Linking Services to Websites by Leveraging Semantic Data

D. Wolters, S. Heindorf, J. Kirchhoff, and G. Engels

Linking Services to Websites by Leveraging Semantic Data

Dennis Wolters
Paderborn University
dennis.wolters@uni-paderborn.de

Stefan Heindorf
Paderborn University
heindorf@uni-paderborn.de

Jonas Kirchhoff
Paderborn University
jonaskir@mail.uni-paderborn.de

Gregor Engels
Paderborn University
engels@uni-paderborn.de

***Abstract*—Websites increasingly embed semantic data for search engine optimization. The most common ontology for semantic data, schema.org, is supported by all major search engines and describes over 500 data types, including calendar events, recipes, products, and TV shows. As of today, users wishing to pass this data to their favorite applications, e.g., their calendars, cookbooks, price comparison applications or even smart devices such as TV receivers, rely on cumbersome and error-prone workarounds such as reentering the data or a series of copy and paste operations. In this paper, we present Semantic Data Mediator (SDM), an approach that allows the easy transfer of semantic data to a multitude of services, ranging from web services to applications installed on different devices. SDM extracts semantic data from the currently displayed web page on the client-side, offers suitable services to the user, and by the press of a button, forwards this data to the desired service while doing all the necessary data conversion and service interface adaptation in between. To realize this, we built a reusable repository of service descriptions, data converters, and service adapters, which can be extended by the crowd. Our approach for linking services to websites relies solely on semantic data and does not require any additional support by either website or service developers. We have fully implemented our approach and present a real-world case study demonstrating its feasibility and usefulness.**

***Index Terms*—Services; Websites; Semantic Data; schema.org; Data Conversion; Interface Adaptation; Mediation**

I. Introduction

As of today, a lot of websites do not only display data to end users, but also embed semantic data understandable by machines. The main motivation for providing semantic data is search engine optimization (SEO), since the presence of semantic data improves a website's search ranking and allows search engines to display parts of this data prominently within search results. At the time of writing, over 1 billion web pages, i.e., more than 38% of pages on the web, offer semantic data [1]. The de-facto standard to describe this data is schema.org, which is supported by all major search engines and defines over 500 data types, including events, recipes, products, and TV shows.

The data provided by websites is not only relevant for search engines, but also for a lot of services provided by user-oriented applications. For example, users would like to insert events into their calendars, add recipes to their electronic cookbooks, or send information about their favorite TV shows to their TVs. Unfortunately, most websites only provide links to a small set of services, like social media sites, while the majority of services are neglected. Users are left alone to transfer the data manually, which is a cumbersome and error-prone task. Merely, adding an event to a calendar application requires copying at least start date, end date, and event name as well as optionally further fields like event description, repetitions, location, attendees, or attachments. Automating this task might be relatively easy for a single example, however, the conversion from hundreds of semantic data types to thousands of heterogeneous data formats of different services is a major challenge. An additional challenge arises if the target application runs on another device. This is for example the case if the calendar is managed solely on the user's smartphone or if a TV receiver needs to be programmed to record a TV event listed on a website.

In this paper, we present Semantic Data Mediator (SDM), an approach that empowers end users to link a wide range of services to websites. SDM consists of four major steps: (1) Extracting semantic data from the currently open web page with a browser extension, (2) offering suitable services on this data to the user (3) converting the data to appropriate data formats, and (4) forwarding the data to a wide range of services, including web services as well as those provided by locally-installed applications and smart devices. For doing so, we introduce an approach for automatic converter composition taking both semantic data types and syntactic data formats into account. Moreover, we employ service adapters to invoke services not offering a web-accessible interface. For finding suitable services, our approach uses a personalized service registry containing only those services relevant for a specific user. To enable reuse, the personalized service registry in turn is derived from a large, central service repository, which can be extended by the crowd.

Our approach does not require any additional effort by either website or service developers. Website developers simply continue adding semantic data to their websites for SEO and application developers develop apps in the usual manner. Thereupon, our approach flexibly combines data conversion and service adapters to support a wide range of services and websites. As we show in our case study, this enables unprecedented use cases for the data provided by websites.

The remainder of this paper is organized as follows: Section II reviews related work. Section III gives an overview of our approach. Subsequently, Section IV provides a detailed description of how we link services to websites. Section V extends our approach to also work in combination with locally-installed applications. Section VI gives details of our implementation and Section VII demonstrates the feasibility and usefulness of our approach in a case study. Section VIII concludes the paper and gives an outlook on future work.

II. Related Work

While it is common practice to link, compose and integrate services [2, 3, 4, 5] with each other as well as to do the same with semantic data [6, 7, 8], there is no comprehensive approach allowing end users to bridge the gap between the world of services and the world of data. As we will show in this section, all existing approaches are either very restricted (Browser Extensions, Share Buttons), tackle only the provisioning of an interface, but do not handle service integration (Adapter Services), or aim at creating new composite applications (End-User Service Compositions).

