Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-1030 Wien, Seidlgasse 22 / 9
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

A-8010 Graz, Inffeldgasse 16a
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461
ZVR: 948166612
UID: ATU60778947

ORCHESTRATION VON VERTEILTEN ANWENDUNGEN

Projektbericht

Version 1.0, 10.11.2016

Bojan Suzic – bojan.suzic@a-sit.at

Zusammenfassung:

Web-APIs spielen eine wichtige Rolle bei der Integration von Daten und Diensten. Viele Web-Dienste bieten heutzutage ein gesondertes Interface in der Form eines Web-APIs an, das den Austausch von Daten und die Verwendung von zur Verfügung gestellten Funktionalitäten ermöglicht. Jedoch ist das Management von Sicherheitsaspekten dieser Schnittstellen komplex und oft undurchsichtig. Aus der Rolle von Web-APIs, die als ein Grundstein und Treiber des modernen Internet und domänenübergreifender Transaktionen gelten, ist es notwendig, die Modellierung und folglich das Sicherheitsmanagement dieser Prozesse weiter zu entwickeln.

A-SIT hat bereits verschiedene Autorisierungsmodelle erforscht und Bibliotheken von Cloud-Anwendungen analysiert[1], was zur Entwicklung eines multifunktionellen Frameworks führte[2]. Im Zuge dieses Projekts wurde ein Konzept erarbeitet und praktisch umgesetzt, das maschinenlesbares Strukturieren von verschiedenen Web-APIs ermöglicht. Diese Strukturierung wird zuerst für das Sicherheitsmanagement und dessen Durchsetzung angewandt. In dieser Iteration wurden zuerst 20 APIs aus 7 unterschiedlichen Produktkategorien analysiert. Diese APIs werden von populären Diensten zur Verfügung gestellt und oft im Rahmen von Integrationsprozessen als Dienste oder Datenquellen in weiteren Workflows ausgeführt.

Basierend auf den in der ersten Iteration gewonnenen Erkenntnissen haben wir ein konzeptuelles Meta-Modell entwickelt, das die Strukturierung von APIs in Kategorien ermöglicht. Dieses Konzept ermöglicht somit eine breitere und abstraktere Anwendung, als es in Vorgängerprojekten realisiert wurde. Dieses Konzept wurde im Zuge des Projekts praktisch realisiert und gegen Case-Szenarien bei ausgewählten Anwendungen getestet. Als Ergebnis dieser Aktivitäten wurde das bestehende Softwareframework entsprechend erweitert, um die neue Konzeptualisierung unterstützen zu können.

---

[1] Sichere Integration in der Cloud - https://demo.a-sit.at/sichere-integration-in-der-cloud/

[2] Multidimensionale Informationssicherheitsrichtlinien - https://demo.a-sit.at/multidimensional-security-policies/

Inhaltsverzeichnis

Abbildungen

Tabellen

1. Einleitung

Datengesteuerte sowie generell die digitale Wirtschaft, sind für die heutige Gesellschaft von wesentlicher Bedeutung. Digitale Daten sind heutzutage Grundlage nahezu aller Bereichen der Wirtschaft oder der öffentlichen Verwaltung. Diese Daten werden in fast allen Bereichen des wirtschaftlichen, gesellschaftlichen oder persönlichen Lebens generiert, erfasst und verarbeitet. In den letzten Jahren sind viele innovative Unternehmen entstanden, deren Hauptgeschäfts-gegenstand digitale Daten sind.

Für Verwaltung und Austausch von digitalen Daten spielen Web APIs eine wichtige Rolle. Diese APIs ermöglichen strukturierte Verwendung von verschiedenen Funktionalitäten, sowie die Abfrage, Speicherung oder Austausch von Daten zwischen vielen Akteuren.

Im Zuge dieser Arbeit wurde eine Gruppe von repräsentativen Web-APIs ausgewählt und analysiert. Das Ziel war das Erkennen und der Vergleich von verschiedenen Implementationen sowie Ansätzen, die die Unternehmen für die Bereitstellung von APIs sowie zur Integration von Daten verwenden. Basierend auf dieser ersten Iteration der Analyse wurde ein Konzept entwickelt, das die abstrakte Repräsentation von verschiedenen APIs ermöglicht.

