daraus folgt, dass auch die Integration von Richtlinien in Applikationen an einer zentralen Stelle erfolgen kann, welche unabhängig von der eigentlichen Applikation ist.

Ein weiterer Vorteil ist, dass der Entwickler, welcher die eigentliche Applikation entwickelt, Richtlinien und deren Verwendung nicht berücksichtigen muss. Es ist dem Entwickler nicht möglich, auf Basis des Applikationscodes abzuleiten, ob die Applikation Richtlinien verwendet bzw. an welchen Stellen diese verwendet werden. Dies ist eine große Entlastung für den Entwickler, da keine Kenntnis über Usage Control im Allgemeinen oder die korrekte Anwendung der D°-Mechanismen notwendig sind, um Applikationen mit D° zu entwickeln.

Somit wird durch die Umsetzung des Paradigmas der policy-agnostischen Programmierung in D° eine Kapselung der Komplexität von Usage Control zur Entwicklungszeit, sowie die Entlastung der Entwickler erreicht. Durch den Einsatz von Codegenerierung zur Übersetzungszeit wird die Komplexität der korrekten Verknüpfung der Richtlinienumsetzung mit der Applikationslogik adressiert. Durch den Einsatz eines generativen Ansatzes ist die korrekte Verknüpfung von Richtlinen und Applikationslogik für beliebige Szenarien gewährleistet. Durch diese Automatisierung muss sich der Entwickler nicht mit der Integration von notwendigen Richtlinien in die ausführbare Applikation befassen. Die damit verbundene Komplexität ist vollständig in den D°-Compiler verlagert und gelöst.

Als Nächstes wird betrachtet welche Einschränkungen sich durch die gewählte Umsetzung der policy-agnostischen Programmierung für D° ergeben. Dabei wird auch die allgemeine Eignung des Programmierparadigmas für die Umsetzung von Datennutzungskontrolle untersucht.

Die größte Limitierung ergibt sich daraus, dass die Verknüpfung von Applikationslogik und Datennutzungskontrolle automatisiert durch den Compiler durchgeführt wird. Durch dieses generative Vorgehen folgt, dass nur solche Richtlinien in D° umgesetzt werden können, welche durch die definierte Schnittstelle für Richtlinien abgebildet werden können. Falls eine Richtlinie in einer Applikation verwendet werden soll, welche sich nicht durch die Schnittstelle abbilden lässt, gibt es keinen Umweg, welcher beispielsweise eine manuelle Integration ermöglicht.

Aus diesem Grund wurde die Schnittstelle für Richtlinien so definiert, dass eine Vielzahl von Richtlinien abgebildet werden kann. Dabei kann die Schnittstelle in zwei Teile unterteilt werden. Ein statischer Teil, welcher zu festen Zeitpunkten der Applikationsausführung aufgerufen wird, erlaubt die Definition von Vor- und Nachbedingungen. Der dynamische Anteil wird ausgeführt, wenn bestimmte Funktionen in der verwendeten Hostsprache benutzt werden. Der Demonstrator in Kapitel 5, sowie die Betrachtungen in den Abschnitten 4.1.3 und 4.1.3 zeigen, dass D° dazu geeignet ist eine Vielzahl verschiedener Richtlinien umzusetzen. Dennoch existieren Einschränken hinsichtlich der Richtlinien, welche mit D° umgesetzt werden können. Bedingt durch die enge Koppelung von Richtlinienüberprüfungen und der Ausführung der Applikationslogik, können generell keine Richtlinien umgesetzt werden, welche Überprüfungen oder Aktionen zu Zeitpunkten erfordern, welche nicht in der Ausführungszeit der Applikationslogik liegen.

Das Programmierparadigma der policy-agnostischen Programmierung wurde zuerst im Bereich der Informationsflusskontrolle eingesetzt [Yan15a]. Durch die Ergebnisse, die in der vorliegenden Arbeit präsentiert werden, wurde aufgezeigt, dass dieses Programmierparadigma auch für das übergeordnete Thema der Datennutzungskontrolle verwendet werden

kann. Dabei wird ersichtlich, dass es sich hierbei um ein gut geeignetes Vorgehen handelt, da die Datennutzungskontrolle von D° auch im Vergleich zu anderen Lösungsansätzen gut abschneidet (siehe Abschnitt 4.1.3).

6.4 Remote Evaluation und Usage Control

Forschungsfrage 4 – Durch welche Designentscheidungen können Usage Control Mechanismen bei der Erlangung der Datensouveränität unterstützt werden?

Die Integration von Datennutzungskontrolle in Applikationen erlaubt es Daten sicher auf eigenen Systemen zu verarbeiten. In Szenarien, in denen Daten kooperativ genutzt werden, ergibt sich dennoch das Problem, dass Daten auf Systemen verarbeitet werden müssen, welche nicht unter der Kontrolle des Rechteinhabers sind.

Durch eine Implementation des Remote Evaluation Paradigmas für D° wurde eine Möglichkeit geschaffen, wie Rechteinhaber die Nutzung ihrer Daten auf ihren eigenen Systemen für Dritte ermöglichen können. Gleichzeitig stehen dabei dem Rechteinhaber die Usage Control Mechanismen von D° zur Verfügung und ermöglichen so die Kontrolle über die Nutzung der eigenen Daten [Bru21a].

Neben diesem Vorteil für den Rechteinhaber ergibt sich gleichzeitig auch ein Nachteil für den Datennutzer, welcher die Applikation zur Verfügung stellt. Es ist erforderlich, dass der Datennutzer die Applikation an den Rechteinhaber zur Ausführung versendet. In der entwickelten Implementation für D° geschieht dies nicht in Form von ausführbaren Applikationen, sondern als Programmcode (inkl. notwendiger Abhängigkeiten). Der Rechteinhaber übersetzt die Applikationen zur Ausführung dann auf seinen Systemen. Daraus folgt, dass der Datennutzer die von ihm verwendete Applikation auf der Ebene des Quellcodes für den Rechteinhaber offenlegen muss.

Somit wird die Problematik, dass der Rechteinhaber nicht möchte, dass seine Daten auf die Systeme Dritter übertragen werden müssen, umgedreht. Es ist unter Umständen im Interesse des Datennutzers, dass die Applikationen, welche er zur Datenverarbeitung verwendet, nicht an Dritte übertragen werden müssen. Ein möglicher Grund hierfür ist, dass die Applikationen Algorithmen enthalten können, welche Geschäftsgeheimnisse abbilden.

Somit ist möglich, durch eine Kombination des Remote Evaluation Paradigmas und Mechanismen zur Datennutzungskontrolle, die Risiken für den Rechteinhaber zu reduzieren. Dabei muss betrachtet werden, dass es sich nicht um eine optimale Lösung für jedes Szenario handelt.

Die Parteien, die an der kooperativen Datennutzung beteiligt sind, müssen auf Einzelfallbasis entscheiden, wie die Datennutzung durchgeführt werden soll. Dabei ist die Umsetzung des Remote Evaluation Paradigmas für D° ein valider Kandidat, welcher zur Umsetzung in Betracht gezogen werden kann.

Zusammenfassend kann festgestellt werden, dass mit dem Remote Processing Prozess ein Werkzeug zur Verfügung steht, welches kooperative Datennutzung und die Wahrung der Datensouveränität begünstigt. Nichtsdestotrotz ist dieses Werkzeug nicht in allen Szenarien anwendbar, da ein Teil Problematik vom Rechteinhaber der Daten zum Datennutzer, welcher die datenverarbeitende Applikation bereitstellt, verlagert wird.