Browser Extensions: There exist a few browser plugins which link services to websites. However, almost all are specific to single services, for example, offering a “Skype” button or “Send to Kindle” button. Perhaps the two most similar approaches to ours are Semantic Spider [9] and Piggy Bank [10]. They provide browser plugins to parse semantic data on websites and offer simple operations on the data, e.g., viewing them or storing them in a local database. However, in contrast to our approach, they do not allow to link a wide range of services to the data. For example, their approach works for web-based calendars, but is not extensible for a TV receiver, smart refrigerator, or an Android cookbook application.

Share Buttons & Co: Most major web applications like Twitter, Facebook, or Google provide APIs such that their services can be integrated by website developers. For integrating multiple of these services at once, intermediary services like *sharethis.com* or *addthis.com* emerged. However, those services seldom go beyond sharing simple content and they are restricted to a few popular services. Also, a major drawback is that website visitors depend on website developers embedding them.

Adapter Services: Han and Tokuda [11] tackle the problem of many web applications not offering machine-accessible APIs by proposing an adapter service: Their approach extracts HTML elements such as lists, tables or images from websites and offers them as a service. The service requester has to give detailed instructions how to find the element in the page, e.g., by giving the name or ID of the HTML element. Hence, their approach remains on a syntactic level. In contrast, our approach operates on a semantic level, e.g., we are able to recognize that a table represents a recipe and that it can be forwarded to a cookbook application. Besides this adapter service for web applications, there are approaches offering data which is stored on Android devices as a service [12, 13]. Those approaches, however, need to be explicitly integrated by website developers, and they are somewhat complementary to our approach as we are transferring data in the opposite direction.

Semantic Services: Approaches for semantic service description and composition [14, 3, 15, 2, 4, 16, 17] operate on semantic data. However, they do not bridge the gap between the world of services and the world of data. They enrich existing services by semantic annotations, detailing the services’ operations and parameters on a high conceptual level with concepts from ontologies. In contrast to our approach, they rely on existing service endpoints and cannot directly operate on data. Furthermore, while there is a standard for semantic data on websites (schema.org), as of today, there is no standardized ontology for semantic service descriptions necessitating for error-prone conversion between heterogeneous ontologies [18].

End-User Service Composition: Various approaches enable end users to create service compositions [19]. The goal of such compositions is often a mashup, a composite application based on services provided by other applications [20], e.g., mashups of web applications [21, 22] or user interface integration using ontologies [23]. We do not strive to create new composite applications, but to link existing services to existing websites to enable a large variety of new use cases for the provided data. Wu et al. [24] present an approach for the fast retrieval of compositions. In contrast to our approach, their approach neither takes data mediation into account nor does it distinguish between data types and formats.

III. Overview of Service-to-Website Linking

In this section, we give an overview of Semantic Data Mediator (SDM), our approach to link services to websites. We introduce our design goals, the solution idea, and an example scenario illustrating the usage of SDM.

A. Design Goals

Our approach shall be usable *independent* of the support by both website and service developers, because it cannot be assumed that website developers integrate all services that users are potentially interested in. For example, website developers might not even be aware of a service’s existence or the service integration effort might be prohibitively expensive. Hence, we do not require website developers to integrate our approach, but instead rely on the presence of semantic data due to SEO. Similarly, we cannot expect all service providers to offer browser extensions or other means enabling the usage of their services in combination with any website.

Furthermore, our approach shall support a *multitude* of websites and services: Users interact daily with numerous websites, offering news, events, recipes, etc., as well as with all kinds of services offered by web applications, or applications installed on their smartphones, tablets and desktop computers. Moreover, different users expect different services on the same data, for example, users use different web applications or have different applications installed on their devices. It shall be possible to conveniently transfer data from websites to all those services provided by applications, independent of the device on which they are provided. Therefore, we need to support data conversion, since the same semantic data, might be represented in various different data formats by services.

B. Solution Idea

Figure 1 gives an overview of our approach. Instead of requiring website developers to link services to websites, we utilize the embedded semantic data originally intended for search engines. This enables end users to independently create novel and unforeseen use cases for the data provided by these websites going far beyond mainstream ‘share’ buttons.

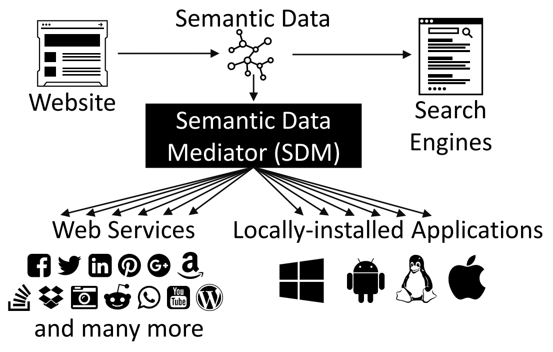

Figure 1. Linking services to websites: We extract standardized semantic data from websites and offer this data to a multitude of services, including a user's web services and application-based services on local devices.

We rely on semantic data for mainly two reasons: (1) By the end of 2016, over 38% of websites already offered semantic data [1]. This is an increase of 27% compared to 2015 and we expect this number to increase even further, since providing semantic data as part of SEO is important for a website's success. Moreover, (2) techniques embedding and describing semantic data are standardized (see Subsection IV-A). Consequently, by using embedded semantic data, we neither need to depend on proprietary and heterogeneous APIs provided by website developers to gain their data, nor to depend on error-prone scraping techniques to infer the data semantics from HTML.

