Mobile Widget Sharing by Mining Peer Groups

Xi Bai[1], Baoping Cheng[2], and Dave Robertson[1]

[1] School of Informatics, University of Edinburgh, UK
[2] China Mobile Research Institute, Beijing, China
xi.bai@ed.ac.uk, chengbaoping@chinamobile.com, dr@inf.ed.ac.uk

Abstract. Developers have began to wrap desired functionalities into widgets based on Web 2.0 techniques on mobile devices. Demand for these light-weight Web applications is expected to grow rapidly in near future. Since a widgets can be recognized as a unit providing a specific service, from the perspective of choreography, this paper adapts widgets into a mobile Peer-to-Peer (P2P) environment and proposes a light weight group discovery approach to assist service discovery based on domain ontologies and semantic clustering. Latent Semantic Indexing (LSI) and Singular Value Decomposition (SVD) are employed and assist our approach in building up a Knowledge Base (KB) on each peer. This approach can prune the service search space and trigger the initial formation of peer communities. Its performance is also assessed via simulation results. A framework for porting widgets to different widget engines has been designed, making use of the above ontologies, and a basic widget transformation platform is implemented and tested in a case study.

1 Introduction

A widget is a light-weight Web application, which can be used to implement a single function and access to Internet with Web 2.0 techniques. At time of this writing, there is no unified definition for a widget and different organizations gave diverse descriptions for this kind of Web application, such as widget [6], gadget [7] or widget desktop application. W3C also gave a definition for a widget in order to normalize the development of widgets [8]. In terms of different run time environments, widgets mainly fall into three categories: Desktop Widgets (DW, e.g., Dashboard Widgets, Yahoo! Widgets, Sidebar Gadgets, Opera, DesktopX, Google Gadgets, Klip Folio, AveDesk, Adobe Air, Samurize etc.); Web Widgets (WW, e.g., Myspace, iGoogle, Facebook, Friendster, eBlogger etc.); Mobile Widgets (MW, e.g., Nokia WidSets and Mojax Moblets). There is a high possibility that widgets will become the next generation of applications on mobile devices taking the place of traditional techniques like J2ME.

Each widget or mobile application can be treated as a unit providing a specific service. Several solutions [1] [2] [3] have been proposed to describe service semantics from the perspective of orchestration, but little work has been done on service description and service discovery from the perspective of choreography. This paper proposes an mobile widget sharing and reuse strategy within a Peer-to-Peer (P2P) environment from the perspective of choreography that is achieved through the Interaction Model (IM) such as the one depicted in Figure 2. A light weight group discovery approach is proposed based on domain ontologies and semantic clustering. This machine-learning-driven approach can not only prune the service search space but also assist each peer in building up its Knowledge Base (KB) that acts like a profile describing its interests and further triggers the initial formation of peer communities that provide peers with a better environment for sharing their knowledge. For widget publishers, generic ontology is not required and any type of ontology matchmaker can be plugged into our implementation. Its performance is then accessed through simulation. Unlike traditional applications, widgets make use of normalized Web techniques including HTML, XML, CSS and Javascript. It is possible to port them to diverse engines on mobiles or PCs. However, it is not easy to run a widget directly on another type of widget engine and the reasons will be introduced in Section 5. Existing transforming tools like Amnesty Generator [1] and Widgetop [2] are too limited to be taken as general solutions. In this paper, a framework for transforming widgets with diverse formats is designed with the aid of the above ontologies and its initial implementation is presented via a case study.

The remainder of the paper is organized as follows. Section 2 present related work. Section 3 illustrates the P2P network structure for mobile widget sharing. Section 4 presents our light-weight group discovery approach and gives a choreography-based solution for widget searching. Section 5 describes a framework for porting widgets to different widget engines. Section 6 discusses our group discovery approach through a simulation and gives a case study of widget transformation. Section 7 concludes the paper and outlines our future directions.