6.5 Zusammenfassung

Abschließend kann festgestellt werden, dass alle vier Forschungsfragen, welche in Abschnitt 3.3 definiert wurden, durch die vorliegende Arbeit beantwortet werden.

Die vorgestellte Taxonomie von Richtlinien kann verwendet werden, um Richtlinien und Usage Control Ansätze miteinander zu vergleichen. Mit D°, dem verwendeten Compiler und von diesem erzeugten Applikationen steht ein mächtiges Programmiersystem zur Verfügung. Dieses besitzt durch die Umsetzung der policy-agnostischen Programmierung ausdrucksstarke Usage Control Mechanismen und kann in diversen Szenarien verwendet werden, um einen Beitrag zur Datensouveränität zu leisten. Durch die Umsetzung der policy-agnostischen Programmierung ist es gelungen die Komplexität der Anwendung der Usage Control Mechanismen zu kapseln und aus der eigentlichen Applikationsentwicklung zu entfernen. Die Komplexität der korrekten Integration in die ausführbare Applikation wurde durch die Verwendung von Codegenerierung im D°-Compiler aufgelöst, wodurch Anwender von D° zusätzlich entlastet werden. Mit dem Remote Processing werden die Möglichkeiten zur Datennutzungskontrolle von D° auch in Szenarien verfügbar gemacht, in denen es zwingend erforderlich ist, dass Daten auf den Systemen des Rechteinhabers verbleiben.

Somit wird durch die vorgestellten Ergebnisse ein erheblicher Beitrag zum Thema der Datennutzungskontrolle und damit der Datensouveränität geleistet. Dabei wird der Hauptbeitrag mit D° auf der technischen Seite der Thematik geleistet. Auch das Remote Processing stellt einen technischen Beitrag dar und bietet einen alternativen Lösungsansatz für Szenarien der kooperativen Datennutzung. Für jede der Forschungsfragen konnten Einschränkungen identifiziert werden, welche auf den gewählten Lösungsansätzen beruhen. Grundsätzlich handelt es sich bei den erkannten Einschränkungen nicht um drastische Hindernisse oder Beschränkungen, welche die vorgestellte Lösung unbrauchbar machen. Stattdessen werden durch die Einschränkungen die möglichen Szenarien, in welchen die erzielten Ergebnisse sinnvoll eingesetzt werden können, definiert.

KAPITEL 7

Zusammenfassung

In diesem Kapitel wird eine kurze Zusammenfassung über die erzielten Arbeitsergebnisse und erlangten Schlussfolgerungen gegeben. Darüber hinaus werden offen gebliebene Punkte und Ansatzpunkte für mögliche künftige Arbeiten aufgezeigt. Abschließend wird auf die ursprüngliche Motivation der Arbeit eingegangen und beschrieben, welchen Beitrag die vorliegende Arbeit zu den beschriebenen Problemen hat.

Es wurde eine Taxonomie für Usage Control Richtlinien vorgestellt. Diese Taxonomie kann als Werkzeug verwendet werden, um Richtlinien zu kategorisieren und miteinander zu vergleichen. Außerdem erlaubt die Taxonomie den Vergleich unterschiedlicher Usage Control Ansätze. Hierzu wird betrachtet, welche Richtlinienkategorien von den jeweiligen Ansätzen umgesetzt werden können.

Des Weiteren wurde ein Verfahren erarbeitet, wie Mechanismen zur Datennutzungskontrolle als fester Bestandteil in eine Programmiersprache integriert werden können. Dabei wurde großer Wert darauf gelegt, dass die Komplexität, welche durch die Datennutzungskontrolle in die Programmiersprache eingefügt wird, gekapselt wird. Durch diese Kapselung wird entsprechenden Experten die Möglichkeit gegeben die notwendigen Arbeiten isoliert von der eigentlichen Applikationsentwicklung durchzuführen. Ein weiterer zentraler Aspekt, welcher fortlaufend betrachtet wurde, ist die Erweiterbarkeit der entwickelten Lösung. Diese wurde erreicht, indem das Programmierparadigma der policy-agnostischen Programmierung, welches ursprünglich im Bereich der Informationsflusskontrolle entwickelt und eingesetzt wurde, für die Datennutzungskontrolle adaptiert wurde.

Dabei wurden Schnittstellen für Aktivitäten (– welche im entwickelten Konzept atomare Bausteine darstellen, welche Logik ausführen können –) und Richtlinien (– elementare Bausteine zur Umsetzung der Datennutzungskontrolle –) entwickelt. Des Weiteren wurde ein Verfahren entwickelt, welches die Verknüpfung der einzelnen Sprachelemente erlaubt, um komplexe Elemente zu erhalten, welche in Applikationen verwendet werden können, dem Entwickler jedoch eine einfache Schnittstelle bieten. Durch die Beschreibung eines Compilers und einer Ausführungsumgebung für die entwickelten Applikationen, wurde die Programmiersprache komplementiert.

Alle erarbeiteten Konzepte und Verfahren wurden dabei in einem Prototypen umgesetzt, woraus die Programmiersprache D° entstanden ist. Auf Basis von D° wurde ein Demonstrator entwickelt, welcher repräsentativ für Systeme ist, in denen Daten mit anhängigen Richtlinien zwischen mehreren Parteien geteilt werden und eine Datennutzung unter Einhaltung der Richtlinien erlauben. Dabei war es möglich durch den Demonstrator die Validität der entwickelten Konzepte aufzuzeigen.

Ausgehend von den erarbeiteten Konzepten und dem entwickelten Prototypen wurde eine Umsetzung des Remote Evaluation Paradigmas, welches die Vorteile der Datennutzungskontrolle integriert, konzipiert und umgesetzt. Hierdurch wurden die möglichen Einsatzbereiche der entwickelten Lösung zusätzlich um Szenarien erweitert, in denen der Rechteinhaber der Daten vermeiden möchte seine Daten auf den Systemen Dritter verarbeiten zu lassen.

Auf Basis eines Praxisbeispiels wurde aufgezeigt, dass die entwickelte Lösung auch in realen Umgebungen (– in diesem Fall den International Data Spaces –) eingesetzt werden kann. Dabei ist die Mächtigkeit der Datennutzungskontrolle vergleichbar mit oder besser als alternative Lösungen in der Zielumgebung.

Die entwickelten Artefakte sind in folgendem git-Repository als Open Source Software verfügbar: https://github.com/FraunhoferISST/data-app-language Dabei sind die folgenden Artefakte veröffentlicht:

- Der D°-Compiler.
- Ein core-Sprachmodul, welches diverse Datentypen, Aktivitäten und Richtlinien enthält.
- Die Ausführungsumgebung für das sog. Remote Processing, welches das Remote Evaluation Paradigma für D° umsetzt.
- Das Sprachmodul, welches für den Demonstrator entwickelt wurde und eine Vielzahl von Datentypen, Aktivitäten und Richtlinien enthält.
- Die D°-Applikationen, welche zur Umsetzung des Demonstrators entwickelt wurden.

7.1 Ausblick

Es existieren diverse Anknüpfungspunkte, welche die Ergebnisse der vorliegenden Arbeit weiter anreichern können. Beispielsweise fehlen Komponenten, um eine produktive Applikationsentwicklung mit D° zu ermöglichen und die Sprache aus dem Status des Prototypen zu heben. Dies umfasst neben unterstützenden Komponenten, wie eine Integration in IDEs, auch wichtige und komplexe Komponenten, wie einen dedizierten Debugger und Testframeworks.

