Distributed Affordance: An Open-World Assumption for Hypermedia

Ruben Verborgh
Ghent University – iMinds – Multimedia Lab
9050 Ghent, Belgium
ruben.verborgh@ugent.be

Michael Hausenblas
DERI, NUI Galway
IDA Business Park
Lower Dangan Galway, Ireland
michael.hausenblas@deri.org

Thomas Steiner
Universitat Politècnica de Catalunya – Department LSI
08034 Barcelona, Spain
tsteiner@lsi.upc.edu

Erik Mannens
Ghent University – iMinds – Multimedia Lab

Rik Van de Walle
Ghent University – iMinds – Multimedia Lab

ABSTRACT

Hypermedia links and controls drive the Web by transforming information into affordances through which users can choose actions. However, publishers of information cannot predict all actions their users might want to perform and therefore, hypermedia can only serve as the engine of application state to the extent the user's intentions align with those envisioned by the publisher. In this paper, we introduce *distributed affordance*, a concept and architecture that extends application state to the entire Web. It combines information inside the representation with knowledge of action providers to generate affordance from the user's perspective. Unlike similar approaches such as Web Intents, distributed affordance scales both in the number of actions and the number of action providers, because it is resource-oriented instead of action-oriented. A proof-of-concept shows that distributed affordance is a feasible strategy on today's Web.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

affordance, distributed systems, hypermedia, REST, Web

1. INTRODUCTION

The world around us is filled with *affordances*—properties of objects that allow us to perform actions. For example, a door handle is the affordance that lets us open a door, and a pen is the affordance that allows us to write a note. However, that same pen can afford stirring a cup of coffee and, with some skill, even opening a beer bottle. While psychologist James J. Gibson initially defined "affordance" in 1977 as an objectively measurable action opportunity provided by an object [18], it was Donald Norman who saw the potential of making those opportunities *subjective*, depending on who wants to use the object [25].

Little did Gibson and Norman know that their notion would become a key concept of the World Wide Web. Thanks to the invention of hyperlinks, documents became affordances through which actions could be performed. More than 40 years after Gibson, Roy T. Fielding put it as follows [15]:

> *When I say hypertext, I mean the simultaneous presentation of information and controls such that the information becomes the affordance through which the user (or automaton) obtains choices and selects actions.*

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

Interestingly, on today's Web, the subjectiveness of affordances works in a publisher-driven way rather than a user-driven way. It is indeed the content publisher who decides what hyperlinks the user will have at his disposal. However, how can a publisher possibly know *all* the actions a user might want to perform? For example, if someone publishes a book review and adds a link to buy that book on Amazon, this will cover one possible action. But what if the user wants to read the book online, or borrow it from his local library? Although the *"read online"* action might reasonably be foreseen, *"borrow from local library"* is entirely dependent on the user and his current context, and can therefore not be determined beforehand. In general, just like with the pen that is used as a bottle opener, providers can never guess all actions users might want to perform.

The question gets even more intriguing when hypermedia drives the application state. Fielding explained that, in hypermedia APIs, the user advances the application state through controls in representations of resources. Consequently, every hypermedia representation must contain the controls leading to next steps [15]. For example, if a user is buying books, the representation of his virtual shopping cart must include links that allow him to order more or to proceed to the payment. Again, we can ask ourselves the same question: how can a publisher know what next steps the user might want to take? Sure, all possible steps *within* the application might be known, but what if the user wants to do something else, such as reading a sample chapter on the author's site, or asking social network friends which of two books they recommend?

In a Web for the 21st century, we can no longer confine application state to the boundaries of a single application. Instead, we should see application state on a Web scale, where the affordance provided by a piece of information is distributed across different Web applications. This paper therefore introduces a scalable approach to construct *distributed affordance* by generating personalized hypermedia controls that are obtained from several external sources. This transforms hypermedia affordance into a subjective experience, not imposed by the publisher, but created around the *user*.

This paper is structured as follows. In the next section, we discuss related work by looking at the broader hypermedia context and examining other solutions for personalizing affordances. Section 3 introduces the distributed affordance concept, and Section 4 describes its proposed architecture. Section 5 shows a proof-of-concept that brings distributed affordance to today's Web browsers. The possibilities and limitations of our solution are discussed in Section 6. Finally, we conclude and outline future work in Section 7.

2. RELATED WORK

2.1 The Web and Hypermedia

The invention of the World Wide Web [6] finally enabled hypermedia at a global scale. The requirements of such a large distributed system have been investigated by Fielding [16, 17], who derived the Representational State Transfer (REST) architectural style as a framework to develop and analyze distributed hypermedia systems. One of the constraints imposed by REST is *hypermedia as the engine of application state*, the notion that representations should contain the controls to advance the application state, *e.g.,* hyperlinks and forms that lead to subsequent steps. This constraint is essential to guarantee independent evolution of client and server and therefore long-term evolvability.