2 Related Work

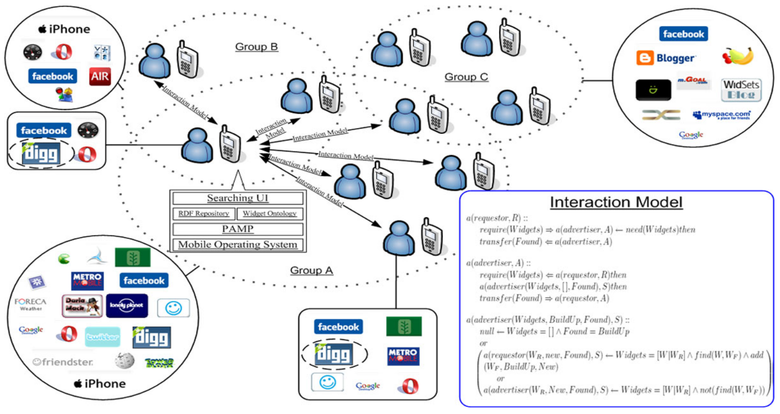
Widgets, created based on standard Web 2.0 techniques, are becoming widely used on mobile devices day by day. Due to the traditional structure of the mobile network, bandwidth and widgets provided are both limited by servers in a centralized network. It is therefore interesting to explore if a P2P environment that caters to the user's needs could be designed and applied to widget sharing. One of the core problems of adapting P2P architectures and Sematic Web techniques to mobile devices is how to cope with the computational costs. Orchestration and choreography are two perspectives from which researches currently investigate Web Services. The former describes the behaviors of a single peer and the latter describes a service system in a top view manner. Focusing on orchestration, several approaches have been proposed for applying semantics to Web Services, such as OWL-S [1], WSDL-S [2] and SAWSDL [3]. Consequently, several matchmakers such as OWLS-MX [4] and SAWSDL-MX [5] have been proposed and used for service discovery. However, little work has been done on service description and service discovery from the perspective of choreography.

[1] `http://www.amnestywidgets.com/GeneratorWin.html`
[2] `http://www.widgetop.com`

Moreover, limited computation capability of the mobile device also hampers the progress of service description strategies.

At time of this writing, there are two transformation tools for widgets in different formats as follows: Amnesty Generator and Widgetop. Amnesty Generator gives a way of transforming from Google Gadget, GrazrRSSreader and YouTube video to gadgets in the side bar. Widgetop is a Web desktop service based on AJAX techniques using the MAC UI style. However, these two tools both have their limitations. Amnesty Generator can only transform from Google Gadgets to gadgets running in the side bar of Window Vista and Widgetop can only transform from Apple Dashboard widgets to Web widgets.

3 Network Structure For Mobile Widget Sharing



```
a(requestor, R) ::
    require(Widgets) ⇒ a(advertiser, A) ← need(Widgets)then
    transfer(Found) ⇐ a(advertiser, A)

a(advertiser, A) ::
    require(Widgets) ⇐ a(requestor, R)then
    a(advertiser(Widgets, [], Found), S)then
    transfer(Found) ⇒ a(requestor, A)

a(advertiser(Widgets, BuildUp, Found), S) ::
    null ← Widgets = [] ∧ Found = BuildUp
    or
    ( a(requestor(W_R, new, Found), S) ← Widgets = [W|W_R] ∧ find(W, W_F) ∧ add
      (W_F, BuildUp, New)
        or
      a(advertiser(W_R, New, Found), S) ← Widgets = [W|W_R] ∧ not(find(W, W_F)) )
```

Fig. 1. Mobile P2P network illustration

The overall P2P network structure for widgets sharing among mobile devices is depicted in Figure 1. In this figure, each peer has an operating system, a PAMP bundle, a KB including an RDF repository and widget ontologies, and a searching UI. The ontologies and UI are provided and updated by a peer that plays the provider role. Here, we also assume that this role can assure that the exchanging process is quick and secure. When a requester asks for a widget, based on his or her inputs, several groups will be established based on the approach described in Section 4. Then IMs will be invoked as a protocol between peers in a group until all members are searched. Then the searched widgets, regarded as candidates, will be returned to the original requester. Meanwhile, these candidates are sorted by the values of their rank properties descendingly. Finally, the requester decides which widgets will be downloaded.