Durch die Hervorhebung der D°-Syntax, Autovervollständigung und komfortable Navigation im Code bietet die Integration von D° in IDEs einen massiven Zugewinn an Komfort und Effizienz für Entwickler. Dabei könnte in diesem Zusammenhang untersucht werden, ob die Erstellung der Erweiterungen für die IDEs automatisiert werden kann. D° setzt auf eine ANTLR[1]-Grammatik. Wenn die Artefakte, welche zur Integration von D° in IDEs notwendig sind, automatisch generiert werden könnten, wäre dies auf diverse andere Sprachen, welche ebenfalls ANTLR verwenden, übertragbar.

Für die produktive und effiziente Softwareentwicklung ist ein Debugger ein zwingend notwendiges Werkzeug. Unter dem Aspekt existiert für einen D°-Debugger eine besondere Herausforderung: Die Applikationslogik, welche in D°-Code abgebildet wird, verwendet zur Ausführungszeit beliebigen Programmcode in der verwendeten Hostsprache. Für die besten

1 https://www.antlr.org/

Einblicke in den Ablauf müsste der Debugger diese beiden Ausführungsebenen parallel verfolgen und darstellen können.

Ein Debugger, welcher nur den D°-Code verfolgen kann, könnte allerhöchstens zur groben Lokalisation von Fehlern verwendet werden, würde sich jedoch nicht dazu eignen einen Fehler in der Logik einer Aktivität zu identifizieren. Auf der anderen Seite könnte ein existierender Debugger für die Hostsprache verwendet werden, um Fehler an ihrer Quelle zu identifizieren. Das Problem dabei ist, dass der D°-Applikationscode auf der Ebene der Hostsprache nicht mehr verfügbar ist und es somit nur mit hohem manuellem Aufwand möglich ist den Ablauf einer D°-Applikation nachzuverfolgen.

Ähnlich wichtig wie ein Debugger ist eine Möglichkeit entwickelte D°-Applikationen automatisiert zu testen. Mithilfe eines Testframeworks sollte es ermöglicht werden Testfälle für eine Applikation zu definieren, welche im Rahmen des Übersetzungsvorgangs automatisiert ausgeführt werden. Solche Lösungen sind aus anderen Programmiersprachen bekannt. Dabei würde es ausreichen, wenn das Testframework auf Applikationsebene funktioniert, d.h. überprüfen kann, ob eine Applikation bei bestimmten Eingaben eine erwartete Ausgabe erzeugt. Aktivitäten und Richtlinien können auf der Ebene der Hostsprache getestet werden und die dort zur Verfügung stehenden Mechanismen verwenden.

Nicht nur im Bereich der Programmiersprache, auch im Bereich der Datennutzungskontrolle können noch Erweiterungen durchgeführt werden, welche einen Mehrwert für D° darstellen können. Beispielsweise erlaubt D° die Verknüpfung von Aktivitäten mit Richtlinien. Dieses Konzept könnte für Datentypen erweitert werden, hinsichtlich der Definition von Datentypen und der anschließenden Erzeugung von Instanzen, welche mit Richtlinien verknüpft sind. In der D°-Applikation würden dann Instanzen dieser Instanzen verwendet.

Diese Funktionalität kann den Aufbau von D°-Applikationen vereinfachen, wenn an allen Stellen, an denen ein bestimmter Datentyp verwendet wird, gewisse Richtlinien überprüft werden müssen. Durch die direkte Verknüpfung der Richtlinie mit den Datentypen könnten mögliche Redundanzen vermieden werden. Aktuell ist es notwendig, dass die entsprechenden Richtlinien mit allen verwendeten Aktivitäten verknüpft wird.

Bevor diese Funktionalität umgesetzt werden kann, wäre es notwendig eine wichtige Frage zu klären. Es muss zunächst bestimmt werden, wie verschiedene Instanzen eines Datentyps, welche mit unterschiedlichen Richtlinien verknüpft sind, zueinander kompatibel sind. Auf der Ebene der Attribute sind die Instanzen vollständig zuweisungskompatibel. Allerdings ist es notwendig die Richtlinien mit einzubeziehen, andernfalls könnten diese durch einfache Zuweisungen umgangen werden. Ebenso könnte es zu Situationen kommen, in denen nach einer (falschen) Zuweisung Richtlinien für eine Instanz überprüft werden, welche eigentlich nicht notwendig sind.

Diese Funktionalität würde primär den Nutzungskomfort von D° steigern und weniger einen Beitrag zur Mächtigkeit der Datennutzungskontrolle leisten. Es besteht auch die Möglichkeit Funktionalitäten zu entwickeln, welche die Mächtigkeit der Datennutzungskontrolle positiv beeinflussen. Eine große Limitation ist, dass die Richtlinien in D° nur zu festen Zeitpunkten überprüft werden. Eine mögliche Erweiterung besteht darin einen zusätzlichen Ausführungspunkt für Richtlinien zu schaffen. Dieser wird zu Zeitpunkten ausgeführt, welche zur Laufzeit durch die Richtlinie selber bzw. durch die verarbeiteten Daten festgelegt wird.

Dabei kann dieser zusätzliche Endpunkt direkt in der Applikation ausgeführt werden. Dies würde bedeuten, dass die Applikation nicht beendet werden darf bevor alle zukünftigen Richtlinienüberprüfungen ausgeführt wurden. Eine Alternative wäre die Bereitstellung eines zusätzlichen Services, welcher die Ausführung übernimmt. Richtlinien würde die Möglichkeit gegeben bei diesem Service die Ausführung eines Endpunkts in der Zukunft zu beantragen. Der Service wäre dann dafür verantwortlich zu den definierten Zeitpunkten die beauftragten Überprüfungen vorzunehmen.

Der Vorteil wäre, dass die Umsetzung zukünftiger Überprüfungen unabhängig von der eigentlichen Applikation sind. Nachteilig zu bewerten ist, dass dieser zusätzliche Service nicht den Einblick in den Zustand der Applikation und der enthaltenen Daten hat, was ursprünglich eines der Hauptargumente für die Entwicklung von D° war.

Zusammenfassend kann festgestellt werden, dass an diversen Stellen von D° Raum für zukünftige Arbeiten gegeben ist.

7.2 Beitrag zur Datensouveränität

Wie in Kapitel 1 beschrieben, können Usage Control Mechanismen verwendet werden, um eine technische Umsetzung der Datensouveränität zu gewährleisten. Durch die vorliegende Arbeit werden zwei unterschiedliche Beiträge zur Wahrung bzw. Erlangung der Datensouveränität geleistet.

Der Hauptbeitrag sind die entwickelten Verfahren, welche in der Programmiersprache D° implementiert worden sind. Mithilfe der integrierten Usage Control Mechanismen können diverse Anforderungen an die Datennutzungskontrolle direkt in den datenverarbeitenden Applikationen abgebildet und umgesetzt werden. Dahingehend wurde bei der Entwicklung der Verfahren großer Wert darauf gelegt die Komplexität, welche mit der Integration und Verwendung von Usage Control Mechanismen in einer Applikation einhergeht, von der eigentlichen Applikationsentwicklung abzukapseln. Teile der Komplexität sind dabei vollständig in den Compiler verlagert und gelöst worden. Hierdurch wird die Verwendung der entwickelten Verfahren und die Anwendung von Usage Control für den Nutzer vereinfacht.

Der zweite Beitrag ist die Taxonomie für Usage Control Richtlinien. Diese ermöglicht die Kategorisierung und den Vergleich eigener Richtlinien. Darüber hinaus kann die Taxonomie eine Hilfestellung sein, wenn konkrete Usage Control Ansätze ausgewählt werden sollen, um definierte Anforderungen im Bereich Usage Control umzusetzen.

