Tagging-Aware Portlets

Oscar Díaz, Sandy Pérez, and Cristóbal Arellano

ONEKIN Research Group, University of the Basque Country,
San Sebastián, Spain
{oscar.diaz,sandy.perez,cristobal-arellano}@ehu.es
http://www.onekin.org

Abstract. A corporate portal supports a community of users on cohesively managing a shared set of resources. Such management should also include social tagging, i.e. the practice of collaboratively creating and managing tags to annotate and categorize content. This task involves to know both *what* to tag (hence, the rendering of the resource content) and *how* to tag (i.e. the tagging functionality itself). Traditionally both efforts are accomplished by the same application (*Flickr* is a case in point). However, portals decouple these endeavours. Tagging functionality is up to the portal, but content rendering can be outsourced to third-party applications: the portlets. Portlets are Web applications that *transparently* render their markup through a portal. The portal is a mere conduit for the portlet markup, being unaware of what this markup conveys. This work addresses how to make portlets tagging-aware, i.e. portlets that can be seamlessly plugged into the portal tagging infrastructure. The main challenge rests on consistency at both the back-end (i.e. use of a common structure for tagging data, e.g. a common set of tags), and the front-end (i.e. tagging interactions to be achieved seamlessly across the portal using similar rendering guidelines). Portlet events and *RDFa* annotations are used to meet this requirement. A running example in *WebSynergy* illustrates the feasibility of the approach.

Keywords: tagging, portlets, Web portals, RDFa, WSRP, Liferay.

1 Introduction

Corporate portals play a three-fold role. As a means by which to manage and access content, portals play the role of content managers. As the mechanism to integrate third party applications using portlets or gadgets, portals can be regarded as front-end integrators. Finally, portals also offer a conduit for on-line communities. It is in this third role where the importance of incorporating social networking facilities in current portal engines emerges. Portals, to a bigger extent than other Web applications, have the notion of community deeply rooted inside its nature. Specifically, corporate portals are borne to support the employees within an organization. Therefore, it is just natural to integrate social networking into these portals (hereafter referred to as just "portals"). Among social networking activities, this paper focuses on social tagging.

Social tagging brings sociality into tagging. Attaching labels to resources is no longer a secluded activity, but seeking or tagging resources is achieved based on previous inputs from the community, and this activity, in turn, serves to build the community itself by clustering users based on tag preferences.

Traditional tagging sites such as *Delicious, Youtube* or *Flickr* can be characterized as being *self-sufficient* and *self-centered*. The former implies that all it is need for tagging (i.e. the description of the resource, the tag and the user) is kept within the tagging site. *Delicious* keeps the bookmark URL, the tags and the user community as assets of the site. On the other hand, self-centeredness indicates that all *Delicious* care about is its own resources, tags and users. No links exists with other tagging sites, even if they tag the same resources (e.g. *CiteULike*).

This situation changes when moving to a portal setting. A hallmark of portals is integration. Rather than providing its own services, a portal is also a conduit for external applications. So offered applications are technically known as *portlets* [3]. Portlets can be locally deployed or be provided remotely through third-party providers. For instance, a portal can offer the possibility of blogging, purchasing a book, or arranging a trip, all without leaving the portal. Some of these portlets can be built in house whereas others can be externally provided by third parties (e.g. *Amazon* or *Expedia*). The portal mission is to offer a common gateway that hides such distinct origins from the user. This has important implications on the way tagging can be incorporated into portals, namely:

– portals are not self-sufficient. Taggers (i.e. the portal community) and tags are portal assets. However, and unlike self-sufficient tagging sites, portals could not hold the description of all tag-able resources. For instance, the description of the books or hotels offered through the portal could be remotely kept by e.g. *Amazon* and *Expedia*, respectively. This outsource of content description does not imply that the external resources are not worth tagging. This leads to distinguish between two actors: *the resource provider*, which keeps the content of the tag-able resources (e.g. book description in the case of *Amazon*), and *the resource consumer* (i.e. the portal), which holds the tagger community and the tags,
– portals are not self-centered. Traditional tagging sites are "tagging islands": each site keeps its own tagging data. Providing an integrated view of these heterogeneous tagging silos is at the user expenses. By contrast, portals strive to glue together heterogeneous applications. This effort implies offering a consistent set of tags no matter the resource nor the portlet through which the tagging is achieved. That is, our expectation is that employees would use a similar set of tags no matter the portlet that holds the tagged resource.

Based on these observations, consistency is identified as a main requirement, i.e. tagging should be seamlessly achieved across the portal, regardless of the type (messages, books, hotels, etc), or origin (i.e. *Amazon*, *Expedia*, etc) of the resource. This consistency is two-fold. *"Back-end consistency"* implies the use of a common structure for tagging data, e.g. a common set of tags. On the other hand, *"front-end consistency"* entails tagging interactions to be achieved seamlessly and cohesively across the portal using similar rendering recourses and aesthetic guidelines.

To this end, we present a novel architecture to support tagging capabilities as a portal commodity. This implies that portlets should be engineered to be plugged into this commodity rather than building their own tagging functionality. In the same way, that portlets adapt their rendering to the aesthetic guidelines of the hosting portal, tagging through portlets should also cater for the peculiarities of the consumer portal. The paper presents how these ideas have been borne out for the *WebSynergy* portal engine, using *RDFa* annotations and a publish/subscribe mechanism. The running example is available for consultation at `http://tagging.onekin.org/`.

The rest of the paper is organized as follows. Section 2 reviews tagging in current portal engines. Section 3 provides a brief on portlets. The different actors that interact during tagging are introduced in Section 4. Back-end consistency and front-end consistency are the topics of Sections 5 and 6, respectively. A revision and some conclusions end the paper.