Hauptmotivation dieser Arbeit war die Weiterentwicklung von Konzepten und Tools des domänen-übergreifenden Sicherheitsmanagements, die schon in Rahmen von Vorgängerprojekten entwickelt wurden. Basierend auf dieser Arbeit wurde ein Softwareframework weiterentwickelt und erweitert.

2. API Modelle und Cloud-Anbieter

Im Rahmen dieser Arbeit wurde eine Auswahl an repräsentativen Cloud-Anbietern und deren Diensten ausgewählt und analysiert.

In der Auswahl waren zwei Kriterien entscheidend:

- Die Anwendung soll bei etablierten Cloud-Integrationsdiensten, wie Zapier[3], Elastic.IO[4] oder IFTTT[5] unterstützt sein

- Die Anwendung soll populären Kategorien angehören, wie: Storage, Email, Calendar, CRM, Spreadsheets, Social und eCommerce. Diese sind populäre Anwendungsgruppen und repräsentieren unterschiedliche Business- und Datenverarbeitungsmodelle.

2.1. Liste von APIs in der Cloud

Zum Zeitpunkt der Berichterstellung unterstützte Zapier mehr als 500 Dienste. Darüber hinaus unterstützte IFTTT mehr als 700, in 42 Kategorien klassifizierte, Dienste.

Tabelle 1 enthält die Liste von analysierten Diensten.

Name	Kategorie	API-Dokumentation
Google Drive	Storage	https://developers.google.com/drive/v3/reference
Box	Storage	https://docs.box.com/reference

[3] https://zapier.com/

[4] https://www.elastic.io/

[5] https://ifttt.com/discover

Dropbox	Storage	https://www.dropbox.com/developers/documentation/http/overview
OneDrive	Storage	https://dev.onedrive.com/readme.htm
Google Calendar	Calendar	https://developers.google.com/google-apps/calendar/v3/reference/
Outlook Calendar	Calendar	https://msdn.microsoft.com/office/office365/api/calendar-rest-operations?f=255&MSPPError=-2147217396
Outlook Mail	Email	https://msdn.microsoft.com/en-us/office/office365/api/mail-rest-operations
Gmail	Email	https://developers.google.com/gmail/api/v1/reference/
Zoho	Email	https://www.zoho.com/mail/help/api/
Google Sheets	Spreadsheets	https://developers.google.com/sheets/reference/rest/
Office365 Excel	Spreadsheets	https://graph.microsoft.io/en-us/docs/api-reference/v1.0/resources/excel
SalesForce	CRM	https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_list.htm
Microsoft Dynamics	CRM	https://msdn.microsoft.com/en-us/library/mt607689.aspx?f=255&MSPPError=-2147217396
Magento	eCommerce	http://devdocs.magento.com/guides/v2.0/rest/bk-rest.html
Amazon Marketplace WS	eCommerce	http://docs.developer.amazonservices.com/en_US/feeds/index.html
Shopify	eCommerce	https://help.shopify.com/api/reference
PayPal	eCommerce	https://developer.paypal.com/docs/api/
Facebook	Social	https://developers.facebook.com/docs/graph-api/reference
Twitter	Social	https://developers.google.com/drive/v3/reference
Linkedin	Social	https://docs.box.com/reference

Tabelle 1: Analysierte Dienste

Jede dieser Anwendungen hat ein eigenes API, wobei sich die Datenmodelle, API-Organisation sowie unterstützte Funktionalitäten wesentlich unterscheiden. Das Ziel dieser Analyse war daher, ein passendes Meta-Model zu erzeugen, das die verschiedene Funktionalitäten und Ansätze möglichst nah unterstützt.

2.2. Meta-Modelle für API-Ressourcen

Um die Organisation einer beliebigen API beschreiben zu können, wurde zuerst das Meta-Model für die Darstellung der APIs definiert. Das Model ermöglicht eine abstrakte Definition, die bei der Modellierung von diversen APIs zur Anwendung kommt. Dieses Modell, sowie das Vorgehen, basieren auf dem DASP-Framework [1, 2]. Es wurden hier die vorhandenen *Vocabularies* entsprechend angepasst, um neuen Anforderungen bezüglich der Diversität von APIs zu genügen.