Betrachtet man das gesamte Themengebiet der Datensouveränität, fällt auf, dass die Ergebnisse der vorliegenden Arbeit sich auf die technischen Aspekte in Form von Usage Control fokussieren. Aus diesem Grund reicht ein Einsatz der vorgestellten Verfahren nicht aus, um alle Bedürfnisse zur Wahrung und Erlangung der Datensouveränität zu erfüllen. Diese müssen zumeist mit organisatorischen Maßnahmen aus dem Bereich der Data Governance ergänzt werden, um eine vollständige Wahrung bzw. Erlangung der Datensouveränität sicherzustellen.

Im Bereich der technischen Umsetzung von Datensouveränität stellen die Ergebnisse der vorliegenden Arbeit keine Lösung für alle Anwendungsbereiche und Szenarien statt. Dies liegt an der Art und Weise wie Usage Control Mechanismen in die entwickelten Verfahren eingebettet sind. Durch die direkte Integration in die datenverarbeitenden Applikationen ist es nicht möglich die vorgestellten Verfahren zu verwenden, um bestehende Software bzw.

Ökosysteme nachträglich mit Usage Control Mechanismen auszustatten. Die intendierten Einsatzszenarien für die vorgestellten Verfahren sind solche, in denen datenverarbeitende Applikationen zunächst entwickelt werden müssen und dabei direkt mit Usage Control Mechanismen ausgestattet werden sollen.

Auch in den Szenarien, auf welche die Ergebnisse dieser Arbeit abzielen, muss zunächst überprüft werden, ob die entwickelten Verfahren angewandt werden können. Dies liegt darin begründet, dass die vorgestellten Verfahren nicht alle denkbaren Richtlinien umsetzen können. Da es möglich ist beliebige Richtlinien zu definieren, existieren immer auch Richtlinien welche nicht mit der vorgestellten Lösung umgesetzt werden können. Dies ist keine besondere Einschränkung für die vorgestellten Ergebnisse, sondern betrifft alle Lösungen für Usage Control gleichermaßen.

Zusammenfassend kann festgestellt werden, dass die Ergebnisse, die in der vorliegenden Arbeit präsentiert werden, einen erheblichen Beitrag im Bereich Usage Control liefern und hierdurch auch einen Beitrag zur Datensouveränität leisten. Nichtsdestotrotz handelt es sich nicht um eine Lösung, welche sämtliche Szenarien abdeckt, da Usage Control ein sehr breites Themengebiet ist, welches sich nicht nur auf die Integration von Datennutzungskontrolle in Applikationen beschränkt. In Szenarien, in denen datenverarbeitende Applikationen entwickelt und direkt mit Usage Control Mechanismen ausgestattet werden sollen, sind die vorgestellten Verfahren eine gute Lösung. Es muss lediglich überprüft werden, ob die notwendigen Richtlinien in dem präsentierten Usage Control Mechanismus abgebildet werden können. Generell zeigen die Ergebnisse der vorliegenden Arbeit, dass es von Vorteil ist, wenn Anforderungen an die Datennutzungskontrolle bereits zur Entwicklungszeit von Applikationen berücksichtigt werden. Wenn diese Berücksichtigung Einzug in die allgemeine Applikations- und Systementwicklung findet, kann hierdurch ein großer Beitrag zur Wahrung der Datensouveränität geleistet werden.

Literatur

[Ala84] Alavi, Maryam: „An assessment of the prototyping approach to information systems development“. *Communications of the ACM* (1984), Bd. 27(6): S. 556–563 (siehe S. 25).

[Ara19] Araujo, Victor, Karan Mitra, Saguna Saguna und Christer Åhlund: „Performance Evaluation of FIWARE: A Cloud-Based Iot Platform for Smart Cities“. *Journal of Parallel and Distributed Computing* (2019), Bd. 132: S. 250–261 (siehe S. 4).

[Aus12] Austin, Thomas H und Cormac Flanagan: „Multiple facets for dynamic information flow“. *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 2012: S. 165–178 (siehe S. 8).

[Aus13] Austin, Thomas H., Jean Yang, Cormac Flanagan und Armando Solar-Lezama: „Faceted execution of policy-agnostic programs“. *Proceedings of the Eighth ACM SIGPLAN workshop on Programming languages and analysis for security*. 2013: S. 15–26 (siehe S. 7).

[Bad19] Bader, Sebastian R und Maria Maleshkova: „Towards Enforceable Usage Policies for Industry 4.0.“ *LASCAR@ ESWC*. 2019: S. 75–84 (siehe S. 1).

[Bao04] Baoshi, Zhu: „Digital rights management for electronic documents“. (2004), Bd. (siehe S. 12).

[Bar09] Bartoletti, Massimo, Costa Gabriele, Martinelli Fabio und Zunino Roberto: „Securing Java with local policies“. (2009), Bd. (siehe S. 12).

[Bey05] Beydeda, Sami, Matthias Book, Volker Gruhn u. a.: *Model-driven software development*. Bd. 15. Springer, 2005 (siehe S. 22).

[Bic18] Bichhawat, Abhishek, Akash Trehan, Jean Yang und Matt Fredrikson: „ESTRELA: Automated Policy Enforcement Across Remote APIs.“ *CoRR* (2018), Bd. (siehe S. 8).

[Bie20] Biegel, Fabian u. a.: „GAIA-X: Driver of digital innovation in Europe“. (Mai 2020), Bd. Zugriff am 24. Januar 2021, von https://www.data-infrastructure.eu/GAIAX/Redaktion/EN/Publications/gaia-x-driver-of-digital-innovation-in-europe.pdf?__blob=publicationFile&v=8 (siehe S. 4).

[Bit20] Bitkom: *Nutzung von Cloud Computing in Unternehmen in Deutschland in den Jahren 2011 bis 2019)*. Zugriff am 20. Januar 2021, von https://de.statista.com/statistik/daten/studie/177484/umfrage/einsatz-von-cloud-computing-in-deutschen-unternehmen-2011/. Juni 2020 (siehe S. 2).

[Boß16] Bosselmann, Steve, Markus Frohme, Dawid Kopetzki, Michael Lybecait, Stefan Naujokat, Johannes Neubauer, Dominic Wirkner, Philip Zweihoff und Bernhard Steffen: „DIME: a programming-less modeling environment for web applications“. *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2016: S. 809–832 (siehe S. 22).

[Bro04] Brooks, Richard R: „Mobile code paradigms and security issues“. *IEEE Internet Computing* (2004), Bd. 8(3): S. 54–59 (siehe S. 20).

[Bru21a] Bruckner, Fabian und Falk Howar: „Utilizing Remote Evaluation for Providing Data Sovereignty in Data-sharing Ecosystems“. *54th Hawaii International Conference on System Sciences, HICSS 2021, Kauai, Hawaii, USA, January 5, 2021*. ScholarSpace, 2021: S. 1–10 (siehe S. 45, 68, 105).

[Bru20] Bruckner, Fabian, Julia Pampus und Falk Howar: „A Framework for Creating Policy-agnostic Programming Languages“. *Proceedings of the 9th International Conference on Data Science, Technology and Applications, DATA 2020, Lieusaint, Paris, France, July 7-9, 2020*. Hrsg. von Hammoudi, Slimane, Christoph Quix und Jorge Bernardino. SciTePress, 2020: S. 31–42 (siehe S. 54, 81, 85, 97, 102).