2.2 Web Intents

Today, the idea of dynamically adding affordances is mostly associated with *Web Intents* [9, 20]. Initially conceived by Paul Kinlan of Google, Web Intents aimed to become a Web version of the Intents system found on Android devices, where intents are defined as *"messages that allow Android components to request functionality from other components"* [37]. With Web Intents, Web applications can declaratively indicate their *intention* to offer a certain action, and websites can indicate they afford this action. For example, social media sites can indicate they enable the action "sharing", and a photo website can offer their users to "share" pictures. When a user initiates the *"share"* action on the website, Web Intents then allow him to share the photo through his preferred supported application.

While Web Intents' goals are similar to ours, there is a crucial difference in their architecture that severely limits their applicability. The benefit of Web Intents is that they are scalable in the number of *action providers*. Without Web Intents, publishers have to decide which action providers they support. For instance, the publisher of the photo website would have to decide which specific sharing applications it would offer its users. With Web Intents, the user can share photos through his preferred application, without the publisher having to offer a link to it. A major drawback of Web Intents is that they do *not* scale in the number of *actions*. Although the OpenIntents initiative allows to define custom actions [26], a publisher still has to decide which actions to include. In the photo website example, the publisher might opt to include a "share" action, but that is not useful if the user wants to order a poster print of a picture, download it to his tablet, or edit it in his favorite image application. While this strategy works on a device platform such as Android, where the set of possible actions is limited because of device constraints, such a closed-world assumption cannot hold on a Web scale. In November 2012, Web Intents support was removed from the Chrome browser [8]. Summarizing, we can say that Web Intents do not solve the core issue: a publisher still has to determine what affordances a user might need. The problem thus shifts from deciding which action providers to support to deciding which actions to support. Therefore, Web Intents only offer personalized affordance to a limited extent and do not sufficiently scale.

2.3 Social Interactions

A very prominent form of external affordances on today's Web are widgets that provide social interactions. We consider them external because they are commonly included in HTML representations as `script` or `iframe` tags whose source URL leads to an external domain, classifying them as *embedded link* hypermedia factors [4]. Examples include the Facebook *Like* button and the Twitter *Tweet* button [11]. Some of those widgets demonstrate personalized affordance: for instance, Facebook can personalize its button with pictures of the user's friends with links to their profiles. However, it is still the publisher who must decide what sharing applications to include. An additional issue is that different applications demand different metadata for optimal widget integration, which can make adding widgets costly [34].

Services such as AddThis [1] offer simple personalized access to different social networking sites. Publishers only have to include one external script to provide access to many different interaction providers. If the visitor has an AddThis account, he may indicate his preferred sharing applications, which are then shown on visited pages that include AddThis. We could see AddThis as a version of Web Intents with only one action, albeit a platform in which the supporting applications do not have to indicate their intention since AddThis indexes them centrally—and manually. Therefore, the principle is not scalable in the Web sense of the word.

2.4 Context-based Hyperlinks

Since the early days of the Web, we have seen context-based hyperlinks. One typical example are advertisements. On the one hand, on-demand hyperlinks from advertisers can appear in search results for certain queries. While these links are certainly personalized, they are not distributed because they are generated by the server, based on a user profile. In that sense, they are part of the regular application state flow, because they effectively point to next steps a user might want to take. The decision of which links to include is however determined by an application-specific user profile as opposed to an independent, cross-application preference indication. The collection of possible hyperlinks is also centralized—and closed, because of the financial aspects involved in advertising. On the other hand, advertisements can be embedded in publishers' websites, which slightly changes the situation. Still, they will not add affordance to the resource the user is viewing, but rather show links to related services that reflect in the first place the interest of the advertiser and are thus again publisher-driven.

Adaptive navigation support [13] is the corresponding topic within adaptive hypermedia [12], a research field that concerns the personalisation of the hypermedia experience. However, most adaptive navigation systems seem to work with recommendation suggestion rather than automatic link generation. Systems that do generate links mostly work on a closed set of documents and affordances. In several cases, the generated links do not specifically enhance the consulted resource, but merely act as an affordance of the resource's context instead of the resource itself, decreasing their relevance.

3. CONCEPT

3.1 Motivation

Every time we browse the Web, we take for granted the revolution HTTP hyperlinks have brought to information consumption. Nonetheless, the idea that you simply *click* on a piece of information to possibly retrieve a resource from the other side of the world started a groundbreaking revolution.

Like any good technological invention [5], we notice it only when it is *absent*: when we are on a Web page and want to perform an action for which we find no hyperlink. We identify three distinct possible causes that can lead to this:

1. **The affordance is present, but overlooked.**
   This is where Gibson's [18] affordances differ from Norman's [25]. Whereas Gibson considers *all* action possibilities, even those (seemingly) inaccessible for a subject, Norman focuses on *perceived* affordance.

2. **The affordance is present with another provider.**
   This happens when the affordance leads to an action that addresses the user's intention but not his preferred handling. For instance, a site offers the "share" action with Facebook but not with Twitter.

3. **The affordance is not present.**
   The user has to fall back to other mechanisms, since the desired action cannot be completed with hypermedia. This is caused by a mismatch between the publisher's expectations and the user's actual intentions.

Each of the above three causes involves the following actor groups. The *publisher* offers a representation of a resource and its associated affordance. The *user* consumes this representation and depends on the affordance therein to perform subsequent actions. The *provider* offers the actions desired by the user; this actor can either be the publisher itself or a third party.