To support a multitude of services, we cannot simply focus on web services. To also enable the linking of services provided by applications installed locally or on other devices, we employ various different service adapters. By utilizing a central repository consisting of service and converter descriptions, users derive personal service registries containing only those services they are interested in, i.e., those provided by applications they are using. Based on the selected services, converters to translate data according to the needs of the respective services are added to the registry as well. Thereby, we are not limited to the few services able to consume semantic data directly. To enable the usage of locally-installed applications, users can install a variety of service adapters. We already developed a large number service adapters, e.g., for Android and Windows applications, as well as data converters, but for our approach to scale to millions of services, we plan on harnessing the power of the crowd. For this, we have simplified adapter and converter development and allow the extension of our repository by end user developers.

C. Example Scenario

For describing our approach, we consider the following simple scenario: Alice is sitting in front of her desktop computer and finds a soup recipe on the web. Now she wants to add this recipe to the cookbook app on her smartphone, which she uses to manage all her favorite recipes. Unfortunately, the website cannot provide a simple button for this case, since the cookbook app resides on a different device. Also, the cookbook app only allows to import recipes described in a proprietary XML format, which is not offered by the website. Without our approach, Alice would need to manually enter this recipe into her cookbook app, which is cumbersome because she needs to add all instructions and ingredients. Copying associated images is even more complicated, since they have to be downloaded, moved to the smartphone and added to the recipe, perhaps even in a specific order.

In contrast, Figure 2 shows our approach. It consists of a personal service registry, knowing about services relevant for Alice, e.g., the import service of her cookbook app. When Alice visits the recipe website, our approach detects the semantic data describing the recipe and offers additional services, including the one to import the recipe into her cookbook app installed on her smartphone. When she selects this service, the recipe is converted into the suitable proprietary XML format with references to associated images. The converted recipe is then passed to the import service of her cookbook app. Most of this happens in the background, while Alice only needs to select the service from the list of services provided by our approach.

Of course, this is just one of many possible usage scenarios for our approach (for more see Section VII). Alternatively, Alice could also send this recipe to one of her friends by using a messenger on her smartphone or share it on basically any social network, including those for which the recipe website does not offer any share buttons.

A technical explanation of our approach is given in the upcoming sections by explaining the different steps shown in Figure 2: Section IV focuses on the service registry, data conversion and linking services to websites, whereas, Section V describes service adapters and invocation.

IV. Linking Services to Websites

In this section, we describe how our approach links services to websites. For our description, we follow the example scenario in Figure 2 and focus on the first six steps, while the last two steps are discussed in Section V.

A. Semantic Data Formats and Ontologies

When Alice visits the recipe website in Step 1 of Figure 2, she only sees a human-readable version of the recipe. If the site is optimized for search engines, like most commercial websites, it also contains machine-understandable semantic data describing the shown recipe. This semantic data is invisible for website users, and as mentioned in Section I, is mainly used by search engines to semantically answer search requests or to display the data more prominently within search results. There are four common techniques to embed semantic data into websites. They can be distinguished according to whether or not they utilize an external ontology, and how they are embedded in a web page.

JSON for Linked Data (JSON-LD) is a lightweight extension of the popular JSON format and enables the description of semantic data. JSON-LD objects can be stored in an invisible part of a web page, e.g., in a script tag. The data is typed over an external ontology. In practice, the by far most common ontology is schema.org. Listing 1 shows an example for a recipe from the page in Figure 2. The `@context` property