[Bru21b] Bruckner, Fabian, Julia Pampus und Falk Howar: „A Policy-Agnostic Programming Language for the International Data Spaces“. *Data Management Technologies and Applications*. Hrsg. von Hammoudi, Slimane, Christoph Quix und Jorge Bernardino. Cham: Springer International Publishing, 2021: S. 172–194 (siehe S. 86, 102).

[Bun] Bundesregierung: *Datensouveränität ist höchstes Gebot)*. Zugriff am 23. Januar 2021, von https://www.bundesregierung.de/breg-de/themen/digitalisierung/kanzlerin-bei-digitalgipfel-1686406 (siehe S. 2).

[Car16] Carniani, Enrico, Davide D'Arenzo, Aliaksandr Lazouski, Fabio Martinelli und Paolo Mori: „Usage control on cloud systems“. *Future Generation Computer Systems* (2016), Bd. 63: S. 37–55 (siehe S. 18).

[Che12] Chen, Deyan und Hong Zhao: „Data Security and Privacy Protection Issues in Cloud Computing“. *2012 International Conference on Computer Science and Electronics Engineering*. Bd. 1. IEEE. 2012: S. 647–651 (siehe S. 2).

[Che98] Chess, David M: „Security issues in mobile code systems“. *Mobile agents and security* (1998), Bd.: S. 1–14 (siehe S. 20).

[Cos11] Costa, Gabriele: „On the Security of Software Systems and Services“. Diss. 2011 (siehe S. 12).

[Cza03] Czarnecki, Krzysztof und Simon Helsen: „Classification of model transformation approaches“. *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*. Bd. 45. 3. USA. 2003: S. 1–17 (siehe S. 22).

[Dea99] Dean, Richard Drews: *Formal aspects of mobile code security*. Princeton University Princeton, 1999 (siehe S. 20).

[Eck14] Eckartz, Silja M., Wout J. Hofman und Anne Fleur Van Veenstra: „A Decision Model for Data Sharing“. *Electronic Government*. Hrsg. von Janssen, Marijn, Hans Jochen Scholl, Maria A. Wimmer und Frank Bannister. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014: S. 253–264 (siehe S. 2).

[Eit21] Eitel, Andreas, Christian Jung, Robin Brandstädter, Arghavan Hosseinzadeh, Sebastian Bader, Christian Kühnle, Pascal Birnstill, Gerd Brost, Mark Gall, Fabian Bruckner, Norbert Weissenberg und Benjamin Korth: „Usage Control in the International Data Spaces: Version 3.0“. (2021), Bd. Zugriff am 08. April 2021, von https://internationaldataspaces.org/download/21053/ (siehe S. 4, 26, 55, 57).

[Eit19] Eitel, Andreas, Christian Jung, Christian Kühnle, Fabian Bruckner, Gerd Brost, Pascal Birnstill, Ralf Nagel und Sebastian Bader: „Usage Control in the International Data Spaces: Version 2.0“. (2019), Bd. Zugriff am 24. Januar 2021, von https://www.internationaldataspaces.org/wp-content/uploads/2020/09/IDSA-Position-Paper-Usage-Control-in-IDS.pdf (siehe S. 4, 26).

[Eur14] Eurostat: *Anteil der Unternehmen, die Cloud Computing nutzen, in Europa* nach Wirtschaftsbereich im Jahr 2014)*. Zugriff am 20. Januar 2021, von https://de.statista.com/statistik/daten/studie/381434/umfrage/nutzung-von-cloud-computing-in-unternehmen-nach-wirtschaftsbereich/. Dez. 2014 (siehe S. 2).

[Fel15] Felleisen, Matthias, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi, Eli Barzilay, Jay McCarthy und Sam Tobin-Hochstadt: „The Racket Manifesto“. *1st Summit on Advances in Programming Languages (SNAPL 2015)*. Hrsg. von Ball, Thomas, Rastislav Bodik, Shriram Krishnamurthi, Benjamin S. Lerner und Greg Morrisett. Bd. 32. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015: S. 113–128 (siehe S. 8, 22).

[Fel05] Felmetsger, V. und G. Vigna: „Exploiting OS-level mechanisms to implement mobile code security“. *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*. 2005: S. 234–243 (siehe S. 20).

[Fet03] Fetscherin, Marc und Matthias Schmid: „Comparing the usage of digital rights management systems in the music, film, and print industry“. *Proceedings of the 5th international conference on Electronic commerce*. 2003: S. 316–325 (siehe S. 12).

[Fow10] Fowler, Martin: *Domain-specific languages*. Pearson Education, 2010 (siehe S. 21).

[Fug98] Fuggetta, Alfonso, Gian Pietro Picco und Giovanni Vigna: „Understanding code mobility“. *IEEE Transactions on software engineering* (1998), Bd. 24(5): S. 342–361 (siehe S. 19).

[Fun10] Fung, Benjamin C. M., Ke Wang, Rui Chen und Philip S. Yu: „Privacy-Preserving Data Publishing: A Survey of Recent Developments“. *ACM Comput. Surv.* (Juni 2010), Bd. 42(4) (siehe S. 2).

[Glu18] Gluchowski, Peter: „Data Governance - Betriebliche Daten als Wirtschaftsgüter verstehen und behandeln“. *TDWI E-Book* (2018), Bd. (siehe S. 3).

[Han14] Hance, Travis J: „Jelf: a web framework for automatic privacy policy enforcement“. Diss. Massachusetts Institute of Technology, 2014 (siehe S. 8).

[Hev10] Hevner, Alan und Samir Chatterjee: „Design science research in information systems“. *Design research in information systems*. Springer, 2010: S. 9–22 (siehe S. 25).

[Hev04] Hevner, Alan, Salvatore T March, Jinsoo Park, Sudha Ram u. a.: „Design science research in information systems“. *MIS quarterly* (2004), Bd. 28(1): S. 75–105 (siehe S. 25).

[Hua14] Huang, Qinlong, Zhaofeng Ma, Yixian Yang, Xinxin Niu und Jingyi Fu: „Attribute based DRM scheme with dynamic usage control in cloud computing“. *China Communications* (2014), Bd. 11(4): S. 50–63 (siehe S. 18).

[Jar19] Jarke, Matthias and Otto, Boris and Ram, Sudha: „Data Sovereignty and Data Space Ecosystems“. *Business & Information Systems Engineering* (2019), Bd. 61(5): S. 549–550 (siehe S. 1).

[Jea01] Jeager, Trent: „Managing Access Control Complexity Using Metrices“. *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*. SACMAT '01. Chantilly, Virginia, USA: Association for Computing Machinery, 2001: S. 131–139 (siehe S. 11).

[Jun15] Jung, Christian und Ralf Kalmar: „New Interpretation of Data Security Treasure Trove of Date and Business Models“. *ATZelektronik worldwide* (2015), Bd. 10(4): S. 14–19 (siehe S. 15).

[Jür02a] Jürjens, Jan: „Principles for secure systems design“. Diss. University of Oxford, 2002 (siehe S. 23).

[Jür02b] Jürjens, Jan: „UMLsec: Extending UML for secure systems development“. *International Conference on The Unified Modeling Language*. Springer. 2002: S. 412–425 (siehe S. 23).

[Kas15] Kasem-Madani, Saffija und Michael Meier: „Security and privacy policy languages: a survey, categorization and gap identification“. *arXiv preprint arXiv:1512.00201* (2015), Bd. (siehe S. 10).

[Kes03] Keselev, Ivan: *Aspect-oriented programming with AspectJ*. Sams, 2003 (siehe S. 24).