2 Tagging in Current Portal Engines

Motivation for bringing tagging at the working place admits a two-fold perspective. From the company's viewpoint, tagging is an affordable approach to account for knowledge sharing and retention in the context of an organization, preventing leaking critical data outside the company [4]. From an employee's perspective, distinct studies [10,2,7] conclude that individual motivations for tagging at the working place, such as creating a personal repository or re-finding one's own resources, remained important. Additionally, tagging is also a means for professional promotion, community construction and information dissemination. Indeed, tags can serve distinct purposes: identifying what the resource is about (e.g. descriptive tags such as *"ajax"*), identifying what it is (e.g. *"book"*); identifying who owns it (e.g. *"peter"*); refining categories (e.g. *"beginning"*); identifying qualities (e.g. *"interesting"*), self reference (e.g. *"myfavorite"*) or task organizing (*"toRead"*, *"forDevelProject"*) [5]. Some of these purposes really turn fully useful in a working context. Tagging a resource as *"forDevelProject"* makes *social* sense if there are other users that share the same project. Insights suggest that "people need concepts and words in common in order to engage in collective actions. Tagging services in general appear to offer a means for achieving such common ground. Tagging services within a work-oriented enterprise would seem to be a particularly promising setting for people to engage in the co-construction of their common understandings" [8]. Tagging emerges as a main opportunity not only for resource self-organization but also for community construction.

Portal engine vendors are well-aware of this fact, and include tagging among their offerings. Tagging can come in different flavours: (1) tagging as part of an external application (i.e. a portlet) offered by the portal, (2) tagging as a portal functionality offered as a separated service, and (3), tagging as a portal commodity which it is not intended to work on its own but for other services to plug into. All three approaches can co-exist in the same portal. The difference stems from who is the owner of the tagging data, and what is the scope of tagging (see Table 1).

Tagging as part of an integrated application. This is the trivial case where the application being integrated (e.g. an external portlet) offers tagging capabilities on its own. Both tagged resources and tagging data are kept by the portlet. The only difference with traditional tagging sites such as *Flickr*, is that now tagging is achieved *through* the portal but not *by* the portal.

Tagging as a portal functionality. This is the approach currently supported by most vendors. The portal is regarded as a content manager. The portal owns the resources,

Table 1. Tagging through portals

Tagging as ...	Tagging Data Owner	Tagging Scope
...part of an external application	application	resources of the application
... portal functionality	portal	resources *kept by* the portal
... portal commodity	portal	resources *offered through* the portal

and provides functionality for tagging. Tagging is restricted to those resources within the realm of the portal. *Liferay* illustrates this approach. *Liferay* allows users to tag web content, documents, message board threads, wiki article, blog posts, etc. Users can then capitalize on their tagging efforts through two *Liferay* applications (delivered as portlets): the *TagsAdmin* portlet and the *AssetPublisher* portlet. The former permits tag addition and organization into categories. The second portlet creates tag-based views out of tag-based queries. Notice however, that all tag-able content should be within the realm of the *Liferay* portal.

Tagging as a portal commodity. Rather than as content managers, portals are now envisaged as integration platforms. As such, portals offer commodities for easing the integration of heterogeneous applications (e.g. the *Single Sign-On* commodity is a popular example[1]). Likewise, we advocate for tagging services to be offered as a commodity. Portlets then plug into this commodity. This is, tagging services are up to the portal but offered through the companion portlets. This insight, its grounds and feasibility, make up the contribution of this work.

3 A Brief on Portlets

Traditional data-oriented Web Services facilitate the sharing of the business logic, but suggest that Web service consumers should write a presentation layer on top of this business logic. By contrast, presentation-oriented Web Services (a.k.a. *portlets*) do deliver markup (a.k.a. *fragments*) rather than a data-based XML document. A fragment is then a piece of XHTML or other markup language, compliant with certain rules that permit a fragment to be aggregated with other portlets' fragments to build up a complete portal page. The aim is for full-fledged applications (not just functions) to provide both business logic and presentation logic as a Web component, easily pluggable into distinct Web applications (e.g. portals).

Portlets can be either local or remote to the hosting portal. This brings component-based development to the portal realm where different teams can be in charge of supporting a given functionality (supported as a portlet) that is later deployed locally at the portal. However, where the notion of portlet gets its full potential is when portlets are deployed remotely, provided by third parties. This scenario requires portlet interoperability, and here is when the *Web Services for Remote Portlets* (WSRP) specification [9] come into play. WSRP specifies a protocol and a set of interfaces that allows portals to consume and publish portlets as Web Services.

[1] This commodity enables a user to log in once at the portal, and gain access to the available applications being offered through the portal without being prompted to log in again.

Additionally, the Java Portlet Specification [6] follows a model of separating concerns in different lifecycle methods, like *processAction*, *processEvent*, *render*. This provides a clean separation of the action semantics from the rendering of the content. During the process action, the portlet can change its state. Next, during the rendering phase, the portlet generates the fragment. Since a portal page can contain several portlets, a *render()* petition is forwarded to each of the participating portlets, each of them returning a fragment. The portal builds a "quilt page" out of these fragments, and renders it back to the user.

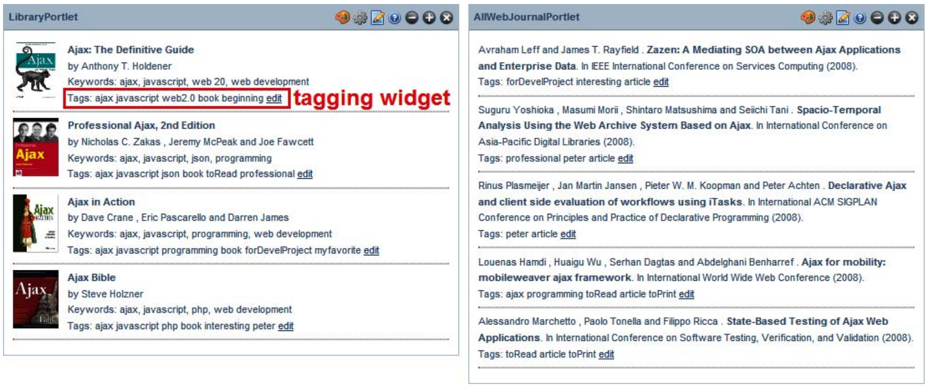
4 The Actors

Being content managers, portals can keep their own resources. Additionally, portals are also integration platforms, making external resources available through *portlets*. Figure 1 provides a snapshot of a portal page offering two portlets: *LibraryPortlet* and *AllWebJournalPortlet* that render content on books and publications, respectively. Both books and publications are kept outside the portal realm. The running example is available at `http://tagging.onekin.org/`.

Notice that both portlets, regardless of their different providers, offer the same set of tags. That is, tagging adapts to the hosting portal. This also implies that if the very same portlet is offered through a different portal then, the rendered tag set will be distinct since tags reflect the projects, roles and ways of working of the organization at hand.