4 Peer Group Discovery

In order to efficiently search for required widgets on peers, a possible strategy is discovering a peer group that can prune the search space for each query. A *group* is a collection of peers that have common interests. In this section, we present our approach for clustering peers into groups. Suppose each peer holds an RDF snippet that describes all the widgets holden by itself. This RDF snippet can describe the relevant properties of each widget such as its URL, title, publisher, platform, categories, size, rank and so on. Due to the heterogeneity of widget properties described by different Web sites, we defined widget domain ontologies that will be introduced in Section 5. So it is unnecessary diverse Web sites use a generic ontology vocabulary to describe their issued widgets but just offer their matchmakers. After being mapped, the aliases will be unified by specific predefined concepts in widget ontologies. Based on these ontologies and a specific matchmaker, the RDF snippets could be automatically generated from a widget publication page in virtue of information extraction techniques. Finally, all the installed widgets on each peer are described by a single RDF file that forms its profile and will be stored in its local repository.

If an RDF snippet is matched with the requirer's query, the host peer will provide its address for download. Alternatively, the peer can provide the original URL of the widget on another peer from which it downloaded this widget before in case the peer has removed this widget but still retained the old version of the RDF repository. By analyzing the widgets on *Widgipedia*[3], we assume that features and descriptions are capable of indicating the functionalities of widgets and the peers that a requester originally wants to contact are previously recorded in its contact list. For instance, widget *flickrstrator* has following features: *article*, *blog*, *flash*, *flickr*, *images*, *photo*, *random*, and *web*. Given these assumption, we present our group discovery process including feature extraction, dimension reduction, group-name extraction and peer distribution as follows:
-Feature Extraction. Several methods have been proposed for selecting document features. Taking efficiency and limited computation resources into consideration, we use a Vector Space Model (VSM) to model the features of peers by querying the RDF repository on each peer. From the following query described in SPARQL [11], we can get all features and descriptions, which indicate the functionalities of services a peer can offer.

```
SELECT ?feature ?description
WHERE {
    ?wi rdf:type wp:WidgetInfo.
    ?wi wp:feature ?feature.
    ?wi wp:description ?description.}
```

According to VSM, each peer can be denoted by a feature vector with n dimensions: $(\alpha_{w_1} \cdot tf_{w_1}, \alpha_{w_2} \cdot tf_{w_2}, ..., \alpha_{w_n} \cdot tf_{w_n})$, where n denotes the number of values for node "feature" in the local repository, tf_{w_i} denotes the frequency of the ith values and α_i denotes the weighting factor of the ith value. We use the *inverse document frequency*, denoted by idf_{w_i}, as the weighting values so the feature vector can be denoted by $(tf_{w_1} \cdot idf_{w_1}, tf_{w_2} \cdot idf_{w_2}, ..., tf_{w_n} \cdot idf_{w_n})$. Here, $idf_{w_i} = \log(N/df_{w_i} + 0.01)$, where df_{w_i} is the *document frequency* that denotes the number of peers in which the ith value appears, and N denotes the total number of peers in the requester's contact list. The requester's contact list can

[3] http://www.widgipedia.com/widgets

be represented by a *feature-peer* matrix. Here, a *feature* denotes the value of a feature node. Suppose there are in total m peers in the contact list and they contain n different features. Then this contact list can be denoted by a $m \times n$ matrix M. The row vectors of M are named *feature vectors*, and the column vectors of M are named *peer vectors*. We use the suffix array [13] to calculate the frequency of each feature. Moreover, we also omit the frequencies that do not exceed a predefined frequency-threshold value $freq_{min}$.

-Dimension Reduction. Though the average number of the feature values in an RDF snippet is small, the number of peers and the number of widgets on each peer will be increased day by day and the dimension of feature vector will be very large consequently. Since some of features are redundant, here we reduce the dimension of the feature vector using Latent Semantic Indexing (LSI) [12]. The *feature-peer* matrix M is first constructed. Then Singular Value Decomposition (SVD) is applied to this matrix and we have $M = USV^T$, where U denotes the *left singular vectors* of M, V denotes the right singular vectors of M, and S denotes a matrix that has the singular values of M sorted decreasingly along its diagonal. After selecting the value of k, we can get a k-rank approximation M_k of matrix M, $M_k = U_k S_k V_k^T$, where U_k and V_k denote two matrices whose columns are the first k columns of U and the first k columns of V, respectively, and V_k denotes a matrix whose diagonal elements are the k largest singular values of M. Here, we select the minimum integer k by the following inequation:

$$\frac{||M_k||_F}{||M||_F} = \frac{\sqrt{\sum_{i=1}^{k}(\sigma_i^2)}}{\sqrt{\sum_{i=1}^{r_M}(\sigma_i^2)}} \geqslant \rho \qquad (1)$$