[Kom] Kommision, Europäische: *Eine europäische Datenstrategie (COM/2020/66)*. Zugriff am 23. Januar 2021, von https://eur-lex.europa.eu/legal-content/DE/ALL/?uri=COM:2020:66:FIN (siehe S. 2).

[Kou12] Kourie, Derrick G und Bruce W Watson: *The Correctness-by-Construction Approach to Programming*. Springer Science & Business Media, 2012 (siehe S. 23).

[Kra00] Kramer, Thomas, Frederick Proctor und Elena Messina: *The NIST RS274NGC Interpreter - Version 3*. en. 2000-08-01 2000 (siehe S. 79).

[Kwo02] Kwok, Sai Ho: „Digital rights management for the online music business“. *ACM Sigecom exchanges* (2002), Bd. 3(3): S. 17–24 (siehe S. 12).

[La 17a] La Marra, Antonio, Fabio Martinelli, Paolo Mori, Athanasios Rizos und Andrea Saracino: „Improving MQTT by inclusion of usage control“. *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer. 2017: S. 545–560 (siehe S. 18).

[La 17b] La Marra, Antonio, Fabio Martinelli, Paolo Mori und Andrea Saracino: „Implementing usage control in internet of things: a smart home use case“. *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE. 2017: S. 1056–1063 (siehe S. 18).

[Laz12] Lazouski, Aliaksandr, Gaetano Mancini, Fabio Martinelli und Paolo Mori: „Usage control in cloud systems“. *2012 International Conference for Internet Technology and Secured Transactions*. IEEE. 2012: S. 202–207 (siehe S. 18).

[Laz08] Lazouski, Aliaksandr, Fabio Martinelli und Paolo Mori: „A survey of usage control in computer security“. *Istituto di Informática e Telemática, CNR* (2008), Bd. (siehe S. 10).

[Laz10] Lazouski, Aliaksandr, Fabio Martinelli und Paolo Mori: „Usage control in computer security: A survey“. *Computer Science Review* (2010), Bd. 4(2): S. 81–99 (siehe S. 4, 9, 10).

[Lee14] Lee, Jay, Hung-An Kao, Shanhu Yang u. a.: „Service Innovation and Smart Analytics for Industry 4.0 and Big Data Environment“. *Procedia Cirp* (2014), Bd. 16(1): S. 3–8 (siehe S. 1).

[Li15] Li, Chao, Daniel Yang Li, Gerome Miklau und Dan Suciu: „A Theory of Pricing Private Data“. *ACM Trans. Database Syst.* (Dez. 2015), Bd. 39(4) (siehe S. 1).

[Lin06] Lindqvist, Hakan: „Mandatory access control“. *Master's thesis in computing science, Umea University, Department of Computing Science, SE-901* (2006), Bd. 87 (siehe S. 9).

[Liu03] Liu, Qiong, Reihaneh Safavi-Naini und Nicholas Paul Sheppard: „Digital rights management for content distribution“. *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003-Volume 21*. Citeseer. 2003: S. 49–58 (siehe S. 12).

[Men06] Mens, Tom und Pieter Van Gorp: „A taxonomy of model transformation“. *Electronic notes in theoretical computer science* (2006), Bd. 152: S. 125–142 (siehe S. 22).

[Mer] Merkel, Angela: *Rede von Bundeskanzlerin Merkel beim Digital-Gipfel am 29. Oktober 2019 in Dortmund)*. Zugriff am 23. Januar 2021, von https://www.bundesregierung.de/breg-de/suche/rede-von-bundeskanzlerin-merkel-beim-digital-gipfel-am-29-oktober-2019-in-dortmund-1686444 (siehe S. 2).

[Mic20] Micinski, Kristopher, David Darais und Thomas Gilray: „Abstracting Faceted Execution“. *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. IEEE. 2020: S. 184–198 (siehe S. 8).

[Mic18] Micinski, Kristopher, Zhanpeng Wang und Thomas Gilray: „Racets: Faceted execution in racket“. *arXiv preprint arXiv:1807.09377* (2018), Bd. (siehe S. 8).

[Mil15] Milicevic, Aleksandar u. a.: „Advancing declarative programming“. Diss. Massachusetts Institute of Technology, 2015 (siehe S. 8).

[Mon14] Monostori, László: „Cyber-physical Production Systems: Roots, Expectations and R&D Challenges“. *Procedia CIRP* (2014), Bd. 17. Variety Management in Manufacturing: S. 9–13 (siehe S. 1).

[Mül21] Müller-Quade, Jörn, Roger Gutbrod, Volkmar Lotz, Fabian Biegel, Benny Fuhry und Jeremias Mechler: „SES-14: Sichere und zuverlässliche Systeme: Datensouveränität“. *INFORMATIK 2020* (2021), Bd. (siehe S. 4).

[Nil08] Nilchi, AR Naghsh, A Vafaei und H Hamidi: „Evaluation of security and fault tolerance in mobile agents“. *2008 5th IFIP International Conference on Wireless and Optical Communications Networks (WOCN'08)*. IEEE. 2008: S. 1–5 (siehe S. 20).

[Nyr11] Nyre, Åsmund Ahlmann: „Usage Control Enforcement - A Survey“. *Availability, Reliability and Security for Business, Enterprise and Health Information Systems*. Hrsg. von Tjoa, A. Min, Gerald Quirchmayr, Ilsun You und Lida Xu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011: S. 38–49 (siehe S. 4, 10).

[Ott17] Otto, B. u. a.: „Reference Architecture Model for the Industrial Data Space“. https://www.internationaldataspaces.org/wp-content/uploads/dlm_uploads/2019/05/Industrial-Data-Space_Reference-Architecture-Model-2017.pdf, zugegriffen am 22.01.2021. 2017 (siehe S. 26).

[Ott19a] Otto, B. u. a.: „Reference Architecture Model Version 3.0“. https://www.internationaldataspaces.org/wp-content/uploads/2019/03/IDS-Reference-Architecture-Model-3.0.pdf, zugegriffen am 22.01.2021. Apr. 2019 (siehe S. 4, 18, 26).

[Ott19b] Otto, Boris: „Interview with Reinhold Achatz on 'Data Sovereignty and Data Ecosystems'“. *Business & Information Systems Engineering* (2019), Bd. 61(5): S. 635–636 (siehe S. 4).

[Ott18] Otto, Boris, Michael ten Hompel und Stefan Wrobel: „Industrial Data Space“. *Digitalisierung: Schlüsseltechnologien für Wirtschaft und Gesellschaft*. Hrsg. von Neugebauer, Reimund. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018: S. 113–133 (siehe S. 4).

[Ott16] Otto, Boris, J Jürjens, J Schon, S Auer, N Menz, S Wenzel und J Cirullies: „Industrial Data Space White Paper“. *Fraunhofer-Gesellschaft: Munich, Germany* (2016), Bd. (siehe S. 4).

[Pal17] Palesha, Kushal: „Policy-agnostic programming on the client-side“. (2017), Bd. (siehe S. 8).

[Par04] Park, Jaehong und Ravi Sandhu: „The UCON_{ABC} Usage Control Model“. *ACM Trans. Inf. Syst. Secur.* (Feb. 2004), Bd. 7(1): S. 128–174 (siehe S. 3, 9, 10).

[Par02] Park, Jaehong und Ravi Sandhu: „Towards Usage Control Models: Beyond Traditional Access Control“. *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*. SACMAT '02. Monterey, California, USA: Association for Computing Machinery, 2002: S. 57–64 (siehe S. 3, 9).