This provides grounds for tagging to be supported as a portal commodity, and introduces three actors in portal tagging, namely: **portlets**, which provide the tag-able resources; **the portal**, which embodies the portal users as potential taggers; and **the tagging commodity**, i.e. a portal component that provides tagging utilities. This paper looks at two such tagging functionalities: tag assignment and tag-based querying.

However, the existence of three different actors should not jeopardize one of the portal hallmarks: interaction consistency across the portal. Tagging should be homogenously achieved throughout the portal, no matter where the resource resides (i.e. which portlet renders it). Additionally, and on top of the portal mandate, the desire


LibraryPortlet

Ajax: The Definitive Guide
by Anthony T. Holdener
Keywords: ajax, javascript, web 20, web development
Tags: ajax javascript web2.0 book beginning edit

Professional Ajax, 2nd Edition
by Nicholas C. Zakas , Jeremy McPeak and Joe Fawcett
Keywords: ajax, javascript, json, programming
Tags: ajax javascript json book toRead professional edit

Ajax in Action
by Dave Crane , Eric Pascarello and Darren James
Keywords: ajax, javascript, programming, web development
Tags: ajax javascript programming book forDevelProject myfavorite edit

Ajax Bible
by Steve Holzner
Keywords: ajax, javascript, php, web development
Tags: ajax javascript php book interesting peter edit

AllWebJournalPortlet

Avraham Leff and James T. Rayfield . **Zazen: A Mediating SOA between Ajax Applications and Enterprise Data**. In IEEE International Conference on Services Computing (2008).
Tags: forDevelProject interesting article edit

Suguru Yoshioka , Masumi Morii , Shintaro Matsushima and Seiichi Tani . **Spacio-Temporal Analysis Using the Web Archive System Based on Ajax**. In International Conference on Asia-Pacific Digital Libraries (2008).
Tags: professional peter article edit

Rinus Plasmeijer , Jan Martin Jansen , Pieter W. M. Koopman and Peter Achten . **Declarative Ajax and client side evaluation of workflows using iTasks**. In International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (2008).
Tags: peter article edit

Louenas Hamdi , Huaigu Wu , Serhan Dagtas and Abdelghani Benharref . **Ajax for mobility: mobileweaver ajax framework**. In International World Wide Web Conference (2008).
Tags: ajax programming toRead article toPrint edit

Alessandro Marchetto , Paolo Tonella and Filippo Ricca . **State-Based Testing of Ajax Web Applications**. In International Conference on Software Testing, Verification, and Validation (2008).
Tags: toRead article toPrint edit

Fig. 1. A portal page offering two portlets (i.e. *LibraryPortlet* and *AllWebJournalPortlet*)

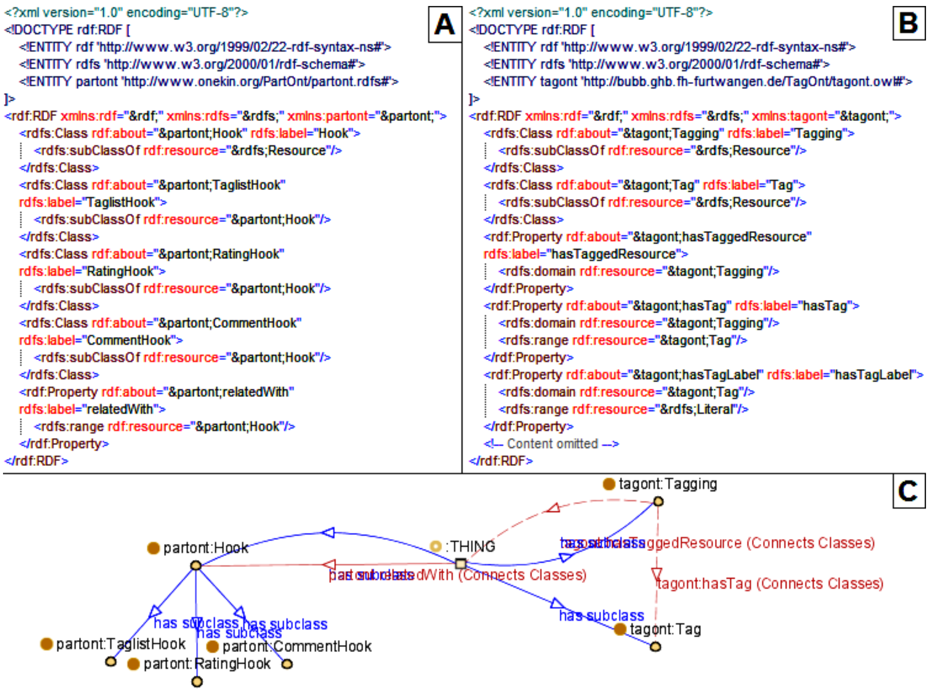
for tag consistency emerged as a major request among portal users, (e.g. “how will others find my content if I don’t use the same tags over and over?”) as drawn from a recent study [10].

This consistency is two-fold. “Back-end consistency” implies the use of a common structure for tagging data, e.g. a common set of tags. On the other hand, “front-end consistency” entails tagging interactions to be achieved seamlessly and cohesively across the portal using similar rendering guidelines. Next sections delve into the details.

5 Back-End Consistency

Back-end consistency implies tagging data to be a portal asset rather than being disseminated across different silos. Tagging data basically refers to tags, taggers and tag-able resources. Both, taggers and tags, are naturally held by the portal. However, tag-able resources can be outside the portal realm. Although tagging could become a portal duty, some tag-able resources would still be provided by third-party portlets. Therefore, a mechanism is needed for portlets to make the portal aware of their tag-able resources.