The first of the three causes is the result of the user's subjective perception and therefore not a technical but a usability-related issue, which the publisher can solve with various strategies [27]. The second and third causes concern objectively missing affordances and therefore demand a different category of solutions.

For the second cause, where the publisher supplies the actions but not the desired provider, solutions such as Web Intents exist, as discussed in the previous section. The explicit assumption of these solutions is that a publisher cannot foresee all providers for all possible actions, which is correct. However, their implicit assumption is that a publisher *can* foresee all *actions* a user might want to perform, which is equally unrealistic on a Web scale as foreseeing all providers. Therefore, the second cause is actually a corner case of the third, especially if we consider a desired action as the combination of an intention and a *specific* provider.

Finally, for the third cause, and for the portion of the second cause where existing methods fall short, there are presently no solutions, which is the main motivation behind this paper. In essence, the issue arises because current hypermedia systems maintain a closed-world assumption [30]. Since none of the common definitions of hypertext state that *all* possible next steps should be contained in a hypertext document [14, 15, 24] (which is impossible anyway with such a multitude of users), the notion of hypermedia as "the engine of application state" is limited to those situations where the publisher has correctly predicted the actions a user (or machine client) might want to take. Unfortunately, more often than not, publishers have specific use cases in mind and will only supply affordance they consider relevant. In that sense, "hypermedia as the engine of application state" remains confined to the borders of a single application—unless we find a way to realize an open-world assumption for hypermedia.

3.2 Observations

In order to devise a solution, we will look at current user behavior when desired affordance is missing in the hypermedia representation. The flexibility of the term "affordance" becomes apparent when we see that users are still able to realize their intention by achieving the action through other means. For example, if a user wants to share a photo on his preferred social website, but a hyperlink is missing, he can use the browser's affordances and type the website's URL in the address bar and manually share the photo. The same applies to other actions such as ordering a print and downloading the photo to a tablet. This reminds us of pre-hypermedia times, before information became an affordance. It is also not unlike machine clients that are consuming a non-hypermedia driven API and, for lack of affordance in the representation, are forced to know themselves how to advance the application state.

This does not mean the representation lacks the necessary *information* to proceed to the action. It does, however, mean that the *affordance* for this action does not reside in the representation itself, and has to be created manually from non-actionable information in that representation and out-of-band knowledge about action providers. For example, a user reading a book review might want to perform actions such as buying the e-book version or borrowing it from a local library. If the controls that afford buying are not present in the review page, the user might copy the book's title from the review text (non-actionable information), paste it into a search engine, and then find the relevant search result that does afford buying (knowledge about action providers). However, such an approach that relies on manual creation of external affordance is far less efficient than affordance resulting from the "simultaneous presentation of information and controls" [15].

3.3 Distributed Affordance

Based on the observation that representations often contain the necessary information, albeit in non-actionable form, and the fact that knowledge about action providers is available, we introduce the concept of *distributed affordance*. The idea is to dynamically create affordance based on the information already present in the representation, with knowledge from distributed sources. This will augment the affordance of the representation with controls that directly relate to the representation itself, instead of merely to its context. Furthermore, we do not need to assume the publisher knows the desired actions the user wants to perform or the providers he prefers, as the only data needed from the publisher is information about the representation itself. Based on the user's profile, we construct the most relevant affordance, depending on his preferences and current browsing context. For example, for the user reading the book review, hyperlinks to the e-book version and the user's local library could be inserted automatically as distributed affordance.

The technical challenges are, firstly, to extract the non-actionable information from the representation and secondly, to organize the knowledge about actions offered by providers. Thirdly, we need to capture a user's preferences to, fourthly, combine the non-actionable information and the knowledge about action providers into possible actions. Fifthly, we must integrate affordances for these actions into the original representation. In the next section, we introduce an architecture that supports the creation of distributed affordance, starting with the components that realize these five functions.

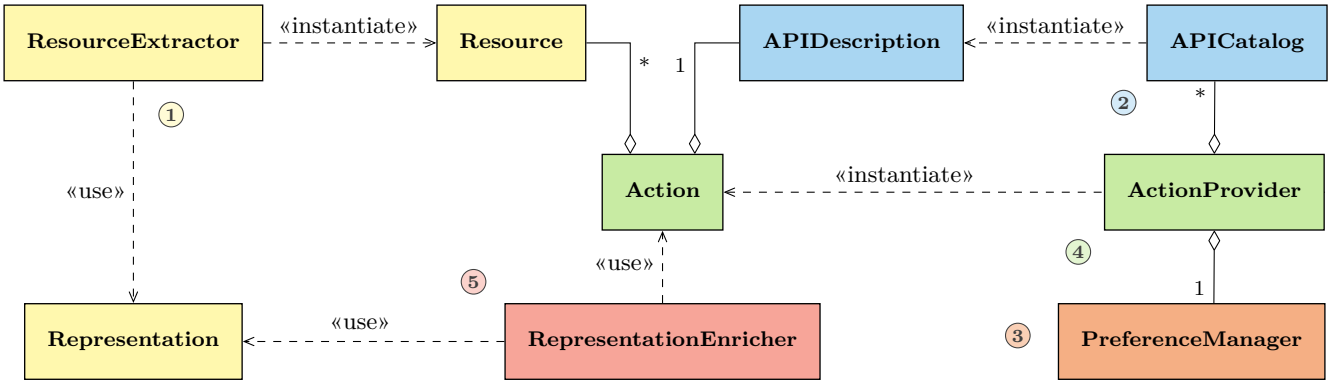


Figure 1: The resources inside a representation are extracted (1) and combined with API descriptions (2), based on the user's preferences (3), into actions (4), for which affordances are added to the representation (5).