Here, r_M denotes the rank of matrix M, σ_i denotes the ith singular value, and $||M||_F$ denotes the *Frobenius* norm of matrix M. We can map the original feature vectors to a new feature space and finally reduce their dimension by:

$$d^* = d^T V_k^T S_k^{-1} \qquad (2)$$

Here, d denotes the original feature vector of a peer and d^* denotes the new one after dimension reduction.

-Group-Name Extraction. Here, the distance between two feature vectors is calculated by *cosine distance*. We use this distance to select the most representative name for each group. Since group names may be not only terms but also phrases, we reconstruct a new matrix R with the dimension $m \times (r + m)$, which expresses both terms and phrases in the column space of the *feature-peer* matrix. Here, r denotes the total number of the phrases in the requester's contact list. After SVD, each column vector of matrix U_k denotes the abstract features of widgets for a peer. We can calculate the *cosine distance* between a abstract feature vector and a term-and-phrase vector and get the distance matrix $D = U_k^T R$. For each row vector, we find out the maximum score. Its corresponding term or phrase is regarded as a candidate group name. Since a group name should be

concise, we should deal with the names which have duplicate semantics. Regarding the candidate names as peers, we construct another *feature-peer* matrix S with scores as its elements. After length normalization, we can get the matrix containing similarities between candidate names by formula $S^T S$. For each row of this matrix, only the name with the highest score will be retained. The effect for this step can be see in Table 2 and Table 3 in Section 6.
-Peer Distribution. After above three steps, we can determine the number of groups in the requester's contact list. Then we make all the peers fall into these groups. We define a matrix P in which each group label is represented by a column vector. Then we can get a clustering matrix $C = P^T M$ whose element c_{ij} denotes the similarity between the jth peer and the ith group, where M is the original *feature-peer* matrix. If similarity c_{ij} exceeds the predefined distribution threshold $dist_{min}$, the jth peer will be classified into the ith group finally. We will further discuss the selection of $dist_{min}$ in Section 6.

When a requester sends out a searching request, the local RDF repository will be first searched in order to assure that the requested widget has not been installed locally. Otherwise, the local server will remind the requester of overwriting or updating the existing widget or substituting it with a different one. If the requested widget can not be found locally, the requiring message will be sent to the peers that are in the same group with the requester. Normally, one peer may belong to several groups simultaneously, and different groups may have intersection subgroups. So we should find out which group of the requester should be searched first. On the other hand, if peers in the requester's contact list belong to more than one group to which the requester also belongs, they will be searched more than once. In order to avoid overlapping searches, we maintain a list to record peers that has been searched. The overall searching process is like the above *Peer Distribution* process. The query phrase (i.e., the request) is represented by a feature vector, Q, whose dimension is equal to the dimension of the feature vector after the *Dimension Reduction* process. Then we can get a vector $C' = P^T Q$ whose element C'_i denotes the similarity between the query phrase and the ith group. By ranking these similarities descendingly, we can find out the most relevant group that should be searched first. If the found widgets are not satisfactory, the group with the second highest similarity will be searched consequently. When the requester adds a new peer in his or her contact list, the new peer will be first classified into an existing group or regarded as a member of a new group and the local group information will be also updated. The concrete method is analogous to the above method for matching a query phrase and will not be described here for the sake of brevity. The above group discovery approach triggers the formation of peer communities by providing the basic community seeds. *Community* is a non-empty set of peers that share a non-empty set of interests that they have in common [15]. After being discovered, each group can send out invitations to selected peers in terms of the following query, which allows a peer to start discovering new group members from its neighborhood instead of flooding the network which often causes a big burden on bandwidth and processors.

```
SELECT ?peer
WHERE {
    ?peer rdf:type wp:Peer.
    ?peer foaf:holdsAccount ?user.
    ?group sioc:has_member ?user.}
for each $peer_i$ from ?peer
    SELECT ?friend
    WHERE {
        ?friend rdf:type wp:Peer.
        $peer_i$ foaf:knows ?friend     }
end-for
```