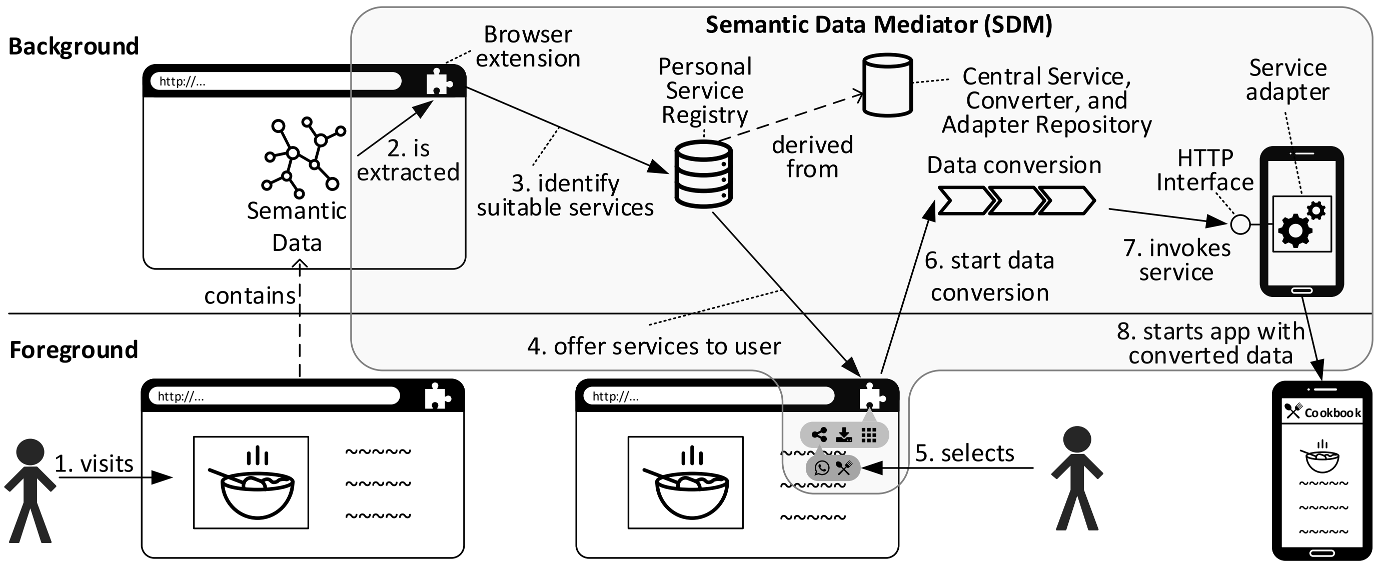

Figure 2. Example usage of our approach to add a recipe from a desktop browser with a cookbook app on a smartphone: (1) A user visits a website containing semantic data. (2) Our browser extension extracts the semantic data and (3) queries the personal service registry for suitable services. (4) The browser extension offers the services to the user who (5) selects the desired service. (6) Then the semantic data is converted as needed by the selected service before (7) a service adapter on the smartphone is invoked that (8) forwards the data to a cookbook app on the user's smartphone.

Listing 1. Example of a JSON-LD object describing a simple soup recipe by utilizing the type Recipe from the schema.org ontology

```
{
  "@context": "schema.org",
  "@type": "Recipe",
  "name": "Simple Soup",
  "ingredients": ["Water", "Instant Soup"]
}
```

defines the ontology, the `@type` property the type within this ontology, and dependent on the type, there are further properties such as `name` and `ingredients` in this recipe example.

Besides JSON-LD, further techniques to provide semantic data in websites include RDFa, Microdata and Microformats. The former two utilize external ontologies like schema.org, too, but in contrast to embedding JSON-LD, special HTML attributes are used to annotate the existing (visible) HTML elements in a web page. While adding those fine-grained annotations involves more effort for website developers, for data consumers, it becomes easier to match semantic annotations to visible content. Similarly, Microformats are also an annotation technique, but they do not utilize an external ontology. Instead about 20 different Microformats have been defined to describe data types like events, contact data, or recipes. By the end of 2016, Microdata was the dominating technique to provide semantic data in websites [1]. Nevertheless, we support the extraction of all these formats.

B. Extracting Semantic Data

We developed a browser extension to extract all semantic data items from the currently opened web page (see Step 2). The result of the extraction is a list of JSON-LD objects, even if RDFa, Microdata or microformats have been used to provide semantic data. By unifying the data format, we simplify further processing. In case of Microformats, the context is "microformats.org". In case of RDFa, Microdata, or embedded JSON-LD any ontology may provide the context, but most websites use schema.org. There is already a significant number of web pages providing semantic data, and we leave it for future work to integrate information extraction techniques to also extract semantic data from web pages, which do not natively contain semantic annotations.

C. Personal Service Registry

A user's personal service registry is the prerequisite to identify suitable services for extracted semantic data. This registry maintains descriptions of all services relevant for a user, e.g., those provided by web or locally-installed applications he/she uses. Yet, demanding that every service is directly able to process JSON-LD objects typed over an ontology like schema.org would greatly limit the number of services that can be linked to a website. Therefore, the registry also maintains descriptions of converter services capable of transforming data so that it can be consumed by other services.

We support all web services having a RESTful HTTP interface. Thus, our approach is applicable on most modern web services, since they already provide such an interface and we can apply interface adaptation to support services providing other types of interfaces (see Subsection V). To add a RESTful HTTP service to our registry, it is described using the OpenAPI specification[1,2], which is tailored to describe RESTful HTTP interfaces and has several benefits, e.g., editor tool support and automated generation of documentation.

Listing 2 shows the description of a converter service which is able to translate a schema.org `Recipe` provided in JSON-LD to the MyCookbook XML format [25]. The latter is the

[1] formally known as Swagger specification [2] http://swagger.io/specification

Listing 2. OpenAPI description of a service able to convert schema.org Recipes to HTML (simplified)

```
 1 swagger: '2.0'
 2 title: Recipe2MyCookbookXML
 3 host: recipe2mycb.converter.example.com
 4 schemes: http
 5 paths:
 6  /convert:
 7   post:
 8     description: Converts Recipe to MyCookbookXML
 9     consumes: application/ld+json
10     parameters:
11       - name: object
12         in: body
13         description: Recipe to be converted
14         required: true
15         schema:
16           $ref: 'schema-org.json#/Recipe'
17     produces: application/x.mycb+xml
18     responses:
19       200:
20         schema:
21           $ref: 'mycb-xml-schema.json#/cookbook'
```

official import/export format of the MyCookbook Android app[3], which we have used in our case study to realize the scenario depicted in Figure 2. The first four lines of Listing 2 contain general information about the service, like specification version and title as well as the host's address and supported schemes. Starting from Line 5, provided endpoints and operations are described. In this case, we have a single endpoint (*/convert*) which only supports one operation (a POST request). Lines 9 to 16 describe how the converter service expects input data, whereas Lines 17 to 21 describe how output data is provided.