4. ARCHITECTURE

4.1 Components

The model architecture consists of five groups of components, as displayed in Figure 1, which we discuss in more detail below.

(1) **Information extraction** Given a resource with non-actionable information items, a `ResourceExtractor` extracts those items as resources with properties. These resources are structured as key/value pairs, for example as JSON [31], or more formally, as RDF triples [21]. The benefit of the latter is that the properties are URIs, which gives them universal meaning across different applications. `ResourceExtractor` itself is only an interface, as many implementations are possible. For example, for HTML representations, we can define implementations that extract resources from RDFa [2] or HTML5 microdata [38]. For (partly) textual representations, extractors could for instance use named-entity recognition techniques [23]. In any case, the possible extractors depend on the media types of the available representations.

(2) **Provider knowledge** The knowledge about actions offered by a provider can be generalized in API descriptions. The information in these descriptions should be structured in such a way that, given certain resource properties, it is simple to decide which APIs support actions on that resource. Different `APICatalog` implementations can support different API description methods, such as ReLL [3] or RESTdesc [35]. It is important to note that the resource extraction method is independent from the API description method, *i.e.,* API catalogs allow to search for APIs based on resources and their relationships, regardless of how these resources were extracted. This prevents coupling between information description and API description, allowing instead to explore all combinations thereof, which results in a more valuable affordance for the user.

(3) **User preferences** A `PreferenceManager` keeps track of a user's preferences and thereby acts as a kind of filter on the `APICatalog`, typically selecting only certain APIs and sorting them according to appropriateness for the user. Various models are possible, such as explicitly asking a user to indicate his preferences or learning from previously used affordances. In simpler implementations of the architecture, the role of the `PreferenceManager` can be taken care of by the `APICatalog`, which then only includes API descriptions that match the user's preferences.

(4) **Action generation** Based on a user's preferences, an `ActionProvider` instantiates possible actions, which are the application of a certain API on a specific set of resources. Thereby, every action is associated with one or more resources inside the representation, which makes the actions specific for the representation instead of merely related to its context.

(5) **Affordance integration** A final category of components are `RepresentationEnricher` implementations, which add affordances for the generated possible actions to a hypermedia representation that is sent to the user. Through these affordances, the user can chose and execute the desired actions directly. Implementations depend on the media type of the desired representation as they need to augment its affordance in a specific way. Technically speaking, this media type could be different from the media type whose non-actionable information was extracted, *e.g.,* resources could have been extracted from a JSON representation while the result is presented as HTML.

4.2 Deployment

Two main deployment strategies exist: the components can be deployed inside the client or offered as a service.

Client-based The architecture that provides the distributed affordance can be implemented directly in a hypermedia client, or as a plugin thereof. In the common case of a Web browser, it can be programmed as a browser extension. The benefit is that some of the client's functional blocks can be reused, such as the representation parser. When the client requests a representation, the extractor will be triggered to find resources therein, which will prompt the action provider to combine these resources with relevant API descriptions into actions. Affordances for these actions can then be added in the interface. In graphical clients, they could become part of the user interface or the hypermedia browsing space. For machine clients, they are added to the existing affordance set.

Affordance as a service The drawback of the above approach is that users need a supporting client to profit from the augmented affordance. This assumption can lead to a bootstrapping problem. Therefore, the architecture can also be offered as a service, exposing a hypermedia API with distributed affordance as resources. Meant as a transitional measure, These resources can be included as embedded links in representations [4] to augment them with dynamically generated affordance. This strategy is thus able to leverage distributed affordance without explicit client support.