Peer is a unit that is capable of achieving specific goals. From the perspective of choreography discussed in Section 2, peers collaborate with each other through interactions. We use Lightweight Coordination Calculus (LCC) to describe these interactions. LCC is a language used for describing the interactions between peers and supporting decentralized systems [9]. LCC describes the interaction between peers using an interaction model, which describes the choreography between peers in appropriate roles in an intended interaction. After being grouped, a peer searches and retrieves desired widgets in its group using the IM described in Figure 2.

An requester, R, sends out a message to a potential widget advertiser, A, in order to retrieve required widgets, Widgets; then A sends back a message to R and transfers the found widgets to R. The set $Widgets$ contains all the widgets that R wants to retrieve. A receives the message containing the R's required widgets; then A changes role to being the advertiser over the set $Widgets$ and finds out if A contains the widgets in $Widgets$; then all the found widgets are saved into set $Found$ and sent back to R. $Found$ is a set containing all the found widgets required by R. The advertiser A searches for a widget, W, in turn from the set $Widgets$; then if W is found in A's local widget repository, W will be added in set $Found$; otherwise, W will be ignored.

$a(requester, R) ::$
$\quad require(Widgets) \Rightarrow a(advertiser, A) \leftarrow need(Widgets) then$
$\quad transfer(Found) \Leftarrow a(advertiser, A)$

$a(advertiser, A) ::$
$\quad require(Widgets) \Leftarrow a(requester, R) then$
$\quad a(advertiser(Widgets, [\,], Found), S) then$
$\quad transfer(Found) \Rightarrow a(requester, R)$

$a(advertiser(Widgets, BuildUp, Found), S) ::$
$\quad null \leftarrow Widgets = [\,] \wedge Found = BuildUp$
$\quad or$

$$\left(\begin{array}{l} a(requester(W_R, new, Found), S) \leftarrow Widgets = [W|W_R] \wedge find(W, W_F) \wedge add \\ (W_F, BuildUp, New) \\ \quad or \\ a(advertiser(W_R, New, Found), S) \leftarrow Widgets = [W|W_R] \wedge not(find(W, W_F)) \end{array}\right)$$

Fig. 2. Widget retrieving IM

5 Widget Format Transformation Based on Domain Ontologies

Normally, a widget originally running on a specific widget engine can not be run on another widget engine without modifications. The main reasons for this are described as follows: firstly, the names of attributes contained in the configuration files are different; secondly, the ways of packaging and the structures of the files inside the packages may differ; thirdly, there are several different widget engines whose APIs are still not standardized. Manually revising the widget source codes is tedious and it is impossible for a normal user who does not have professional knowledge to fulfill this task. Also, the accuracy of the manual modification can not be guaranteed. Table 1 gives a brief description of the file structures belonging to several types of the most popular widgets.

Table 1. Analysis for the structures of main widgets

Engine	Format	Icon	Manifest	Main File	.js	.css
Dashboard	.[zip,wdgt]	Icon.png	Info.plist	any.html	√	√
Nokia WRT	.wgz	Icon.png	Info.plist	any.html	√	√
Google	.gg	any.png	gadget.gmanifest	main.xml	√	—
Opera	.zip	any.[png,gif]	config.xml	index.html	√	√
Joost	.joda	any.[png,svg,jpeg,gif]	config.xml	any.[jwl,html,svg]	√	√

In Table 1, we can see that each type of the widget contains a configuration file, a main file and several JavaScript files. Although requesters can use the group discovery approach and the IM described in Section 4 to search other peers and find out the expected widget with formats they support, usually the widget publishers, especially independent developers, do not provide multiple versions of their widgets. In this section, we propose an automatic transforming framework based on domain ontologies, see in Figure 3. This process will be also wrapped in an OKC held by each peer acquiescently. Following this figure, we now describe the transforming process as follows:

-Template Repository. According to the requester's requirement, the target template corresponding to a target widget is first selected. A *Template* is a kind of description that depicts the file buildup, the naming method, the configuration file and the main file of a widget with a specific format.