We distinguish between two important aspects of input and output data: *data type* and *data format*. The former describes a certain type of data on a semantical level, like an event or a recipe, whereas the latter describes how certain data can be serialized, e.g., as JSON-LD objects. In a service specification, the data format of the input and output of an operation is specified by using *media types* (see Lines 9 and 17), since this is the standard for HTTP-based interfaces. The Internet Assigned Numbers Authority (IANA) maintains a list of official media types (formerly known as MIME types).[4] This list contains media types for most major data formats like iCalendar, vCard, PDF and many more. The media type for JSON-LD is `application/ld+json`. The part before the slash specifies the top level type, e.g., text, video, audio, or application. The actual name of the media type follows after the slash. Suffixes like `+json` and `+xml` indicate that the format is a specialization of JSON or XML respectively. Media types not officially listed by the IANA must carry the prefix "`x.`" in their name. We use such unofficial media types to distinguish between services that consume/produce any type of XML or JSON, and those requiring a specialization of these formats, e.g., we use `application/x.mycb+xml` to refer to the MyCookbook XML format.

Data types are specified by either ontologies, like schema.org, or in case of XML and JSON by schema definitions, e.g., `cookbook` is a type defined in the MyCookbook XML schema [25]. In contrast to ontologies, schemas are not language-agnostic. For instance, schema.org is independent of a concrete data format and can be used in combination with multiple formats, like JSON-LD, RDFa, or Microdata. A JSON or XML schema, however, is bound to JSON or XML, respectively. To encode that the example converter service expects a schema.org `Recipe` and not just any type of JSON-LD object, we have translated the schema.org ontology into a JSON schema, which can be referenced by an OpenAPI specification (see Line 16). Similarly, Line 21 references the type `cookbook` from the MyCookbook XML schema to define that the result of the conversion is an XML-based cookbook.

[3] http://mycookbook-android.com/ [4] http://iana.org/assignments/media-types

D. Central Repository

We maintain a central repository to enable the easy addition of further services to the personal service registry by allowing users to reuse existing service descriptions, converter, and adapter descriptions. Users can search this repository and add relevant services from this central repository to their personal service registry. When doing so, users may have to provide additional parameters, e.g., credentials for services requiring authentication. Furthermore, users can find existing converters and do not need to develop those converters themselves. Service and converter descriptions can be added to the central repository by their respective service developer, or alternatively, by end user developers who want to publish their own descriptions.

E. Finding Suitable Services

Based on the information maintained by the service registry, we want to find suitable services for extracted data items (Step 3). Unfortunately, most services do not directly operate on semantic data, like a schema.org `Recipe` provided as a JSON-LD object, but instead require the data to be converted first, e.g., to the MyCookbook XML format. Our registry exploits information about available converter services to compute possible converter compositions, which enable the usage of services that do not directly consume semantic data.

To find all services capable to process a given data item either directly or indirectly via a sequence of conversions, the service registry internally maintains a graph in which each data type and data format is represented as a node (see Figure 3). Every data type node is contained in a data format node, since a serialization format is always needed. However, not every data format is able to distinguish between different data types. Those not supporting this differentiation do not contain any data type nodes, e.g., plain text or PDF. Currently, we support the type/format distinction for JSON-LD, because it allows to reference external ontologies, and for JSON and XML, since OpenAPI supports schema definitions for those formats. In between these nodes, there are two types of edges: Inheritance edges (—▷) and converter edges (➝). *Inheritance edges* specify the type hierarchy both of data types and data formats. For data types, information about the type hierarchy is available from the respective ontology or schema associated with the service. The inheritance between data formats is given implicitly by the suffixes used in the respective media types (cf. Section IV-C).

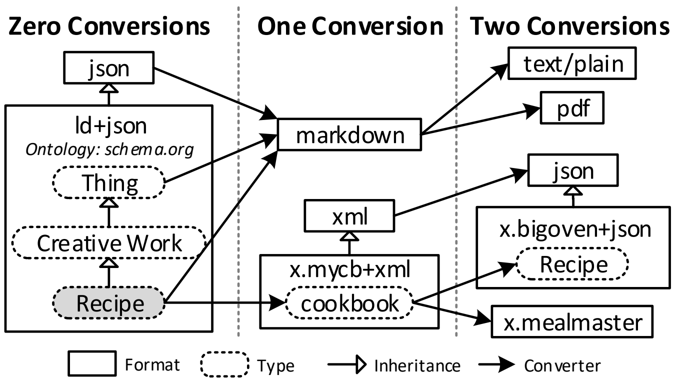

Figure 3. Excerpt of the internal graph maintained by the service registry to determine to which other data formats and types a data item can be converted

Converter edges indicate that there is a converter service able to translate from one type to another. The edge labels specify the corresponding converter service (for better readability those labels are omitted from Figure 3). If there are multiple converter services able to convert between the same types, we allow multiple edges between two nodes. An example excerpt of the graph maintained by the registry is depicted in Figure 3. It shows the ontological inheritance for the schema.org type `Recipe`, which is a subtype of `CreativeWork` which again is a subtype of `Thing`. Also, implicitly we have JSON data, since JSON-LD is specialization thereof. Hence, services able to process any kind of JSON, like a JSON editor or beautifier, could already process this data. In addition, we see that various data conversions are possible, e.g., a recipe can be converted first to Markdown and then into a PDF.