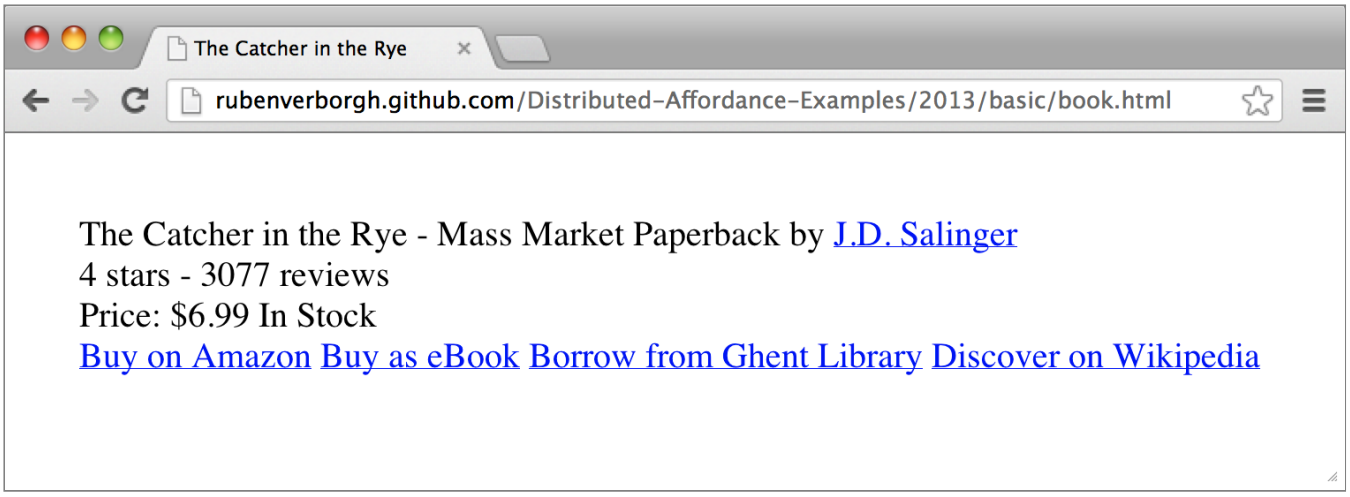


Figure 2: The links at the bottom are automatically generated affordance, resulting from the application of API descriptions to machine-interpretable data extracted from the representation's HTML source.

5. PROOF OF CONCEPT

5.1 Functionality

We have developed a proof of concept platform that implements distributed affordance as a service, allowing to demonstrate the concept without a specific browser extension. This platform, called *Vyperlinks*, allows straightforward integration of distributed affordance into existing representations. Vyperlinks is a hypermedia API that serves distributed affordance resources through the following URI template [19]:

```
http://vyperlinks.org/actions{?for}
```

Herein, the `for` parameter should become a URI with a fragment identifier [7] that points to the resource within a document for which to generate distributed affordance. The presence of the fragment is important, because representations might include different resources that each have different affordances. The resolving strategy differs per media type; for HTML, the fragment corresponds to the element with the same identifier. An instantiation of this template points to an *actions resource* that contains the distributed affordance for the specified resource.

An HTML document can indicate a placeholder for affordances with `<div class="affordances" data-for="id">`, where `id` is the identifier of the element for which affordances should be generated. If the browser does *not* support distributed affordance (neither natively nor with an extension), a so-called *shim* script will take over and request the affordances through Vyperlinks using an `iframe`. The `iframe`'s same-origin policy guarantees the privacy of the user's preferences and maintains the trust of the provided affordance.

5.2 Implementation

When the Vyperlinks server receives a request for an actions resource, it first fetches the resource with the specified URL, without the fragment identifier. If the request successfully returns a representation, the resources it contains are extracted. Then, they are offered to an action provider that generates the actions according to the user's preferences, and are subsequently filtered for those actions that act upon the resource identified by the fragment. Finally, affordances for these actions are represented in a hypermedia format, which is then sent back to the user.

The user can be identified on the Vyperlinks platform with existing mechanisms [29, 32]. This concept is similar to those discussed in Section 2.3 and distinct from the identification mechanism of the application that embeds the affordance. Users that do not have a Vyperlinks account or that are not authenticated receive best-effort personalizations, for instance, based on their language or location. Even without personalization, the affordance remains distributed and relevant, *i.e.,* it is constructed from distributed sources and specifically generated for the specified resource.

5.3 Technologies

Currently, the Vyperlinks server supports HTML representations and HTML5 microdata [38]. The current API description model straightforwardly matches a resource's type and associated properties. We plan to extend Vyperlinks with more extractors and description models, as well as with advanced user preference management, as detailed in Section 7.

5.4 Examples

We have created two examples that demonstrate the integration of distributed affordance in annotated HTML documents,[1] which we will discuss below. When studying these examples, it is convenient to look at both the rendered page in the browser and the HTML source code.

***Book* example** This example, shown in Figure 2, has been adapted from the Schema.org website [10] and features an HTML representation of the book "The Catcher in the Rye" marked up with HTML5 microdata in the Schema.org vocabulary. The corresponding resource has been given the identifier `book`. To this representation, we added a placeholder for affordances relating to the `book` resource. The Vyperlinks platform extracts the properties of the book (such as author and title) and uses them to generate hyperlinks that afford buying and borrowing, among other actions. Rather than merely link to the action providers' start page, the hyperlinks

[1] Available at `http://rubenverborgh.github.com/Distributed-Affordance-Examples/`

directly point to the page about the specific action on the specific resource, *i.e.,* at the furthest possible point in the application state, which is the point to where a manually inserted link would also lead.

***Publications* example** This example shows a scientific publication list that has been annotated with Schema.org markup. Every entry in the list has its own identifier. In contrast with the previous example, multiple affordance placeholders have been inserted, one for each entry. This demonstrates the necessity of dealing with multiple resources in a representation through fragment identifiers, and the fact that affordances can be anchored in different places of the hypermedia document. At the same time, it indicates the capability of the Vyperlinks application to timely serve many actions resources.