-Unpacking and Packing Modules. We use *java.util.zip* package included in JDK for unpacking and packing widget files.

-Preprocessing Module. This module unifies formats of main files from different types of widgets and prepares inputs for the next mapping process. Basically, there are two types of main files: HTML files and XML files. Here, we use XML to standardize the format of main files.

-Analyzing Module. This module identifies the format of the original widget by analyzing the file bundle after the unpacking process.

-Mapping Module. This module associates the elements within the original template with the elements within the target template.

-Updating Module. Our widget ontologies should be updated constantly, according *ontology evolution* [10] theory. We are going to compare new elements and existing elements by the similarity based on *edit distance* [14], but there is no implementation for this module currently.

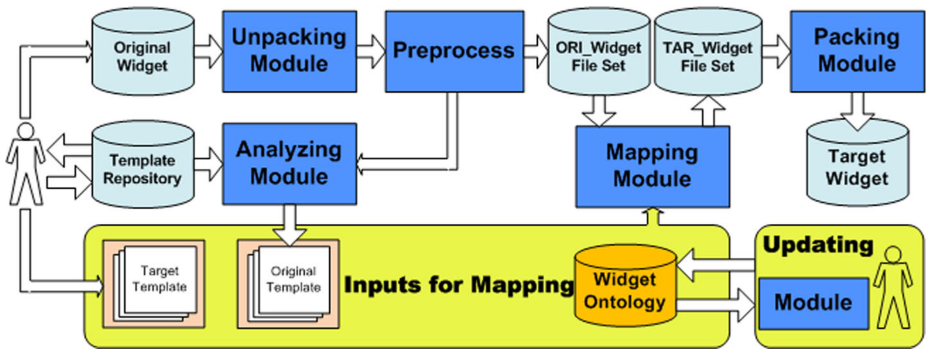


Fig. 3. Framework for the widget format transformation

6 Simulation and Case Study

We use S60 Platform SDKs for Symbian OS (3rd Edition Feature Pack 2) [4] and PAMP [5] to simulate our mobile P2P environment. We collect widgets randomly from *Widgipedia* to create our test set. Since there is no template or matchmaker offered by this Web site, we crawl the information of most popular widgets using our own XSL template and ontology matchmaker. We get 1715 widgets in total and consequently generate 1715 RDF snippets. It is hard to know which users downloaded which widgets since this kind of information is private, and normally Web sites will keep this confidential. On the other hand, dispatching widgets randomly is not realistic, since users tend to just install widgets associated with their interests. Therefore, we assume for simplicity that each peer initially owns a single widget. After dispatch, the average number of triples about the widget stored on each peer is 15.6. Most widget publishers, especially those who are professional, use natural language to give a brief introduction (description) about their widgets. So we also make use of both features and descriptions to do the simulation of group discovery and the comparison of two discovery methods. We discuss the selection of the distribution threshold $dist_{min}$ defined in Section 4 experimentally. We run our group discovery programme 100 times, starting with the threshold from value 0.01 and increasing it by 0.01 each time. Then we get the changing process of the number of discovered groups as depicted in Figure 4. From this figure, when $dist_{min} \geqslant 0.65$, the group structures become stable. We take 0.8 as the value of $dist_{min}$. Finally, all the peers in the requester's contact list are classified into 22 groups based on features and 25 groups based on descriptions respectively. Table 2 and Table 3 describe the group discovery results (*GN* stands for *Group Name*; *NOP* stands for *Number of Peers*; *P* stands for *Percentage*).