The main means for portlet-to-portal communication is the markup fragment that the portlet delivers to the portal. Here, the portal is a mere conduit for portlet markups. Portals limit themselves to provide a common skin and appropriate decorators to portlet



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
  <!ENTITY partont 'http://www.onekin.org/PartOnt/partont.rdfs#'>
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:partont="&partont;">
  <rdfs:Class rdf:about="&partont;Hook" rdfs:label="Hook">
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&partont;TaglistHook"
  rdfs:label="TaglistHook">
    <rdfs:subClassOf rdf:resource="&partont;Hook"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&partont;RatingHook"
  rdfs:label="RatingHook">
    <rdfs:subClassOf rdf:resource="&partont;Hook"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&partont;CommentHook"
  rdfs:label="CommentHook">
    <rdfs:subClassOf rdf:resource="&partont;Hook"/>
  </rdfs:Class>
  <rdf:Property rdf:about="&partont;relatedWith"
  rdfs:label="relatedWith">
    <rdfs:range rdf:resource="&partont;Hook"/>
  </rdf:Property>
</rdf:RDF>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
  <!ENTITY tagont 'http://bubb.ghb.fh-furtwangen.de/TagOnt/tagont.owl#'>
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:tagont="&tagont;">
  <rdfs:Class rdf:about="&tagont;Tagging" rdfs:label="Tagging">
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&tagont;Tag" rdfs:label="Tag">
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
  </rdfs:Class>
  <rdf:Property rdf:about="&tagont;hasTaggedResource"
  rdfs:label="hasTaggedResource">
    <rdfs:domain rdf:resource="&tagont;Tagging"/>
  </rdf:Property>
  <rdf:Property rdf:about="&tagont;hasTag" rdfs:label="hasTag">
    <rdfs:domain rdf:resource="&tagont;Tagging"/>
    <rdfs:range rdf:resource="&tagont;Tag"/>
  </rdf:Property>
  <rdf:Property rdf:about="&tagont;hasTagLabel" rdfs:label="hasTagLabel">
    <rdfs:domain rdf:resource="&tagont;Tag"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdf:Property>
  <!-- Content omitted -->
</rdf:RDF>
```

Fig. 2. The *PartOnt* (a) and the *TagOnt* (b) ontologies together with their *Protégé* rendering counterparts (c).

fragments, being unaware of what this markup conveys. We propose to annotate this markup with tagging concerns using *RDFa* [1].

RDFa is a W3C standard that provides syntax for communicating structured data through annotating the XHTML content. In our case, *RDFa* offers a means for the portlet provider to communicate the portlet consumer the existence of tag-able resources. The idea is to capitalize on the fragment layout to annotate tag-able resources.

Firstly, an ontology is defined which is later used to annotate the fragment. This ontology should serve to indicate both *what* to tag and *where* to tag (see later). This is the aim of *PartOnt* (*Part*icipatory *Ont*ology), an ontology that aims at capturing manners in which users engage in the participatory web. One of these ways is of course, tagging. Rather than defining its own concepts, *PartOnt* capitalizes on existing ontologies that already formalize some of these notions. Specifically, *PartOnt* benefits from *TagOnt*, an ontology that captures tagging by describing resources, tags, taggers, tagging time, etc [11]. Figure 2 shows these ontologies, both the RDF code and the *Protégé* rendering counterpart.

These ontologies are then used to annotate the portlet markup. An example is shown in Figure 3. The JSP script outputs a *LibraryPortlet* fragment. Book data (i.e. title, authors, etc) are rendered as table rows (*TR*), where book keywords are iteratively enclosed within SPAN elements. All of the table cells are wrapped within a table (<*table*>) which in turns is wrapped in another table together with the book-cover image.

This markup is then annotated along the previous ontologies. Specifically, the following structural HTML elements are annotated [2]:

```
<jsp:useBean id="library" scope="request" class="java.util.ArrayList" />
<div   xmlns:books="http://www.onekin.org/library/"
       xmlns:partont="http://www.onekin.org/PartOnt/partont.rdfs#"
       xmlns:tagont="http://bubb.ghb.fh-furtwangen.de/TagOnt/tagont.owl#">