6. DISCUSSION

6.1 Enabling Factors

An important difference between the proposed distributed affordance technique and other solutions that require their own annotation mechanisms [34] is that distributed affordance aims to reuse existing annotations by providing different extractors. The goal is that no specific modifications should be necessary to allow users to benefit from distributed affordance. However, we do require some form of annotations to be available, so we have to trust there will be sufficient incentives for publishers to mark up data, such as better indexing by search engines or increased social media exposure. If not, we can always fall back to content analysis techniques, but they will be more error-prone.

Additionally, we rely on markup of provider services, because we need machine-processable API descriptions to generate affordances for these providers. Fortunately, these descriptions can be created by third parties instead of solely by the providers themselves, as indicated by the examples of Section 5.4, where we created the provider descriptions ourselves. The architecture can support multiple API sources, so complex descriptions in different formats are possible.

6.2 Performance

Affordance should be at the user's disposition as soon as possible, since the user depends on this affordance to perform actions. Furthermore, in traditional hypermedia, the affordance is served synchronously with the information, so only minimal delays are tolerable. Therefore, the performance of distributed affordance platforms is crucial and several aspects require special attention.

The critical path of the current affordance-as-a-service implementation is as follows:

1. fetch the provider's representation;
2. extract resources from the representation;
3. generate actions from the resources;
4. represent the actions;
5. send the representation to the user.

Therefore, an increase or decrease in time on any of the above steps directly influences the total time of each affordance creation. However, the process can be implemented with streams as a pipe-and-filter architecture, *e.g.,* resource extraction can start when the representation has only partially been retrieved and action representation can start as soon as one action becomes available. Nonetheless, the duration of every step must be kept as short as possible.

The current Vyperlinks platform implements several performance optimizations. One of them is caching: the provider's representation is only fetched when needed, such that repeated requests are not necessary if one representation contains multiple distributed affordance containers. Furthermore, Vyperlinks responses are also cacheable on a per-user basis. Another optimization strategy is to generate actions as fast as possible by using proxy URLs. This is necessary because many providers do not directly expose a link to the desired application state. For instance, an online book vendor might have a search result resource that can be accessed directly, but one has to download this result page to obtain a direct link to the desired book page. Instead of fetching such intermediary pages when the affordance is generated, the platform returns a proxy URL, which is dereferenced only when used. Additional improvements could involve client scripting, for instance, to avoid multiple requests when a page contains different affordance containers, and to speed up dereferencing by pre-loading proxy URLs.

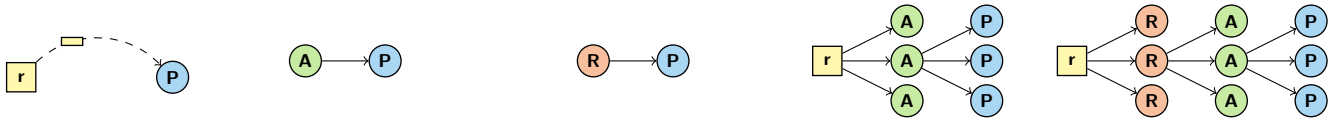
6.3 Scalability

As usual with concepts proposed for the Web, the question is: *will it scale?* We will therefore look at the key facets that make distributed affordance scalable. First, we have to note that the principle of composing affordance from different sources is created to make affordance scalable, *i.e.,* to avoid unnecessary coupling between a client and a server. While the *hypermedia as the engine of application state* constraint improves the independent evolution of client and server—because the client dynamically discovers what next steps it can take—it only works in as far as the server can correctly predict the next steps the client is interested in. Thereby, conversational coupling [28] is traded for *affordance coupling*, which is better, but suboptimal. Distributed affordance removes this coupling by letting the client choose one or more affordance providers.

Second, we must investigate whether the proposed architecture is scalable itself. In the case of a client-based deployment, every client has its own distributed affordance component, and therefore scales with the number of clients. In the case of a service-based deployment, the REST style can be used (as is the case with Vyperlinks), because the statelessness and cacheability constraints facilitate replication across different servers. The proposed architecture does not have many inter-component dependencies, so it is straightforward to distribute components across multiple servers.

6.4 Openness

A final discussion facet is the openness of the distributed affordance concept, because it would be contradictory to offer distributed affordance from a single centralized system. It is therefore crucial to understand that Vyperlinks is *one* implementation of distributed affordance and not *the* implementation. This is why we encourage to use the declarative affordance placeholder, which can be supported by the browser natively or through an extension, or with a JavaScript shim (similar to the Web Intents mechanism [20]). That way, users can indicate their preference for a particular distributed affordance provider.



(a) missing affordance (b) API descriptions (c) functional API descriptions (d) Web Intents (e) distributed affordance

Figure 3: A comparison of distributed affordance and related technologies shows how they differ in the involvement of representations r, resources R, actions A, and action providers P.

Another aspect of openness is the question whether publishers will endorse distributed affordance. After all, it might be in the publisher's interest that users only have a limited affordance at their disposal, for instance, for commercial reasons. We can therefore imagine that publishers might like to exclude certain actions from the generated affordance. These commercial considerations should be balanced with the desire to have an open ecosystem.