Drei Konzepte sind für diese Beschreibung relevant. Die *Ressource* repräsentiert eine abstrakte Entität, die direkt über API-Aufrufe zugreifbar oder modifizierbar ist. In der Regel sind die Entitäten direkt, per eindeutiger URL, durch die API exponiert. In manchen Fällen sind die Entitäten auch teilweise über Sammelpunkte abrufbar. Diese Variante entspricht einem Endpunkt, der allen zu einer Kategorie zuweisende Entitäten zusammengestellt.

Abbildung 1 zeigt das Meta-Model der abstrakten Entität. Jede Entität unterstützt verschiedene Zugriffsmethoden, wie z.B. HTTP GET, PUT oder POST. Weiters wird jede Entität durch einen entsprechenden Endpunkt exponiert, die mittels von *urlPath*-Relationen sequenziell definiert ist. Dieser Endpunkt wird als Teil der URL für die API-Aufrufe verwendet.

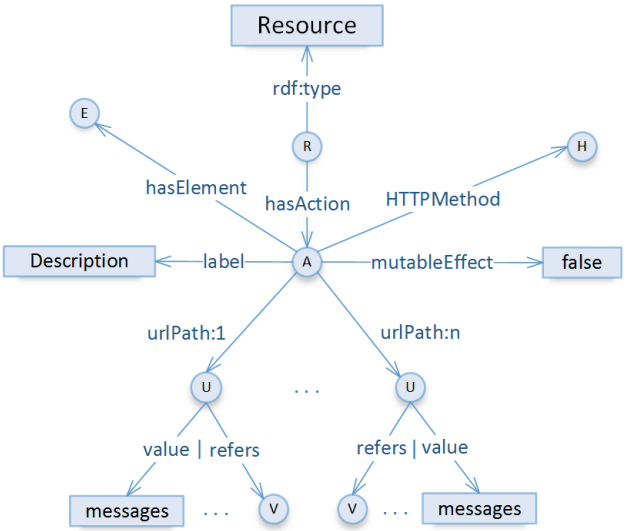

Abbildung 1: Meta-Model für Resource

Jede Entität (oder Ressource) wird als ein Datensatz gespeichert und für weitere Verarbeitung zur Verfügung gestellt. Ein solcher Datensatz beinhaltet oft verschiedene Daten, wird aber seitens der Cloud-Anbieter selten in strukturierter Form dargestellt. Dies erschwert das Sicherheits-management von Datenbeständen, da die Daten nur grob abrufbar sind. Zusätzlich dazu sind die semantischen Eigenschaften von Daten unbekannt, was die Kategorisierung hinsichtlich Sicherheit und Datenschutz weiter kompliziert. Darüber hinaus wird auch die Interoperabilität von Daten und Prozessen allgemein durch die fehlenden semantischen Eigenschaften beeinflusst. Das betrifft insbesondere die komplexen und dynamischen Datenbestände,

Um die Datenbestände strukturiert darstellen und verarbeiten zu können, wird die Beschreibung von Ressourcen auf semantischer und struktureller Ebene integriert. Dies führt eine weitere Granularitätsebene bei der Beschreibung von Ressourcen ein. Ein *Element* stellt somit ein erkennbares und konstituierendes Teil des Datenbestandes dar, das bei weiterer Verarbeitung oder Austausch gesondert berücksichtigt werden kann. Somit sind die Transparenz sowie fortgeschrittene Funktionalitäten des Sicherheitsmanagements bei dem Datenaustausch oder bei der API-Verwendung gestattet. Basierend auf das granulare und semantisch gesteuerte Darstellen werden auch die Verarbeitung und Umwandlung von Ressourcen als kontextbezogene und adaptive Vorgänge unterstützt.