       <c:forEach var="book" items="${library}">
             <table about="[books:${book.id}]"><tr>
                   <td><!-- BOOK'S PAPERBACK IMAGE --></td>
                   <td>
                         <table>
                               <tr><td><!-- BOOK'S TITLE --></td></tr>
                               <tr><td><!-- BOOK'S AUTHORS --></td></tr>
                               <tr><td typeof="tagont:Tagging">
                                     <div rel="tagont:hasTaggedResource" resource="[books:${book.id}]"/>
                                     Keywords:
                                     <span rel="tagont:hasTag">
                                           <c:forEach var="keyword" items="${book.keywords}">
                                                 <span typeof="tagont:Tag" property="tagont:hasTagLabel">
                                                       <c:out value="${keyword}"/>
                                                 </span>
                                           </c:forEach>
                                     </span>
                               </td></tr>
                               <tr><td>
                                     <div style="display:none;" rel="partont:relatedWith"
                                                                typeof="partont:TaglistHook" />
                               </td></tr>
                         </table>
                   </td>
             </tr></table>
       </c:forEach>
</div>
```

Fig. 3. JSP that delivers a fragment markup with annotations along the *TagOnt* and *PartOnt* ontologies

– HTML element that embodies a tag-able resource. In our example, this corresponds to the outer <*table*> element. This element now includes an *"about"* attribute which denotes the existence of a tag-able resource. The identifiers of tag-able resources are supported as *Uniform Resource Identifiers* (URIs). Following Semantic Web practices, these URIs are created by concatenating a namespace with a resource's key,
– HTML element that conveys potential tags. In this case, we identify keywords as playing such role. This implies to annotate the <*span*> element with the *"tagont:Tag*" annotation. These tags are provided for the portal's convenience, and they are expected to describe the resource content. It is up to the portal to incorporate these tags as suggestions during tagging. These portlet-provided tags should not be mistaken with those provided by the portal users.

These annotations permit the portlet consumer (i.e. the portal) to become aware of resources and tags coming from external sources. This external data is incorporated into the portal not when it is rendered but when it is tagged. When the first tag is added, the portal check if the resource ID is already in the tagging repository (see later Figure 4).

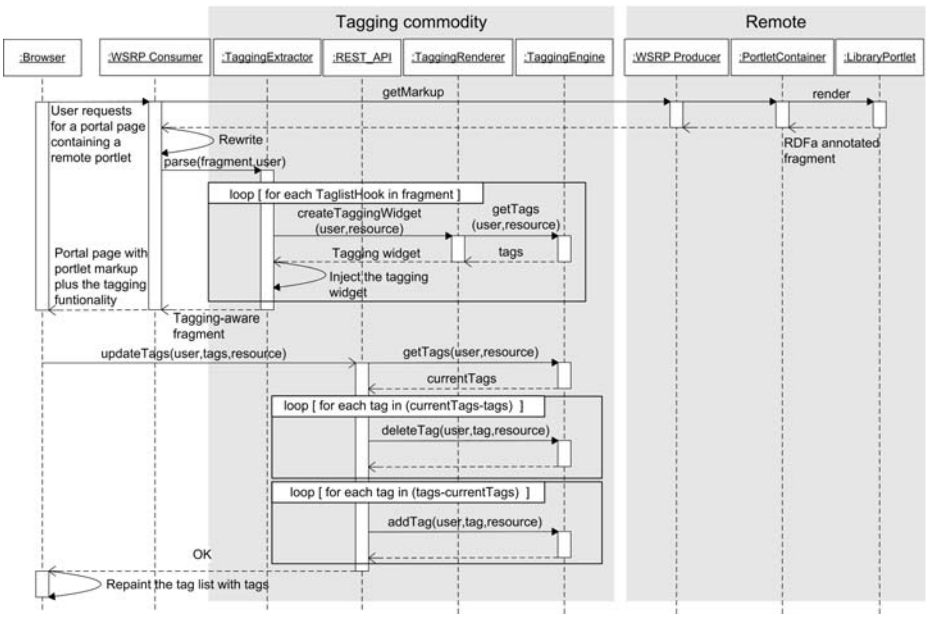

Tagging commodity
Remote
:Browser
:WSRP Consumer
:TaggingExtractor
:REST_API
:TaggingRenderer
:TaggingEngine
:WSRP Producer
:PortletContainer
:LibraryPortlet
getMarkup
render
User requests for a portal page containing a remote portlet
Rewrite
parse(fragment,user)
RDFa annotated fragment
loop [for each TaglistHook in fragment]
createTaggingWidget (user,resource)
getTags (user,resource)
Tagging widget
tags
Inject the tagging widget
Portal page with portlet markup plus the tagging funtionality
Tagging-aware fragment
updateTags(user,tags,resource)
getTags(user,resource)
currentTags
loop [for each tag in (currentTags-tags)]
deleteTag(user,tag,resource)
loop [for each tag in (tags-currentTags)]
addTag(user,tag,resource)
OK
Repaint the tag list with tags

Fig. 4. Interaction diagram: *base requests* vs. *tagging requests*

However, resource IDs and tags are not introduced in the tagging repository right away. Rather, the tagging commodity should include a *"cleaning module*"

[2] As far as this work is concerned, we ignore the resource content (i.e. we do not annotate e.g. titles or authors of book resources). All the portal needs to know is that a tag-able resource is being rendered. The details about the rendering itself are left to the portlet.

to ascertain whether two tags/resources really stand for the same notion. For instance, the same resource can be offered as a book in *LibraryPortlet* and as a publication in *AllWebJournalPortlet*. Likewise, this resource can be tagged as *"ServiceOrientedArchitecture"* in one place and *"SOA"* in the other. This cleaning module will provide some heuristics to ascertain the equality of resources and tags being offered in different forms by different resource providers. This effort is akin to the view of the portal as an *integration* platform, and an argument in favour of tagging being conducted through the portal rather than as a disperse activity performed at each resource provider.

6 Front-End Consistency

Portals are a front-end technology. Much of their added value (and investment) rests on how content is rendered and navigated. In this context, presentation consistency is a must to hide the diverse sources that feed the portal. Tagging wise, consistency mandates tagging interactions to be seamlessly and coherently achieved across the portal. This would not be much of a problem if tagging were only up to the portal. But, this paper highlights that portal tagging is a joint endeavour among the portal and the companion portlets. Rendering wise, this coupling can be achieved at the portlet place (through markup portions referred to as *widgets*) or at the portal place (using a publish/subscribe approach with local portlets). Next subsections address each of these approaches.

6.1 Front-End Consistency through Widgets

Seamlessness calls for tagging to be conducted at the place tag-able resources are rendered (side-by-side rendering). This place is the portlet fragment. But portlets should not deliver their own tagging functionality since a premise of this work is that such functionality should be provided by the portal. But, portals are traditionally mere proxies for the portlet markup. Tagging however, requires portals to take a more active role. Besides skins and decorators, portals now become the purveyors of tagging widgets to be injected into the portlet markup.

The question is how can the portal know where to inject these widgets? Annotations are again used for this purpose. Specifically, the *PartOnt* ontology includes a *Hook* class, with a subclass *TaglistHook* that denotes an extension point for adding markup to update the tag list. This class annotates the HTML element that plays the "hook" role. Figure 3 shows our sample fragment where this role is played by a <*div*> element. At execution time, the portal locates the "hooks" and injects the tagging widget (see later).