[Pas15] Paschalidi, Charikleia: „Data Governance: A Conceptual Framework in Order to Prevent Your Data Lake From Becoming a Data Swamp“. Validerat; 20151222 (global_studentproject_submitter). Diss. 2015 (siehe S. 3).

[Pas20] Pasi, Bhaveshkumar N, Subhash K Mahajan und Santosh B Rane: „Smart supply chain management: a perspective of industry 4.0“. *Supply Chain Management* (2020), Bd. 29(5): S. 3016–3030 (siehe S. 1).

[Pic09] Pickering, Robert: „Language-oriented programming“. *Beginning F#*. Springer, 2009: S. 327–349 (siehe S. 21).

[Pol16] Polikarpova, Nadia, Jean Yang, Shachar Itzhaky und Armando Solar-Lezama: „Type-Driven Repair for Information Flow Security“. *CoRR* (2016), Bd. abs/1607.03445 (siehe S. 9).

[Pre08] Pretschner, Alexander, Manuel Hilty, Florian Schütz, Christian Schaefer und Thomas Walter: „Usage control enforcement: Present and future“. *IEEE Security & Privacy* (2008), Bd. 6(4): S. 44–53 (siehe S. 10).

[Raj09] Rajkumar, P. V., S. K. Ghosh und Pallab Dasgupta: „Application specific usage control implementation verification“. *International Journal of Network Security and Its Applications* (2009), Bd. 1(3): S. 116–128 (siehe S. 11).

[Ric19] Richter, Heiko und Peter R. Slowinski: „The Data Sharing Economy: On the Emergence of New Intermediaries“. *IIC-International Review of Intellectual Property and Competition Law* (2019), Bd. 50(1): S. 4–29 (siehe S. 2).

[Rit17] Riti, Pierluigi: *Practical Scala DSLs: Real-World Applications Using Domain Specific Languages*. Apress, 2017 (siehe S. 21).

[Rod18] Rodriguez, Maria Angeles, Llanos Cuenca und Angel Ortiz: „FIWARE Open Source Standard Platform in Smart Farming-a Review“. *Working Conference on Virtual Enterprises*. Springer. 2018: S. 581–589 (siehe S. 4).

[Rub98] Rubin, Aviel D und Daniel E Geer: „Mobile code security“. *IEEE Internet Computing* (1998), Bd. 2(6): S. 30–34 (siehe S. 20).

[Sah05] Sahai, Amit und Brent Waters: „Fuzzy identity-based encryption“. *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2005: S. 457–473 (siehe S. 18).

[Sam00] Samarati, Pierangela und Sabrina Capitani de Vimercati: „Access control: Policies, models, and mechanisms“. *International School on Foundations of Security Analysis and Design*. Springer. 2000: S. 137–196 (siehe S. 9).

[San98a] Sandhu, Ravi und Qamar Munawer: „How to do discretionary access control using roles“. *Proceedings of the third ACM workshop on Role-based access control*. 1998: S. 47–54 (siehe S. 9).

[San03] Sandhu, Ravi und Jaehong Park: „Usage Control: A Vision for Next Generation Access Control“. *Computer Network Security*. Hrsg. von Gorodetsky, Vladimir, Leonard Popyack und Victor Skormin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003: S. 17–31 (siehe S. 3, 9).

[San98b] Sandhu, Ravi S: „Role-based access control“. *Advances in computers*. Bd. 46. Elsevier, 1998: S. 237–286 (siehe S. 9).

[Sch18] Schütte, Julian und Gerd Stefan Brost: „LUCON: Data flow control for message-based IoT systems“. *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/-BigDataSE)*. IEEE. 2018: S. 289–299 (siehe S. 18).

[Sha21] Shao, Xue-Feng, Wei Liu, Yi Li, Hassan Rauf Chaudhry und Xiao-Guang Yue: „Multistage implementation framework for smart supply chain management under industry 4.0“. *Technological Forecasting and Social Change* (2021), Bd. 162: S. 120354 (siehe S. 1).

[Sta90a] Stamos, James W und David K Gifford: „Remote evaluation“. *ACM Transactions on Programming Languages and Systems (TOPLAS)* (1990), Bd. 12(4): S. 537–564 (siehe S. 19).

[Sta90b] Stamos, James W. und David K. Gifford: „Implementing remote evaluation“. *IEEE Transactions on Software Engineering* (1990), Bd. 16(7): S. 710–722 (siehe S. 19).

[Sta13] Standard, OASIS: *extensible access control markup language (xacml) version 3.0*. 2013 (siehe S. 15).

[Ste] Stefanelli, Cesare, Niranjan Suri und Matteo Rebeschini: „MAST: A Mobile Agent Environment for Securing Hosts and Networks“. (), Bd. (siehe S. 20).

[Ste02] Stefansson, Gunnar: „Business-to-Business Data Sharing: A Source for Integration of Supply Chains“. *International Journal of Production Economics* (2002), Bd. 75(1). Information Technology/Information Systems in 21st Century Production: S. 135–146 (siehe S. 2).

[Ste19] Steffen, Bernhard, Frederik Gossen, Stefan Naujokat und Tiziana Margaria: „Language-driven engineering: from general-purpose to purpose-specific languages“. *Computing and Software Science*. Springer, 2019: S. 311–344 (siehe S. 22).

[Ste16] Steinebach, Martin, Erik Krempel, Christian Jung und Mario Hoffmann: „Datenschutz und Datenanalyse“. *Datenschutz und Datensicherheit-DuD* (2016), Bd. 40(7): S. 440–445 (siehe S. 15).

[Sub06] Subramanya, SR und Byung K Yi: „Digital rights management“. *IEEE potentials* (2006), Bd. 25(2): S. 31–34 (siehe S. 12).

[Sur00] Suri, Niranjan, Jeffrey M Bradshaw, Maggie R Breedy, PT Groth, Gregory A Hill, R Jeffers, TR Mitrovich, BR Pouliot und DS Smith: „NOMADS: Toward an environment for strong and safe agent mobility“. *Proceedings of Autonomous Agents 2000*. Bd. 10. 336595.337103. Barcelona, Spain. 2000 (siehe S. 20).

[Tav12] Tavizi, Tina, Mehdi Shajari und Peyman Dodangeh: „A usage control based architecture for cloud environments“. *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE. 2012: S. 1534–1539 (siehe S. 18).

[Tja17] Tjahjono, Benny, C Esplugues, Enrique Ares und G Pelaez: „What does industry 4.0 mean to supply chain?“ *Procedia manufacturing* (2017), Bd. 13: S. 1175–1182 (siehe S. 1).

[Van00] Van Deursen, Arie, Paul Klint und Joost Visser: „Domain-specific languages: An annotated bibliography“. *ACM Sigplan Notices* (2000), Bd. 35(6): S. 26–36 (siehe S. 21).

[Wan16] Wang, K: „Intelligent predictive maintenance (IPdM) system–Industry 4.0 scenario“. *WIT Transactions on Engineering Sciences* (2016), Bd. 113: S. 259–268 (siehe S. 1).

[Wu15] Wu, Jun, Mianxiong Dong, Kaoru Ota, Muhammad Tariq und Longhua Guo: „Cross-Domain Fine-Grained Data Usage Control Service for Industrial Wireless Sensor Networks“. *IEEE Access* (Dez. 2015), Bd. 3: S. 1–1 (siehe S. 3).

[Wüc12] Wüchner, Tobias und Alexander Pretschner: „Data loss prevention based on data-driven usage control“. *2012 IEEE 23rd International Symposium on Software Reliability Engineering*. IEEE. 2012: S. 151–160 (siehe S. 16).