[4] `http://www.forum.nokia.com`
[5] `http://wiki.opensource.nokia.com/projects/Mobile_Web_Server`

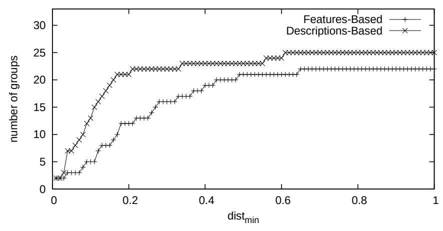


Fig. 4. Discovered groups with the change of $dist_{min}$

Though description-based discovery (Table 3) contains 3 more groups than feature-based discovery (Table 2), there are 100 more widgets that can not be grouped in Table 3 than in Table 2. Moreover, some of the *GN*s in Table 3 are obscure, e.g., *Save Time*, *List* and so on. The reason for this is because natural language is more flexible but less uncontrollable and the descriptions are more heterogenous compared to features. We can also see this from *Blog* and *Blog Website* in Table 3, which actually have the same meaning. Feature-based group discovery outperforms description-based group discovery. According to Table 2, 627 peers, accounting for 36.56% of the contact peers, finally fall into *other group*. They do not associate with other peers very well. Studying the RDF snippets, we find that most of them use unusual words (or phrases) to describe their features. Moreover, a few of them lack feature information or are very badly tagged using just one or two unrepresentative words (or phrases).

For the format transformation part, we take a widget with unknown format as an example and give a case study of changing it to Nokia S60 widget supported by Symbian 60 Feature Pack 2. Needless to say, the target template should be the one for Nokia S60 widgets. After being unpacked, the original widget is converted to a bundle of files. This bundle contains a configuration file named *info.plist* whose content will be compared to the templates in the template repository. Then the original format will be identified by the comparison result, e.g., a Dashboard widget. The *Mapping Module* uses our widget ontologies to associate the elements in the original files with the ones in the target template. For example, the property *CFBundleIdentifier* from a Dashboard widget is corresponding to the property *Identifier* from a Nokia S60 widget and the property *CFBUndleVersion* from a Dashboard widget is corresponding to the property *Version* from a Nokia S60 widget. Based on the above semantic alignments, the values in the original widget are filled in the target template in the end. Sometimes, the corresponding target elements will not be found after the mapping process and this means the target widget engine does not have these functionalities that the original widget engine has. In this case, these elements will be ignored by the *Mapping Module*. Alternatively, this kind of elements can be made up manually. Under this circumstance, our transformation approach will still save a great deal of manpower. Figure 5 illustrates the transformation from a prevailing Dashboard widget *iStatpro* (top subfigure) to a widget running on Nokia S60 FP2 (bottom subfigure). Within the process of creating onologies, we do our best to guarantee the flexibility and extendability of the hierarchy. The current widget ontologies we use is still in its preliminary stage, which contain 17 classes and 57 properties (20 object properties and 37 data type properties).

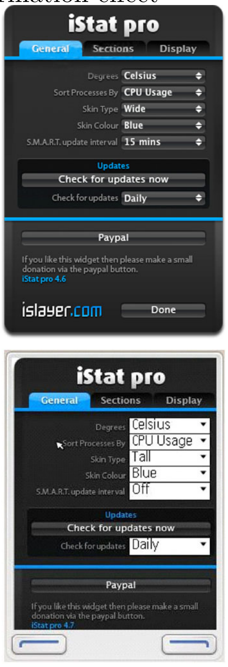
Table 2. Features-based group discovery

GN	*NOP*	*P*
Dashboard	278	16.21%
News	202	11.78%
Search	198	11.54%
NASA	19	11.1%
Blog	114	6.65%
Music	113	6.59%
Fun	108	6.30%
Clock	103	6.01%
Amazon	94	5.48%
Gadget	88	5.13%
eBay	80	4.66%
Games	78	4.55%
Google	70	4.08%
Video	55	3.21%
Shopping	53	3.09%
Mp3	50	2.92%
Radio	45	2.62%
Songs	37	2.16%
TV	35	2.04%
Stock	17	0.99%
Test	15	0.87%
Other	627	36.56%