7. CONCLUSION AND FUTURE WORK

In this paper, we have observed the discrepancy between affordance publishers and consumers, and offered a solution and corresponding architecture for distributed affordance. Based on a user's preferences, affordance is generated dynamically from distributed providers using non-actionable information inside representations. This information does not target a specific API or action, but rather consists of high-level machine-interpretable properties about the resource. In contrast with very specific interfaces, this *semantic* interface enables *serendipitous reuse* [36] of the resource by many different APIs. The task of a distributed affordance platform is to create affordances to the subset of those APIs that is relevant for the user. Such a platform can either be integrated in a client or offered as a service.

In order to position distributed affordance in the broader Web API ecosystem, we schematically compare it to related technologies in Figure 3. We will revisit the example of a publisher that offers a Web page with a book review. First, the important thing to note is that an affordance is not strictly necessary for the user to perform the desired action, such as buying or borrowing the book. If the affordance for the desired action is missing [Fig. 3a], the user can always manually extract the needed information from the representation to carry out the action with a provider. For example, the user could copy the title of the book and paste it into the search engine of an online book store. This indicates the difference between an affordance and an *enabler*. However, a direct affordance is much more convenient. Second, API descriptions might indeed lead to a provider, but their starting point is different. API descriptions without a functional discovery mechanism describe how to access the action with a specific provider [Fig. 3b], such as the required API interactions to buy a book. Functional API descriptions [35] describe in what ways providers can act on a resource [Fig. 3c], for instance, the fact that the book can be bought through one provider and borrowed through another. Third, Web Intents [20] are able to suggest providers for actions indicated in a representation [Fig. 3d]. In that sense, Web Intents are non-functional service descriptions that explain how to invoke a particular action with different providers, offered through a convenient but single-purpose HTML interface. Finally, distributed affordance allows dynamic discovery of possible actions based on the extracted resource properties, offering the highest degree of freedom for affordance generation [Fig. 3e].

We conclude that distributed affordance takes an open-world assumption with regard to the possible actions a user can select, as opposed to current hypermedia formats that require all affordances to be contained within the representation. Not only does this align better with the Web philosophy, it is a step towards a personalized hypermedia experience. The timing of this solution is not coincidental: it is only now that resources become augmented with sufficient machine-interpretable information [22], which is vital for automated affordance generation. Furthermore, the shift towards semantic description of REST APIs, necessary to discover matching actions, is also fairly recent [33]. These two evolutions enable distributed affordance, leading to a Web in which not only the information is open, but also the affordances each piece of information contains.

Many aspects of distributed affordance are of interest for future research. One important task is to design techniques for user modelling [12], solving problems such as determining which affordances are relevant for a user, and how these can be selected and ranked in real-time. We should determine where and how the user's preferences are stored, and how we can guarantee safety and privacy. A related question is the discovery and storage of API descriptions. API providers must be able to publish descriptions in order to be found by users, who can then incorporate these APIs in their preferences. Furthermore, publishers with commercial interests might want to include only certain features, excluding affordances that lead to their competitors, so a filter could be necessary.

In the future, we want to establish the Vyperlinks platform as a leading example of the possibilities of distributed affordance. We also plan to integrate the platform directly into Web browsers. While the implementation of this paper focuses on HTML representations, machine clients can also benefit from distributed affordance in many ways. However, supplying affordance for such clients comes with new challenges, such as extending the concept of preferences to machines. A current bottleneck for the platform is the limited availability of machine-interpretable API descriptions. Therefore, we will explore the automated integration of existing Web APIs. After all, the more APIs are supported, the better the personalized distributed affordance experience becomes. Finally, we aim to investigate the standardization of distributed affordance technology, with the ultimate goal of making it widely available on the Web. On the website `http://distributedaffordance.org/`, you can follow our ongoing research in this area.

8. ACKNOWLEDGMENTS

The described research activities were funded by Ghent University, the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research Flanders (FWO Flanders), and the European Union.

9. REFERENCES

[1] AddThis, http://www.addthis.com/

[2] Adida, B., Birbeck, M., McCarron, S., Herman, I.: RDFa Core 1.1. W3C Recommendation (Jun 2012), http://www.w3.org/TR/2012/REC-rdfa-core-20120607/

[3] Alarcón, R., Wilde, E.: Linking Data from RESTful Services. In: Third Workshop on Linked Data on the Web (2010), http://events.linkeddata.org/ldow2010/papers/ldow2010_paper10.pdf

[4] Amundsen, M.: Hypermedia types. In: Wilde, E., Pautasso, C. (eds.) REST: From Research to Practice, pp. 93–116. Springer, New York (2011), http://dx.doi.org/10.1007/978-1-4419-8303-9_4

[5] Bergman, E.: Making technology invisible: A conversation with Don Norman. Information Appliances and Beyond: Interaction Design for Consumer Products pp. 10–26 (2000)

[6] Berners-Lee, T., Cailliau, R., Groff, J.F.: The world-wide web. Computer Networks and ISDN Systems 25(4–5), 454–459 (1992), http://www.sciencedirect.com/science/article/pii/0169755292900395

[7] Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifier (URI): Generic syntax. IETF Standards Track (Jan 2005), http://tools.ietf.org/rfc/rfc3986

[8] Billock, G.: Status of Web Intents in Chrome (2012), http://lists.w3.org/Archives/Public/public-web-intents/2012Nov/0000.html