[Yan17] Yan, Jihong, Yue Meng, Lei Lu und Lin Li: „Industrial big data in an industry 4.0 environment: Challenges, schemes, and applications for predictive maintenance“. *IEEE Access* (2017), Bd. 5: S. 23484–23491 (siehe S. 1).

[Yan15a] Yang, Jean: „Preventing information leaks with policy-agnostic programming“. Dissertation. Massachusett: Massachusetts Institute of Technology, 2015 (siehe S. 7, 104).

[Yan15b] Yang, Jean, Travis Hance, Thomas H Austin, Armando Solar-Lezama, Cormac Flanagan und Stephen Chong: „End-to-end policy-agnostic security for database-backed applications“. *CoRR, abs/1507.03513* (2015), Bd. (siehe S. 7, 8).

[Yan12] Yang, Jean, Kuat Yessenov und Armando Solar-Lezama: „A language for automatically enforcing privacy policies“. *ACM SIGPLAN Notices* (2012), Bd. 47(1): S. 85–96 (siehe S. 7).

[Zha05] Zhang, Xinwen, Francesco Parisi-Presicce, Ravi Sandhu und Jaehong Park: „Formal model and policy specification of usage control“. *ACM Transactions on Information and System Security (TISSEC)* (2005), Bd. 8(4): S. 351–387 (siehe S. 3).

[Zol16] Zolnowski, Andreas, Towe Christiansen und Jan Gudat: „Business Model Transformation Patterns of Data-Driven Innovations“. *Research Papers* (2016), Bd. 146. https://aisel.aisnet.org/ecis2016_rp/146/ (siehe S. 1).

[Zon20] Zonta, Tiago, Cristiano André da Costa, Rodrigo da Rosa Righi, Miromar José de Lima, Eduardo Silveira da Trindade und Guann Pyng Li: „Predictive maintenance in the Industry 4.0: A systematic literature review“. *Computers & Industrial Engineering* (2020), Bd.: S. 106889 (siehe S. 1).

Abbildungsverzeichnis

Veröffentlichungen

Dieses Kapitel enthält eine Auflistung der Veröffentlichungen, an welchen der Autor des vorliegenden Dokuments beteiligt war. Dabei findet eine Unterteilung in die folgenden Kategorien statt:

- Wissenschaftliche Veröffentlichungen, welche die Kernthemen des vorliegenden Dokuments behandeln
- Sonstige Veröffentlichungen, beispielsweise technische Berichte

Bei allen Veröffentlichungen, die in den Kernbereich der Dissertation fallen, war der Autor der Dissertation der Erstautor und hat die jeweiligen Projekte geleitet. Darüber hinaus wurde die Ideenfindung stark durch ihn geprägt und die Beiträge hauptsächlich durch ihn verfasst. Außerdem hat er die Einreichungen vorbereitet und den Veröffentlichungsprozess begleitet.

Kernbereich der Dissertation

1. **Bruckner, Fabian**, Ralf Nagel, Dominik Krüger, Sven Wenzel und Boris Otto: „Eine Programmiersprache zur souveränen Datenverarbeitung“. *D · A · CH SECURITY 2018, Gelsenkirchen, Germany, September 4-5 2018*. Hrsg. von Horster, Patrick und Norbert Pohlmann. Zugriff über https://www.syssec.at/de/veranstaltungen/dachsecurity2018/papers/DACH_Security_2018_Paper_12A1.pdf. syssec, 2018: S. 35–46.
2. **Bruckner, Fabian**, Julia Pampus und Falk Howar: „A Framework for Creating Policy-agnostic Programming Languages“. *Proceedings of the 9th International Conference on Data Science, Technology and Applications, DATA 2020, Lieusaint, Paris, France, July 7-9, 2020*. Hrsg. von Hammoudi, Slimane, Christoph Quix und Jorge Bernardino. Zugriff über https://www.scitepress.org/Papers/2020/97822/97822.pdf. SciTePress, 2020: S. 31–42.
3. **Bruckner, Fabian** und Falk Howar: „Utilizing Remote Evaluation for Providing Data Sovereignty in Data-sharing Ecosystems“. *54th Hawaii International Conference on System Sciences, HICSS 2021, Kauai, Hawaii, USA, January 5, 2021*. Zugriff über https://scholarspace.manoa.hawaii.edu/handle/10125/71463. ScholarSpace, 2021: S. 1–10.
4. **Bruckner, Fabian**, Julia Pampus und Falk Howar: „A Policy-Agnostic Programming Language for the International Data Spaces“. *Data Management Technologies and Applications*. Hrsg. von Hammoudi, Slimane, Christoph Quix und Jorge Bernardino. Zugriff über https://link.springer.com/chapter/10.1007/978-3-030-83014-4_9. Cham: Springer International Publishing, 2021: S. 172–194.

Technische Berichte und andere Veröffentlichungen

1. Eitel, Andreas, Christian Jung, Christian Kühnle, **Fabian Bruckner**, Gerd Brost, Pascal Birnstill, Ralf Nagel und Sebastian Bader: „Usage Control in the International Data Spaces: Version 2.0“. (2019), Bd. Zugriff am 24. Januar 2021, von https://www.internationaldataspaces.org/wp-content/uploads/2020/09/IDSA-Position-Paper-Usage-Control-in-IDS.pdf.
2. Bader, Sebastian, **Fabian Bruckner**, Gernot Böge, Dennis Oliver Kubitza, Jörg Langkau, Dileep Murthy und Ralf Nagel: „Specification: IDS Meta Data Broker“. https://internationaldataspaces.org/download/16337/, zugegriffen am 12.03.2021. Mai 2020.
3. **Bruckner, Fabian** und Nils Jahnke: *Datenschutz und Datensicherheit in Datenökosystemen - IIP-Ecosphere Whitepaper*. Version 1.0. https://doi.org/10.5281/zenodo.4588330, zugegriffen am 12.03.2021. März 2021.
4. Eitel, Andreas, Christian Jung, Robin Brandstädter, Arghavan Hosseinzadeh, Sebastian Bader, Christian Kühnle, Pascal Birnstill, Gerd Brost, Mark Gall, **Fabian Bruckner**, Norbert Weissenberg und Benjamin Korth: „Usage Control in the International Data Spaces: Version 3.0“. (2021), Bd. Zugriff am 08. April 2021, von https://internationaldataspaces.org/download/21053/.

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich während der Bearbeitung meiner Dissertation unterstützt und begleitet haben. Ohne diese Personen hätte die vorliegende Arbeit nicht in dieser Form realisiert werden können.

Mein herzlichster Dank gilt zunächst meinem Doktorvater Prof. Dr. Falk Howar, der mich im Verlaufe der Dissertation stets intensiv unterstützt und immer wieder mit Anregungen versorgt hat. Ich danke ebenso Prof. Dr. Jan Jürjens für die Betreuung als Zweitgutachter.

Mein besonderer Dank gilt Dr. Ralf Nagel, welcher mir in unzähligen Diskussionen zur Verfügung stand und mir stets mit konstruktiven Anregungen und lieben Worten zur Seite stand. Ebenso danke ich Julia Pampus, die durch ihr hervorragendes Lektorat zum erfolgreichen Abschluss dieser Arbeit beigetragen hat.

Ebenfalls möchte ich meiner Verlobten Sade Mangold danken, die mich unermüdlich mit Geduld und Zuspruch unterstützte. Schlussendlich möchte ich neben vielen Freunden meiner Schwester Katrin Klimmek und meinem Schwager Mirko Klimmek dafür danken, dass sie mir zu diversen Zeitpunkten die notwendige Zerstreuung bescherten.