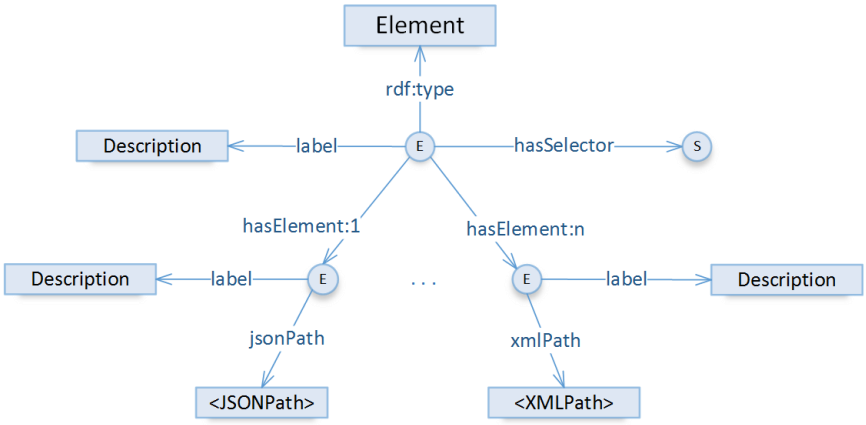

Abbildung 2: Meta-Model für Element

Abbildung 2 zeigt das Meta-Modell für das *Element*. Ein Element ist durch eine beliebige Anzahl von weiteren Elementen oder Instruktionen darstellbar. Instruktionen werden bei der Datenabfrage angewandt, um ein Element zu extrahieren. Verschiedene Ansätze für die Abfrage von Daten sind möglich. Die aktuelle Version geht davon aus, dass die Methoden wie JSONPath oder XMLPath, abhängig von der Darstellung der Ressource, angewandt werden.

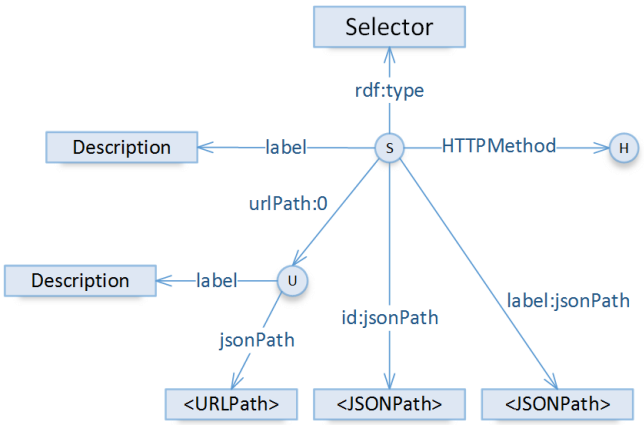

Abbildung 3: Meta-Model für Selector

Für ein granuläres Sicherheitsmanagement ist es oft notwendig, die unterstützten oder vorhandenen Entitäten sowie deren Eigenschaften dynamisch abfragen zu können. Dafür ist ein Selektor vorgesehen, der die Möglichkeiten von strukturierten Abfragen von Elementen dynamisch und programmatisch ermöglicht. Jedes Element kann daher einen oder mehrere Selektoren referenzieren.

In aktueller Version des Frameworks, ein Selektor unterstützt zwei Variablen: *id*, die für die Abfrage der Ressource verwendet wird, und *label*, die für die Beschreibung von der Ressource dient. Die Integration von weiteren Optionen wird bei zukünftiger Weiterentwicklung berücksichtigt.

2.3. Anwendungsszenario PayPal

Um die praktische Anwendung von Meta-Modellen näher zu demonstrieren, wird in diesem Abschnitt das Beispiel einer Integration mit PayPal beschrieben. PayPal ist ein eCommerce Dienst, der verschiedene Funktionen als Zahlungsabwickler und Treuhänder übernimmt.

Dieses Beispiel konzentriert sich auf ein bestimmtes Szenario, das die Integration von zwei Endpunkten vorsieht.

Zuerst betrachten wir den folgenden Endpunkt:

```
GET /v1/payments/payment
```

Dieser Endpunkt stellt eine Liste von geleisteten Zahlungen zur Verfügung, wobei jede Entität auch durch getrennte und eindeutige URLs abrufbar ist. Daher wird dieser Endpunkt auch als Sammelendpunkt bezeichnet, da der eine Liste oder Ansammlung von schon existierenden Entitäten exponiert.

Das in Abbildung 4 dargestellte Schema zeigt das Modell des PayPal-Payments Endpunkts. Die Knoten werden in der Abbildung durch die Kreise dargestellt, die Vierecke dienen zur Bezeichnung der Werte. Basierend auf der Darstellung in der Abbildung können wir den entsprechenden Endpunkt des APIs ableiten. In diesem Beispiel entspricht HTTPMethod *h1* der *HTTP GET* Anfragemethode. Zusätzlich dazu ist auch die Repräsentation der abfragenden Entität durch den Knoten *e1* vorgesehen[6].