[9] Billock, G., Hawkins, J., Kinlan, P.: Web Intents. W3C working draft (Jun 2012), http://www.w3.org/TR/web-intents/

[10] Bing, Google, Yahoo!, Yandex: Schema.org. Specification (Jun 2011), http://schema.org/docs/schemas.html

[11] Bodle, R.: Regimes of sharing. Information, Communication and Society 14(3), 320–337 (Apr 2011)

[12] Brusilovsky, P.: Adaptive hypermedia. User Modeling and User-Adapted Interaction 11(1–2), 87–110 (2001), http://dx.doi.org/10.1023/A%3A1011143116306

[13] Brusilovsky, P.: Adaptive navigation support. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web, pp. 263–290. Springer-Verlag (2007), http://dl.acm.org/citation.cfm?id=1768197.1768207

[14] Conklin, J.: Hypertext: An introduction and survey. Computer 20(9), 17–41 (Sep 1987)

[15] Fielding, R.T.: REST APIs must be hypertext-driven. Untangled – Musings of Roy T. Fielding (Oct 2008), http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[16] Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. Transactions on Internet Technology 2(2), 115–150 (May 2002)

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

[18] Gibson, J.J.: The theory of affordances. In: Shaw, R., Bransford, J. (eds.) Perceiving, Acting, and Knowing: Toward an Ecological Psychology. Lawrence Erlbaum, New Jersey (1977)

[19] Gregorio, J., Fielding, R., Hadley, M., Notthingham, M., Orchard, D.: URI template. IETF Proposed Standard (Mar 2012), http://tools.ietf.org/rfc/rfc6570

[20] Kinlan, P.: Web Intents (2010–2013), http://webintents.org/

[21] Klyne, G., Carrol, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[22] Mühleisen, H., Bizer, C.: Web Data Commons – Extracting structured data from two large Web corpora. In: Proceedings of the 5th Workshop on Linked Data on the Web. vol. 937 (Apr 2012), http://ceur-ws.org/Vol-937/ldow2012-inv-paper-2.pdf

[23] Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Lingvisticæ Investigationes 30(1), 3–26 (Jan 2007), http://dx.doi.org/10.1075/li.30.1.03nad

[24] Nelson, T.: Dream machines. self-published (1978)

[25] Norman, D.A.: The Design of Everyday Things. Doubleday, New York (1988)

[26] OpenIntents, http://www.openintents.org/

[27] Palmer, J.W.: Web site usability, design, and performance metrics. Information Systems Research 13(2), 151–167 (Jun 2002), http://dx.doi.org/10.1287/isre.13.2.151.88

[28] Pautasso, C., Wilde, E.: Why is the Web loosely coupled? – A multi-faceted metric for service design. In: Proceedings of the 18th international conference on World Wide Web. pp. 911–920. ACM, New York (2009), http://doi.acm.org/10.1145/1526709.1526832

[29] Recordon, D., Reed, D.: OpenID 2.0: a platform for user-centric identity management. In: Proceedings of the second ACM workshop on Digital identity management. pp. 11–16. ACM, New York (2006), http://doi.acm.org/10.1145/1179529.1179532

[30] Reiter, R.: On closed world data bases. In: Logic and Data Bases. pp. 55–76 (1977)

[31] Severance, C.: Discovering JavaScript Object Notation. Computer 45(4), 6–8 (Apr 2012), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6178118

[32] Sporny, M., Inkster, T., Story, H., Harbulot, B., Bachmann-Gmür, R.: Web identification and discovery. W3C Editor's Draft (Dec 2011), http://www.w3.org/2005/Incubator/webid/spec/

[33] Verborgh, R., Harth, A., Maleshkova, M., Stadtmüller, S., Steiner, T., Taheriyan, M., Van de Walle, R.: Semantic description of RESTAPIs. In: Pautasso, C., Wilde, E., Alarcón, R. (eds.) REST: Advanced Research Topics and Practical Applications (2013)

[34] Verborgh, R., Mannens, E., Van de Walle, R.: The rise of the Web for Agents. In: Proceedings of the First International Conference on Building and Exploring Web Based Environments. pp. 69–74 (Jan 2013), http://thinkmind.org/download.php?articleid=web_2013_3_30_40070

[35] Verborgh, R., Steiner, T., Van Deursen, D., Coppens, S., Gabarró Vallés, J., Van de Walle, R.: Functional descriptions as the bridge between hypermedia APIs and the Semantic Web. In: Proceedings of the Third International Workshop on RESTful Design. pp. 33–40. ACM (Apr 2012), http://www.ws-rest.org/2012/proc/a5-9-verborgh.pdf

[36] Vinoski, S.: Serendipitous reuse. Internet Computing 12(1), 84–87 (2008), http://steve.vinoski.net/pdf/IEEE-Serendipitous_Reuse.pdf

[37] Vogel, L.: Android Intents – tutorial (2009–2013), http://www.vogella.com/articles/AndroidIntent/article.html

[38] Web Hypertext Application Technology Working Group: HTML – microdata, http://www.whatwg.org/specs/web-apps/current-work/multipage/microdata.html