Based on a start node specified by an extracted data item, the registry explores this graph and collects all service operations capable of consuming any type and format reachable by traversing at most k conversion edges. Figure 3 shows an example of types and formats reachable by at most two conversions when starting with data type `schema.org/Recipe` and data format JSON-LD. Inheritance edges are not counted, since no conversion is required. Varying k allows a tradeoff between computational time and risk of conversion problems on the one hand and flexibility on the other hand. To enable reuse of converter services not accepting semantic data directly, k needs to be greater than 1. During our case study we set $k = 2$.

Besides automatic converter compositions, we allow end user developers to manually build converter compositions by using a simple web application. Such manual compositions can be saved as new converter services. The advantage of manually created converter composition is that they can be adapted and extensively tested before they are used. A manually composed converter service counts as a single conversion step within an automatically created converter composition. Hence, for selected cases, more than k converters can be combined.

Finally, the registry returns all collected service operations along with information about the corresponding services and all relevant conversion options. The latter might include different converter compositions leading to the same type and format.

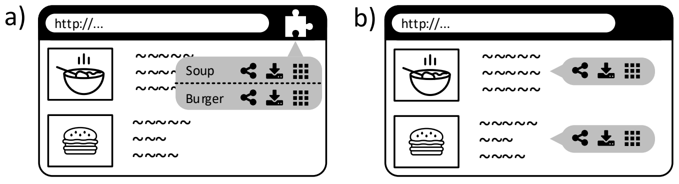

Figure 4. Linking services: (a) single browser-based menu for all data items, (b) multiple injected menus, each specific to the corresponding data item

F. Linking Services to Websites

In the next step, the service operations returned by the registry need to be linked to the website (see Step 4). For convenience, we say that a service is linked to a website if at least one of its operations is linked to this site. If only one data item is described on the website, the services returned by the registry are linked in a browser-based menu like in Figure 2. In case of multiple data items, e.g., two recipes, we follow two strategies: Either distinguish between the data items in the browser-based menu (see Figure 4.a) or inject multiple item-specific menus into the website, which are located directly next to the corresponding data items (see Figure 4.b). The injected menu is currently only supported when Microdata is used to embed semantic data by annotating visible page elements. In both cases, the services linked by our approach are grouped into sharing services, converter services allowing to download the data items in different formats, and other services which do not belong to the first two categories.

G. Data Conversion

When the user selects one of the offered services (see Step 5), the browser extension inspects which conversion steps are needed to translate the data into the needed type and format. If multiple conversion options exist, e.g., two different converter services can be used or different converter compositions exist that lead to the same type/format, we apply a set of heuristics to choose the most appropriate one automatically. A conversion option is preferred over another one if (1) it works on a more specific type, e.g., `Recipe` is more specific than `CreativeWork` or `Thing`, (2) it uses less conversion steps, (3) it is marked as preferred, (4) it was used more often by the user, (5) or it has a better user-rating. Advanced end users can also select a conversion option manually. During this manual selection, converter services can be rated, marked as preferred/default, or permanently disabled. Once a conversion strategy has been selected, automatically or manually, the conversion is initiated (see Step 6). After the data has been converted, it is passed to the selected service.

Due to the absence of services which are able to consume semantic data directly, there is a high demand for converter services for semantic data types, especially those from schema.org. Thus, we provide a converter service template that can be configured by providing a conversion function and meta information like converter name, description, input and output type. This enables developers to fully focus on coding the data transformation, while the corresponding RESTful interface and service registration procedures are automatically provided.

V. Service Adapters

As mentioned in Subsection IV-C, we expect that every service has a RESTful HTTP interface. This basically enables the usage of most modern web services. Unfortunately, nearly all native applications neither offer an external nor an RESTful HTTP interface. Instead, those services just provide an interface to interact with the operating system or other applications on the same device. To support those services as well and not be limited to web services, we employ various forms of adapters:

Android: In [26, 27], we presented the XDAI-A approach which can be used to provide external service interfaces for Android apps. Thereby, the integration of services provided by Android apps is enabled for apps running on different devices as well. To create external service interfaces, XDAI-A provides a domain-specific language and a ready-to-use interpreter. A detailed discuss of XDAI-A is given in [26, 27]. As part of our work on XDAI-A, we already developed adapters for operations commonly provided by Android applications, e.g., sharing or editing data over a RESTful HTTP interface. In combination with the approach presented in this paper, we are now able to use the services provided by Android apps and link them to websites. Thereby, we enable the Steps 7 and 8 depicted in Figure 2 and many more use cases involving Android apps (see Section VII).

Windows: Our Windows service adapter extracts information about installed applications from the Windows registry, i.e., their name and supported data formats. Using this information, we can automatically provide services to open data items in the respective data formats using these applications. An automated registration procedure allows that users can easily add these services to their personal registry.