Markup coming from the portlet should be seamlessly mixed together with markup coming from the portal so that the user is unaware of the different origins. After all, this is the rationale behind letting the portlet specify the tagging hooks: injecting the extra markup in those places already foreseen by the portlet designer so that the final rendering looks harmonious. However, the distinct markup origins become apparent to the portal which needs to propagate the user interactions to the appropriate target. Specifically, *base requests* (i.e. those with the portlet markup) are propagated to the portlet provider, while *tagging requests* (i.e. those with the tagging widget) are processed by the tagging commodity.

Figure 4 provides an overview of the whole process where these two types of interactions are distinguished:

1. *base request.* According with the WSRP standard, user interactions with portlet markup are propagated till reaching the appropriate portlet. In return, the portlet delivers a markup, now annotated with tagging metadata,
2. content annotation processing. At the portal place, the tagging commodity (specifically an *RDFa* parser) extracts both tag-able resources and tags conveyed by the actual markup. This data is kept at the tagging repository.
3. *hook* annotation processing. If the markup also holds *"TaglistHook"* annotations, the tagging commodity (specifically, a markup renderer) outputs the appropriate widget to be injected at the hook place. The markup renderer can need to access the tagging repository, e.g. to recover the tags currently associated with a given resource.
4. markup rendering. The original markup has now become a tagging-aware fragment, i.e. a fragment through which tagging can be conducted,
5. *tagging request.* Now, the user interacts with the tagging markup (e.g. requesting the update of the tag set). This petition is directed to the tagging commodity which checks the additions and removals being made to the tag set kept in the repository. In return, the tagging commodity repaints the tagging markup.

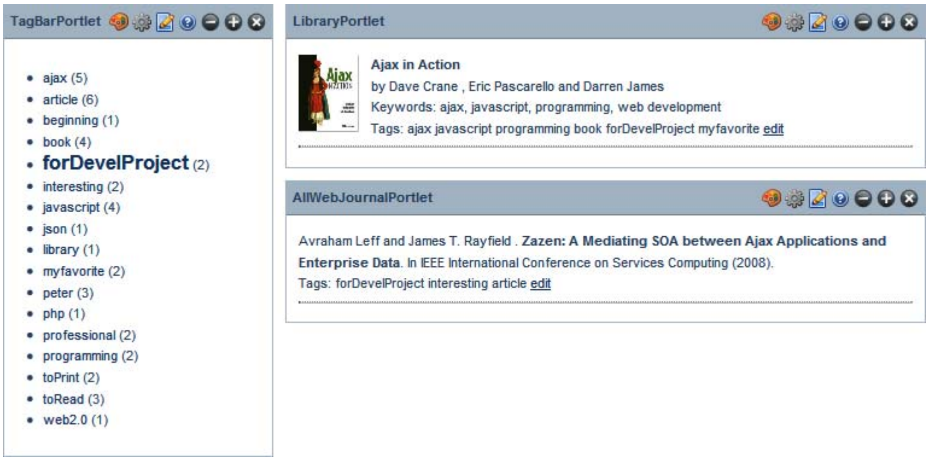
As the previous example illustrates, the co-existence of markups from different origins within the same portlet decorator brings an Ajax-like style to markup production. In Figure 4, lines with solid triangular arrowheads denote synchronous communication whereas open arrowheads stand for asynchronous communication. Specifically, the tagging request is asynchronously processed.

6.2 Front-End Consistency through Local Portlets

Previous subsection illustrates the case of a tagging functionality (e.g. tag update) to be achieved at the portlet place. However, other services can be directly provided by the portal but in cooperation with the companion portlets. Tag-based querying is a case in point.

Comprehensive querying implies the query to expand across resources, no matter their origin. A query for resources being tagged as *"forDevelProject"* should deliver books (hence, provided by the *LibraryPortlet* portlet), publications (hence, supplied by the *AllWebJournalPortlet* portlet), post blogs (locally provided), etc being tagged as used in this project. Such a query can be directly answered through the tagging repository that will return the set of resource identifiers meeting the query condition.

However, portals are a front-end technology. Providing a list of identifiers is not a sensible option when an end user is the addressee. Rather, it is the content of resource what the user wants to see. We need then to *de-reference* these identifiers. Unfortunately, the tagging repository can not "de-reference" those identifiers. The portal owns the tagging data. But it is outside the portal realm to know the resource content as well as


TagBarPortlet

- ajax (5)
- article (6)
- beginning (1)
- book (4)
- **forDevelProject** (2)
- interesting (2)
- javascript (4)
- json (1)
- library (1)
- myfavorite (2)
- peter (3)
- php (1)
- professional (2)
- programming (2)
- toPrint (2)
- toRead (3)
- web2.0 (1)

LibraryPortlet

Ajax in Action
by Dave Crane , Eric Pascarello and Darren James
Keywords: ajax, javascript, programming, web development
Tags: ajax javascript programming book forDevelProject myfavorite edit

AllWebJournalPortlet

Avraham Leff and James T. Rayfield . **Zazen: A Mediating SOA between Ajax Applications and Enterprise Data**. In IEEE International Conference on Services Computing (2008).
Tags: forDevelProject interesting article edit

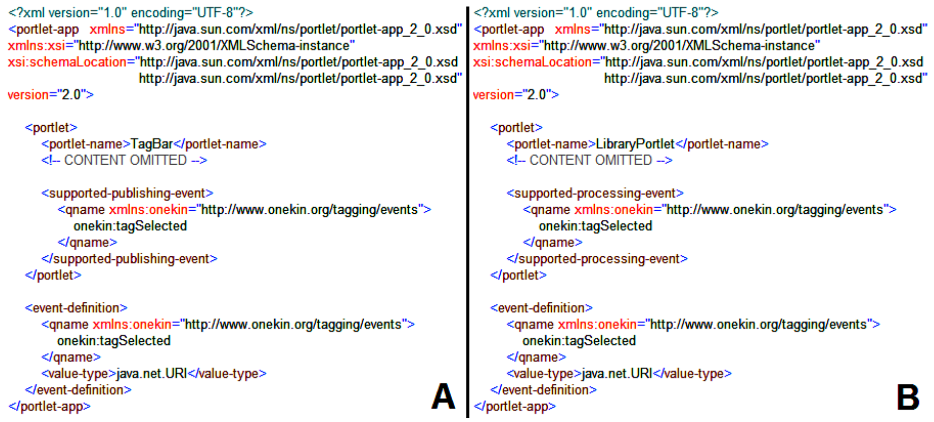
Fig. 5. Split query processing. Query specification goes through *TagBarPortlet*: the tag selected by the user is highlighted. Query outcome is delegated to the portlets holding the resource content, i.e. *LibraryPortlet* and *AllWebJournalPortlet*.

how this content is to be rendered. This is the duty of the resource providers, i.e. the portlets. Therefore, the portal can not accomplish the whole query processing on its own since this also involves content rendering.

Figure 5 illustrates this situation. First, a mean is needed for the user to express the query. For the sake of this paper, a simple portlet has been built: *TagBarPortlet*. This portlet consults the tagging repository, renders the tags available, and permits the users to select one of theses tags. The selection has two consequences. First, the selected tag is highlighted. Second, and more important, the companion portlets synchronize their views with this selection, rendering those resources that were tagged with the selected tag *at this portal*. This last point is important. The very same portlet can be offered through different portals. Hence, the same resource (e.g. a book) can be tagged at different places (i.e. through distinct portals). When synchronized with the *TagBarPortlet* of portal P1, the portlet just delivers those resources being tagged through portal P1.

This scenario again requires a means for portal-to-portlet communication. Previous section relies on the rendering markup as the means of communication. This was possible because the data flew from the portlet to the portal. However, now identifiers/tags go the other way around: from the portal to the portlets. To this end, we follow a publish/subscribe approach where data flows from the publisher (i.e. the portal, better said, the portal representative, i.e. *TagBarPortlet*) to the subscriber (e.g. *LibraryPortlet* and *AllWebJournalPortlet*). The availability of an event mechanism in the Java Portlet Specification [6] comes to our advantage.

Portlet events are intended to allow portlets to react to actions or state changes not directly related to an interaction of the user with the portlet. Portlets can be both event producers and event consumers. Back to our sample case, the query-specification portlet, i.e. *TagBarPortlet*, fires the appropriate event that is broadcasted by the portal to the resource-provider portlets to make then aware of the tag