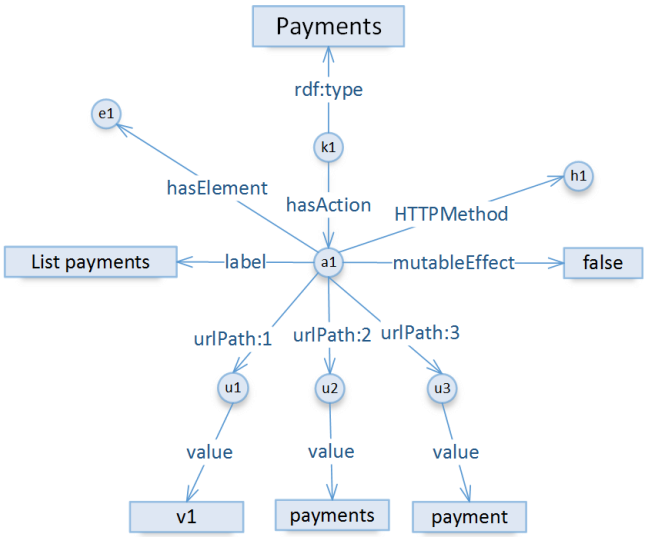

Abbildung 4: Model für eine Transaktionsliste (PayPal)

Ebenso ist der Endpunkt für die Abfrage einzelnen Zahlungstransaktionen wie folgt:

```
GET /v1/payments/payment/<id>
```

In diesem Fall ist die URL-Adresse vom Identifikator[7] der Entität bzw. der Transaktion abhängig. Dieses Element soll daher auch als eine dynamische Referenz betrachtet werden, die jeweils für jede Instanz von Ressourcen spezifisch ist. Dementsprechend werden in der Abbildung 5 die Knoten *u4* und *r1* dargestellt. Im Gegensatz zu anderen Felder bildet der Knoten *r1* bildet nicht einen Wert ab, sondern als eine Instanz des T-Box Konzepts [3, 4], eine spezifische semantische Rolle von diesem URL-Element impliziert.

---

[6] Die Details und die Eigenschaften von e1 sind in der Abbildung nicht weiter dargestellt

[7] Im Endpunkt durch <*id*> dargestellt

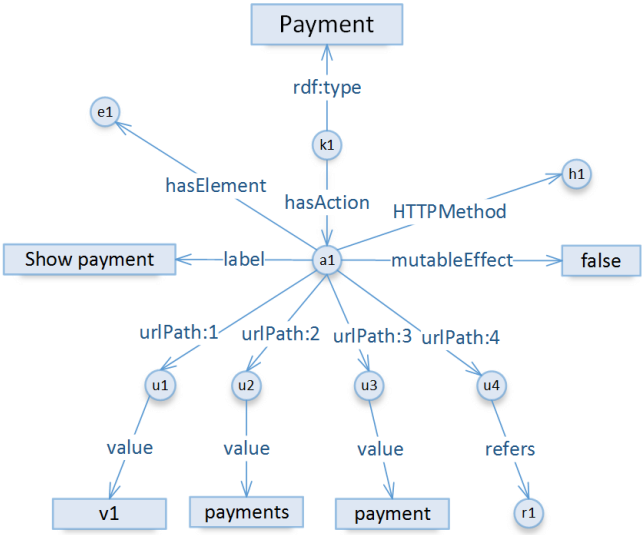

Abbildung 5: Model einer Paypal-Zahlungsanweisung in PayPal-API

Abbildung 6 zeigt das Ergebnis eines API-Aufrufs am Paypal-Sammelpunkt[8]. Der Code beinhaltet zwei Elemente, wobei jedes Element einer Zahlungsanweisung entspricht. Diese Elemente sind auch separat durch den für die einzelnen Elemente designierten Endpunkte abrufbar[9].


```
{
  "payments": [
    {
      "id": "PAY-0US81985GW1191216KOY7OXA",
      "create_time": "2014-06-30T23:48:44Z",
      "update_time": "2014-06-30T23:49:27Z",
      "state": "approved",
      "intent": "order",
      "payer": {
        "payment_method": "paypal"
        ...
      },
      "transactions": [...]
    },
    {
      "id": "PAY-53485002LD6169910KOZQ25I",
      "create_time": "2014-07-01T19:35:17Z",
      "update_time": "2014-07-01T19:36:05Z",
      "state": "approved",
      "intent": "order",
      "payer": {
        "payment_method": "paypal"
        ...
      },
      "transactions": [...]
    },
  ],
  "count": 2,
  "next_id": "PAY-9X4935091L753623RKOZTRHI"
}
```