Scheme-based Applicaton Invocation: Most modern operating systems, including Windows, Android, Linux, MacOS and iOS, support associating applications with certain URI schemes. For instance, the WhatsApp share button used on mobile websites leverages the fact that the app is associated with the scheme `whatsapp://`. By creating a URI using such a scheme, applications can be started with certain parameters, e.g., in the case of WhatsApp, specifying the text to share. For Windows and Android, we already implemented the support for scheme-based application invocation, and analogously, Linux, MacOS and iOS applications can be supported as well.

Command-Line Interface (CLI): A large number of applications for desktop operating systems provide a CLI. We have developed a configurable adapter template that can be used to provide an external RESTful HTTP interface for those applications with just a few lines of code. In particular, this can be very helpful to integrate command-line conversion tools as converter services. For example, by utilizing this adapter template, we provide an adapter for Pandoc[5], a universal document converter supporting over 20 input and more than 40 output formats. Thereby, we have greatly extended the number of possible data conversions and can support a larger number of existing services.

[5] http://pandoc.org/

VI. Implementation

We fully implemented our approach[6] in order to perform a case study discussed in the next section. The browser extension is available for Chrome, but we do not rely on any features unique to Chrome. Thus, we are confident that the extension can be ported to other browsers as well. The converter service template, the CLI adapter, Windows adapter and the personal service registry are developed using JavaScript and the Node.js runtime. The registry uses a document-oriented database (CouchDB) for managing service descriptions and a graph database (Neo4j) to efficiently find conversion options and relevant services. Implementation details of XDAI-A are discussed in [27]. To enable easy deployment, the registry as well as all converters based on our converter service template can be deployed as Docker containers.

VII. Case Study

We have performed a case study to verify whether we actually achieved the design goals formulated in Subsection III-A. Our first goal is to provide a solution that is independent of the website and service developers' support. Hence, we focused on existing websites and services instead of developing our own. Secondly, we want our approach to work for a multitude of services, and therefore, we linked not only web services to websites, but also services of native application running on Windows and Android. Finally, to enable data conversion into a wide range of different formats, we implemented converter services for multiple schema.org types and reused existing conversion tools by using our service adapters.

Over 4 million web pages list schema.org event data [1], but not all of them offer to directly add these events to a calendar application. We provide two simple converters that allow us to pass this event data to Google Calendar and calendar apps on Android devices. Due to our converter service template introduced in Subsection IV-G, the development of these converters only required a few lines of code. To use the Google Calendar, we wrote a short OpenAPI specification, and for Android calendars, we utilized our Android adapter (see Section V). Thereby, we showed that we are able to support existing web services as well as native applications and that it is possible to invoke services of applications running on another device than the one that is currently being used.

There are also over 4 million semantically annotated recipes on the web [1] and we realized multiple use cases for these recipes. We used our converter service template to create a converter from schema.org `Recipe` to Markdown and reused Pandoc utilizing our CLI adapter to enable the download of recipes as Word, PDF, and text documents. By applying our Windows adapter, we can also directly open them in the associated applications. Thanks to XDAI-A, sharing the textual representation is possible with any Android messenger app. Listing 2 shows the specification of a converter from `Recipe` to MyCookbook XML. In addition, we specified a service adapter for the import service of the MyCookbook Android

[6] http://sdm.dwolt.de

app by using XDAI-A. The data converter and service adapter enable that any semantically annotated recipe embedded into a website can directly be added to the MyCookbook app within a single click (like in Figure 2). Similarly, other cookbook apps or web applications could be supported, e.g., bigoven.com already provides an OpenAPI specification of their HTTP interface.

We also linked services of other smart devices to websites by using our approach. For this, we first focused on semantically annotated video objects, e.g., imdb.com provides semantic descriptions of movie trailers. Those videos can be played on a smart TV by the press of a button. We realized this for an Android-based smart TV by again utilizing our Android adapter. Our approach is not limited to Android-based smart devices. For instance, Enigma2-based set-top boxes like the Dreambox or VU2 directly provide an HTTP interface to remotely control the whole device. For these devices, we were also able to support the same video playback functionality as for the Android-based smart TVs. In addition, those devices provide the service to record TV shows. Thereby, we enable that a set-top is being programmed to record a show directly from the website which describes it as a schema.org TV event.

During our case study, we realized various new use cases for data provided by existing websites without needing to change existing websites or services. Thus, we have successfully shown that semantic data can be used as an enabler to link services to websites. Already over 38% of all web pages provide semantic data and since any website seeking commercial success needs to be search engine optimized, we believe this percentage will increase even further. Hence, our approach will be applicable to even more websites in the future. Similarly, more and more devices are becoming smart, and in the future, it might by possible to send a recipe directly to a smart kitchen device, e.g., to a smart refrigerator to check for available ingredients.

VIII. Conclusion and Outlook

In this paper, we propose Semantic Data Mediator as a novel approach to link services to websites by leveraging semantic data. Our approach extracts semantic data on the client-side using a browser extension. Services able to consume this data are identified by querying a user-specific service registry. Thereafter, the data is converted and a wide range of services can be invoked, including web services as well as services offered by locally-installed applications and smart devices. Using this approach, end users are no longer restricted to the few services integrated by website developers and can independently link a multitude of services, e.g., to process calendar events, recipes, products, TV events, and many more.