```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app  xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
                    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
version="2.0">

  <portlet>
    <portlet-name>TagBar</portlet-name>
    <!-- CONTENT OMITTED -->

    <supported-publishing-event>
      <qname xmlns:onekin="http://www.onekin.org/tagging/events">
        onekin:tagSelected
      </qname>
    </supported-publishing-event>
  </portlet>

  <event-definition>
    <qname xmlns:onekin="http://www.onekin.org/tagging/events">
      onekin:tagSelected
    </qname>
    <value-type>java.net.URI</value-type>
  </event-definition>
</portlet-app>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app  xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
                    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
version="2.0">

  <portlet>
    <portlet-name>LibraryPortlet</portlet-name>
    <!-- CONTENT OMITTED -->

    <supported-processing-event>
      <qname xmlns:onekin="http://www.onekin.org/tagging/events">
        onekin:tagSelected
      </qname>
    </supported-processing-event>
  </portlet>

  <event-definition>
    <qname xmlns:onekin="http://www.onekin.org/tagging/events">
      onekin:tagSelected
    </qname>
    <value-type>java.net.URI</value-type>
  </event-definition>
</portlet-app>
```

Fig. 6. *portlet.xml* configuration files for *TagBarPortlet* and *LibraryPortlet*. Both portlets know about the *tagSelected* event

being selected. Publications and subscriptions are parts of the portlet definition and hence, expressed in the configuration file *portlet.xml*. Figure 6 shows those files for *TagBarPortlet* and *LibraryPortlet*. The former defines a published event, *tagSelected*, whereas *LibraryPortlet* acknowledges the capacity to process *tagSelected* events.

Processing *tagSelected* occurrences imply rendering the content of the so-tagged resources at the portlet place. For instance, *LibraryPortlet* should produce markup for those books being tagged with the tag provided in the event payload. However, *LibraryPortlet* holds the resource content but ignores how they have been tagged. This tagging data is kept at the portal. Therefore, the portlet needs to get such data from the tagging commodity. As a result, the tagging-commodity URL is included as part of the event payload, so that the portlet can construct a REST petition asking which of *its* resources are so-tagged at *this* portal. Therefore, the very same portlet can process *tagSelected* occurrences coming from different portals and hence, whose payloads refer to different URLs[3]. In this way, portlet interoperability is preserved.

Figure 7 provides the global view. First, the user selects *"forDevelProject"* as the tag to be used as the filtering criteria. This request is handled by *TagBarPortlet* that signals a *tagSelected* occurrence. The portal forwards this occurrence to their subscribers: *LibraryPortlet* and *AllWebJournalPortlet*. Processing *tagSelected* involves first, to query the *TaggingEngine* about the so-tagged resources. To this end, the *REST_API* provides the *getResourceByTag* method. This method outputs a list of resource identifiers for which the *LibraryPortlet* should locally retrieve the content and produce the markup. This process is accomplished for all the resource-provider portlets. This ends the state changing logic phase of the portlet lifecycle.

[3] An alternative design would have been for *TagBarPortlet* to recover itself all resorce identifiers that exhibit the selected tag, and include the whole set of identifiers as part of the event payload. On reception, the portlet filters out its own resources. However, this solution does not scale up for large resource sets. Additionally, the option of restricting the payload to just those resources of the addressee portlet forces to have a dedicated event for each portlet.

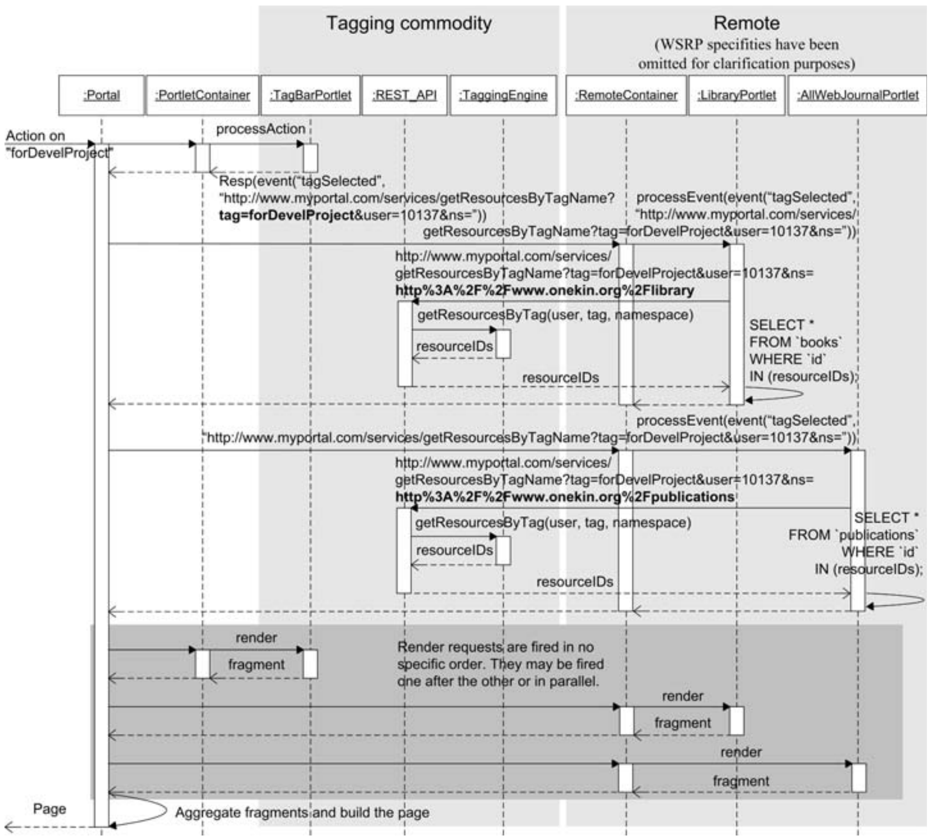

Tagging commodity
Remote
(WSRP specifities have been omitted for clarification purposes)
:Portal
:PortletContainer
:TagBarPortlet
:REST_API
:TaggingEngine
:RemoteContainer
:LibraryPortlet
:AllWebJournalPortlet
Action on "forDevelProject"
processAction
Resp(event("tagSelected", "http://www.myportal.com/services/getResourcesByTagName?tag=forDevelProject&user=10137&ns="))
processEvent(event("tagSelected", "http://www.myportal.com/services/getResourcesByTagName?tag=forDevelProject&user=10137&ns="))
http://www.myportal.com/services/getResourcesByTagName?tag=forDevelProject&user=10137&ns=http%3A%2F%2Fwww.onekin.org%2Flibrary
getResourcesByTag(user, tag, namespace)
resourceIDs
resourceIDs
SELECT * FROM `books` WHERE `id` IN (resourceIDs);
processEvent(event("tagSelected", "http://www.myportal.com/services/getResourcesByTagName?tag=forDevelProject&user=10137&ns="))
http://www.myportal.com/services/getResourcesByTagName?tag=forDevelProject&user=10137&ns=http%3A%2F%2Fwww.onekin.org%2Fpublications
getResourcesByTag(user, tag, namespace)
resourceIDs
resourceIDs
SELECT * FROM `publications` WHERE `id` IN (resourceIDs);
render
fragment