Abbildung 6: Liste von PayPal-Transaktionen

[8] Wie in der Abbildung 4 dargestellt

[9] Entspricht der Abbildung 5

Jedes Element in der Liste wird durch das gleiche Datenformat repräsentiert und mittels eines Identifikators referenziert. Der Selektor ermöglicht das dynamische und kontextabhängige Referenzieren oder die Auswahl jedes Elements, auch wenn die Entitäten selbst noch unbekannt sind.

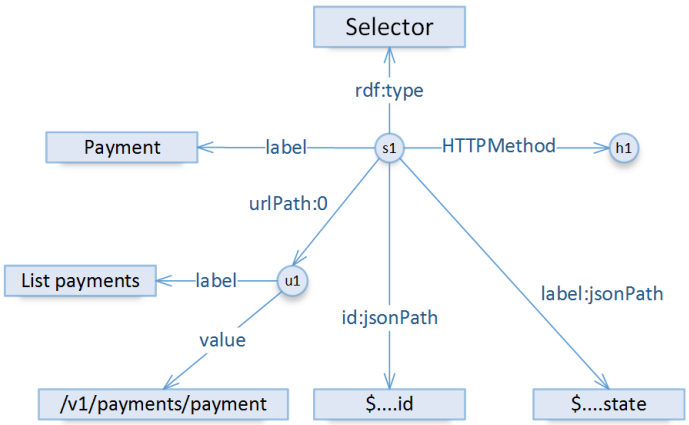

Abbildung 7: Selektor für Identifikator der Paypal-Zahlungsanweisung

Abbildung 7 zeigt, wie der Selektor für die PayPal-Zahlungsanweisung beschrieben ist. Basierend auf diesen Daten kann ein automatisiertes Tool oder einer Agent erstellt werden, der die Wiederverwendung und das Referenzieren von existierenden Instanzen ermöglicht. Zu Beispiel-anwendungen zählen etwa die Erstellung von Sicherheitsrichtlinien höchster Granularität, oder die Verwendung bei automatisierten Benutzeroberflächen.

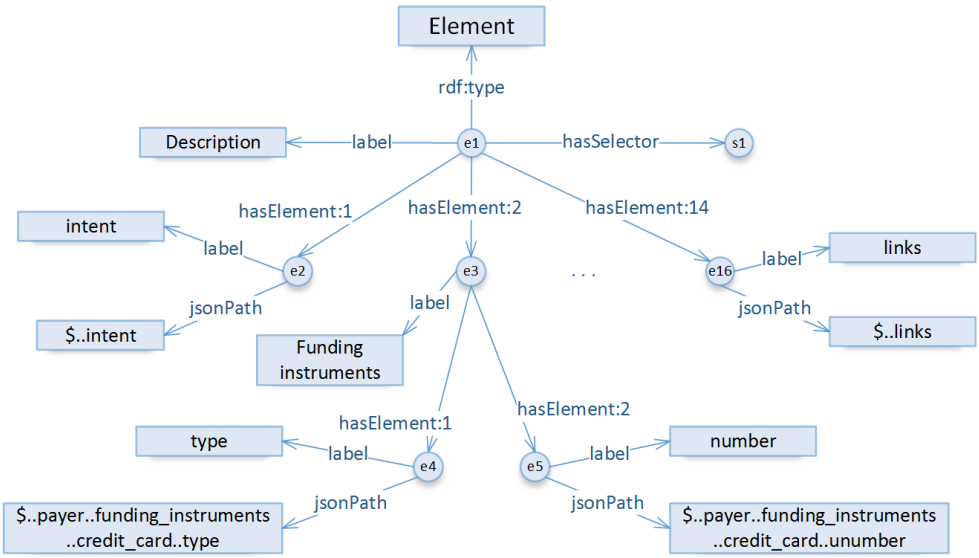

Abbildung 8: Zahlungsanweisung bei PayPal

Das Beispiel in Abbildung 8 zeigt, wie die Zahlungsanweisung in der PayPal-API strukturiert ist. Wie bei anderen Entitäten wird das Element auch als Knoten in einem Graphen dargestellt. Somit ist der Knoten *e1* (in der Abbildung 8) in selber Form überall zugreifbar – auch in der Abbildung 5.