We are currently working on further service adapters for Linux, MacOS and iOS applications. In addition, we are exploring how information extraction techniques can be integrated into our approach to enable the use of SDM on websites not directly containing semantic annotations. Moreover, we are developing a methodology to simplify the converter development to an even greater extent. Last but not least, we plan to deploy our approach and to offer additional tool support as well as guidance for end user developers to integrate further services.

References

[1] C. Bizer, R. Meusel, and A. Primpeli, "Web Data Commons - RDFa, Microdata, and Microformat Data Sets." [Online]. Available: http://webdatacommons.org/structureddata/

[2] S. Dustdar and W. Schreiner, "A Survey on Web Services Composition," *International Journal of Web and Grid Services*, vol. 1, no. 1, pp. 1–30, 2005.

[3] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," in *Semantic Web Services and Web Process Composition*, ser. LNCS. Springer, 2004, no. 3387, pp. 43–54.

[4] S. Kona, A. Bansal, and G. Gupta, "Automatic Composition of Semantic Web Services," in *ICWS 2007*. IEEE, 2007, pp. 150–158.

[5] Y. Syu, S. P. Ma, J. Y. Kuo, and Y. Y. FanJiang, "A Survey on Automated Service Composition Methods and Related Techniques," in *SCC 2012*, 2012, pp. 290–297.

[6] C. Bizer, "The Emerging Web of Linked Data," *IEEE Intelligent Systems*, vol. 24, no. 5, pp. 87–92, 2009.

[7] P. N. Mendes, H. Mühleisen, and C. Bizer, "Sieve: Linked Data Quality Assessment and Fusion," in *EDBT/ICDT Workshops 2012*. ACM, 2012, pp. 116–123.

[8] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion," in *KDD 2014*. ACM, 2014, pp. 601–610.

[9] H. Uchida, R. Swick, and A. Sambra, "The Web Browser Personalization with the Client Side Triplestore," in *ISWC 2014*, ser. LNCS. Springer, 2014, no. 8797, pp. 470–485.

[10] D. Huynh, S. Mazzocchi, and D. Karger, "Piggy Bank: Experience the Semantic Web Inside Your Web Browser," in *ISWC 2005*. Springer, 2005, pp. 413–430.

[11] H. Han and T. Tokuda, "A Method for Integration of Web Applications Based on Information Extraction," in *ICWE 2008*, 2008, pp. 189–195.

[12] J. David and J. Euzenat, "Linked Data from Your Pocket: The Android RDFContentProvider," in *ISWC 2010 - Posters & Demonstrations Track*. CEUR-WS.org, 2010, pp. 129–132.

[13] M.-E. Roşoiu, J. David, and J. Euzenat, "A Linked Data Framework for Android," in *The Semantic Web: ESWC 2012 Satellite Events*, ser. LNCS. Springer, 2012, no. 7540, pp. 204–218.

[14] S. A. McIlraith, T. C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 46–53, 2001.

[15] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara, "Bringing Semantics to Web Services: The OWL-S Approach," in *Semantic Web Services and Web Process Composition*, ser. LNCS. Springer, 2004, no. 3387, pp. 26–42.

[16] D. Kourtesis and I. Paraskakis, "Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery," in *The Semantic Web: Research and Applications*, ser. LNCS. Springer, 2008, no. 5021, pp. 614–628.

[17] ——, "Web Service Discovery in the FUSION Semantic Registry," in *Business Information Systems*. Springer, 2008, pp. 285–296.

[18] P. Shvaiko and J. Euzenat, "Ontology Matching: State of the Art and Future Challenges," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 158–176, 2013.

[19] F. Hang and L. Zhao, "Supporting End-User Service Composition: A Systematic Review of Current Activities and Tools," in *ICWS 2015*. IEEE, 2015, pp. 479–486.

[20] F. Daniel and M. Matera, *Mashups*. Springer, 2014.

[21] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding Mashup Development," *IEEE Internet Computing*, vol. 12, no. 5, pp. 44–52, 2008.

[22] M. Husmann, M. Nebeling, S. Pongelli, and M. C. Norrie, "MultiMasher: Providing Architectural Support and Visual Tools for Multi-device Mashups," in *WISE 2014*, ser. LNCS, no. 8787. Springer, 2014, pp. 199–214.

[23] H. Paulheim, *Ontology-based System Integration*. Springer, 2011.

[24] Y. Wu, C. Yan, Z. Ding, G. Liu, P. Wang, C. Jiang, and M. Zhou, "A Multilevel Index Model to Expedite Web Service Discovery and Composition in Large-Scale Service Repositories," *IEEE Trans. Services Computing*, vol. 9, no. 3, pp. 330–342, 2016.

[25] Maadinfo Services, "My CookBook XML Schema." [Online]. Available: http://mycookbook-android.com/site/my-cookbook-xml-schema/

[26] D. Wolters, J. Kirchhoff, C. Gerth, and G. Engels, "Cross-Device Integration of Android Apps," in *ICSOC 2016*, ser. LNCS. Springer, 2016, no. 9936, pp. 171–185.

[27] ——, "XDAI-A: Framework for Enabling Cross-Device Integration of Android Apps," in *ICSOC 2016 Workshops and Satellite Events*, ser. LNCS. Springer, 2016, (in press).