Render requests are fired in no specific order. They may be fired one after the other or in parallel.
render
fragment
render
fragment
Page
Aggregate fragments and build the page

Fig. 7. Handling a *tagSelected* occurrence

The rendering phase builds up the portal page out of the portlet fragments. This implies sending the *render()* request to each of the portlets of the page, assembling the distinct markups obtained in return, and render the result back to the user. For our sample case, the outcome is depicted in Figure 5.

7 Revision and Conclusions

This work argues for portal tagging to be a joint endeavour between resource providers (i.e. portlets) and resource consumers (i.e. portals). Additionally, portlets are reckoned to be interoperable, i.e. deliverable through different portals. These observations advocate for tagging to be orthogonally supported as a crosscut on top of portlets, i.e. a portal commodity. Driven by two main functionalities, tag update and tag-based querying, this paper identifies main components of such a commodity:

- back-end components: a tagging engine, a tagging extractor (in our case, a *RDFa* parser) and a cleaning module,

– front-end components: a query-specification portlet (e.g. *TagBarPortlet*) and a tagging renderer that output tagging widgets.

This work explores the technical requirements of such solution, namely:

– from the portlet provider perspective, portlet fragments need to be annotated along a basic tagging ontology,
– from the portlet consumer viewpoint (i.e. the portal), a tagging commodity should be available,
– from the position of the container of remote portlets at the portal, the WSRP implementation needs to be extended for the *getMarkup* method to cater for injections of widget markup into the portlet fragment.

A tagging commodity has been implemented for the *WebSynergy*[4] portal engine, a Sun's deliverable of the open-source *Liferay*[5] engine, using TASTY as the tagging engine[6]. This implementation evidences the feasibility of the approach, although real test cases are needed to fully test scalability problems. The benefits include:

– portal ownership of tagging data,
– increases consistency in the set of tags used to annotate resources, regardless of the resource owner,
– facilitates consistency among tagging activities, no matter the portal application through which tagging is achieved,
– permits tagging to be customized based on the user profile kept by the portal. For instance, the suggested set of tags can be based on the user profile, the projects he participates in, etc.

As in other situations where applications need to cooperate, the main challenge rests on agreeing in common terms and protocols. In our case, this mainly implies the standardization of the tagging ontology, and the REST API.

Acknowledgements. This work is co-supported by the Spanish Ministry of Education, and the European Social Fund under contract TIN2008-06507-C02-01/TIN (MODELINE), and the Avanza I+D initiative of the Ministry of Industry, Tourism and Commerce under contract TSI-020100-2008-415. Pérez enjoys a doctoral grant from the Basque Government under the "Researchers Training Program", and Arellano has a doctoral grant from the Spanish Ministry of Science & Education.

References

1. Adida, B., Birbeck, M.: RDFa Primer. Technical report, W3C Working Group (2008), `http://www.w3.org/TR/xhtml-rdfa-primer/`
2. Ames, M., Naaman, M.: Why we tag: motivations for annotation in mobile and online media. In: CHI 2007: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 971–980 (2007)

[4] https://portal.dev.java.net/
[5] http://www.liferay.com/
[6] http://microapps.sourceforge.net/tasty/

3. Díaz, O., Rodríguez, J.J.: Portlets as Web Components: an Introduction. Journal of Universal Computer Science, 454–472 (2004)
4. DiMicco, J., Millen, D.R., Geyer, W., Dugan, C., Brownholtz, B., Muller, M.: Motivations for social networking at work. In: CSCW 2008: Proceedings of the ACM 2008 Conference on Computer Supported Cooperative Work, pp. 711–720 (2008)
5. Golder, S.A., Hubermann, B.A.: The Structure of Collaborative Tagging System. In: CoRR (2005)
6. Java Community Process (JCP). JSR 286: Portlet Specification Version 2.0 (2008), `http://www.jcp.org/en/jsr/detail?id=286`
7. Millen, D.R., Yang, M., Whittaker, S., Feinberg, J.: Social bookmarking and exploratory search. In: ECSCW 2007: Proceedings of the Tenth European Conference on Computer Supported Cooperative Work, pp. 21–40 (2007)
8. Muller, M.J.: Comparing tagging vocabularies among four enterprise tag-based services. In: GROUP 2007: Proceedings of the 2007 International ACM Conference on Supporting Group Work, pp. 341–350 (2007)
9. OASIS. Web Services for Remote Portlets (WSRP) Version 2.0 (2008), `http://www.oasis-open.org/committees/wsrp/`
10. Thom-Santelli, J., Muller, M.J., Millen, D.R.: Social tagging roles: publishers, evangelists, leaders. In: CHI 2008: Proceeding of the twenty-sixth annual SIGCHI Conference on Human Factors in Computing Systems, pp. 1041–1044 (2008)
11. Knerr, T.: Tagging Ontology - Towards a Common Ontology for Folksonomies, `http://tagont.googlecode.com/files/TagOntPaper.pdf`