Zum Zwecke der Übersichtlichkeit zeigt die in der Abbildung 8 dargestellte Struktur insgesamt sechs Elemente. Eine Struktur kann im Prinzip eine beliebige Anzahl von Elementen enthalten, was vom angestrebten Granularitätsgrad abhängig ist. In diesem Fall sind für Paypal-Zahlungsanweisung vierzehn Elemente vorgesehen, wobei jedes Element als eine Teilstruktur in

JSON betrachtet werden kann. Zur Vereinfachung haben wir aber nur fünf Elemente graphisch dargestellt.

```
{
  "id": "PAY-5YK922393D847794YKER7MUI",
  "create_time": "2013-02-19T22:01:53Z",
  "update_time": "2013-02-19T22:01:55Z",
  "state": "approved",
  "intent": "sale",
  "payer": {
    "payment_method": "credit_card",
    "funding_instruments": [
      {
        "credit_card": {
          "type": "mastercard",
          "number": "xxxxxxxxxxxx5559",
          "expire_month": 2,
          "expire_year": 2018,
          "first_name": "Betsy",
          "last_name": "Buyer"
        }
      }
    ]
  },
  ...
  "links": [
    {
      "href":
"https://api.paypal.com/v1/payments/payment/PAY-
5YK922393D847794YKER7MUI",
      "rel": "self",
      "method": "GET"
    }
  ]
  ...
}
```

Abbildung 9: Liste von PayPal-Transaktionen

Mögliche Granularitätsebenen können anhand des Beispiels in der Abbildung 9 illustriert werden. In Abbildung 8 sind Werte für Elemente modelliert: *intent*, *credit_card_type* ,*credit_card_number* und *links*. Das Beispiel zeigt, wie die einzelnen Elemente extrahiert und referenziert werden können.

3. Fazit

Im Zuge dieses Projekts wurde ein Konzept erarbeitet und praktisch umgesetzt, das die maschinenlesbare Strukturierung von verschiedenen Web-APIs ermöglicht. Diese Strukturierung wird zuerst für die Zwecke Sicherheitsmanagements und dessen Durchsetzung angewandt.

In dieser Iteration wurden zuerst 20 APIs aus 7 unterschiedlichen Produktkategorien analysiert. Diese APIs werden von populären Diensten zur Verfügung gestellt und oft im Rahmen von Integrationsprozessen als Dienste oder Datenquellen in umfassenderen Workflows ausgeführt.

Basierend auf den in der ersten Iteration gewonnenen Erkenntnissen wurde ein konzeptuelles Meta-Model entwickelt, das die Strukturierung von APIs in diversen Kategorien ermöglicht. Dieses Konzept ermöglicht somit eine breitere und abstraktere Anwendung, als es in Vorgängerprojekten realisiert wurde. Das wurde praktisch realisiert und gegen Case-Szenarien bei ausgewählten

Anwendungen getestet. Als Ergebnis dieser Aktivitäten wurde ein Softwareframework um die neue Bibliothek erweitert.

4. Referenzen

[1] B. Suzic, *Multidimensional Security Policies,* Graz: Zentrum für sichere Informationstechnologie - Austria (A-SIT), 2016.

[2] B. Suzic, „User-centered Security Management of API-based Data Integration Workflows," *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium,* pp. 1233-1238, 2016.

[3] W3C Owl Working Group, *OWL 2 Web Ontology Language Document Overview,* W3C, 2009.

[4] R. Cyganiak, D. Wood und M. Lanthaler, „RDF 1.1 concepts and abstract syntax," W3C, 2014.

[5] D. Hardt, *The OAuth 2.0 authorization framework.,* 2012.

[6] I. Salvadori und F. Siqueira, *A Maturity Model for Semantic RESTful Web APIs,* IEEE, 2015.

[7] M. Sporny und L. Markus, *Json-ld 1.0-a json-based serialization for linked data,* W3C, 2014.

[8] B. Suzic, „Securing integration of cloud services in cross-domain distributed environments," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016.