Table 3. Descriptions-based group discovery

GN	*NOP*	*P*
Search	249	14.52%
News	184	10.73%
Blog	136	7.93%
Website	126	7.35%
Dashboard	105	6.12%
Clock	96	5.60%
Web	91	5.31%
New	85	4.96%
Game	84	4.90%
Blog Website	76	4.43%
Show	61	3.56%
eBay	61	3.56%
Widget Displays	58	3.38%
Videos	52	3.03%
List	31	1.81%
Watch Live	30	1.75%
Gadget Search	27	1.57%
View the Latest	17	0.99%
Save Time	11	0.64%
Sidebar Gadget	8	0.47%
TV	8	0.47%
Music looked for	6	0.35%
Social Networking	6	0.35%
Music Slide	4	0.23%
Other	727	42.39%

Fig. 5. Widget transformation effect



Degrees Celsius
Sort Processes By CPU Usage
Skin Type Wide
Skin Colour Blue
S.M.A.R.T. update interval 15 mins
Check for updates Daily
If you like this widget then please make a small donation via the paypal button.
iStat pro 4.6

Degrees Celsius
Sort Processes By CPU Usage
Skin Type Tall
Skin Colour Blue
S.M.A.R.T. update interval Off
Check for updates Daily
If you like this widget then please make a small donation via the paypal button.
iStat pro 4.7

7 Conclusions and Future Work

We propose an approach for sharing and reusing widgets between mobile devices and also adapt it in P2P environment. Based on domain ontologies and semantic clustering, a light-weight group discovery approach is presented for pruning the search space and cutting down the bandwidth limited by each peer. We also propose a framework for transforming widgets with diverse formats and give a case study to demonstrate it. After being grouped, a peer can more efficiently find others that have similar interests with itself in the pruned search space. Also, these basic groups trigger the initial formation of the peer community. Any peer can invite friends who have similar interests to join its group and then benefit the overall community. Our future work includes adding multi-language support within group discovery. A peer ranking approach should be applied to our widget transferring process for possibly enhancing the widget search. In order to expedite search, generated RDF files should be indexed, which is also a challenge. Currently we are working on the evolvement of discovered groups, including peer-membership update (entering or leaving a group) and group merging.

Acknowledgement

The research has been supported by the Open Knowledge project (FP6-027253).

References

1. Martin, D., et al: OWL-S: semantic markup for web services. W3C Member Submission. Available at `http://www.w3.org/Submission/OWL-S/`, 2004.
2. Akkiraju, R.: Web service semantics - WSDL-S (Version 1.0). Available at `http://www.w3.org/Submission/WSDL-S/`, 2005.
3. Farrell, J., Lausen, H.: Semantic annotations for WSDL and XML schema. W3C Recommendation. Available at `http://www.w3.org/TR/2007/REC-sawsdl-20070828/`, 2007.
4. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with OWLS-MX. In: Proceedings of the Internatinoal Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'06), pp. 915C922, ACM Press, 2006.
5. Klusch, M., Kapahnke, P.: Semantic web service selection with SAWSDL-MX. In: Proceedings of the International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMR^2'08) on ISWC'08, pp. 3-18, 2008.
6. Getting started with nokia web widget development (Version 1.0). Available at `http://www.forum.nokia.com/info/sw.nokia.com/id/7df9d735-3fe8-4719-aeef-219a85d13552/Getting_Started_with_Nokia_Web_Widget_Development.html`, 2007.
7. Promote your website's content using gadgets. Available at `http://www.google.com/webmasters/gadgets/guidelines.html`, 2007.
8. Caceres, M.: Widgets 1.0 requirements. Available at `http://www.w3.org/TR/2007/WD-widgets-reqs-20070209/`, 2007.
9. Robertson, D.: Multi-agent coordination as distributed logic programming. In: Proceedings of the International Conference on Logic Programming (ICLP'04), LNCS 3132, pp. 416-430, Springer-Verlag, 2004.
10. Noy, N.F., Chugh, A., Liu, W., Musen, M.A.: A framework for ontology evolution in collaborative. In: Proceedings of the International Semantic Web Conference (ISWC'06), LNCS 4273, pp. 544-558, Springer-Verlag, 2006.
11. Angles, R., Gutierrez, C.: The expressive power of SPARQL. In: Proceedings of the International Semantic Web Conference (ISWC'08), LNCS 5318, pp. 114-129, Springer-Verlag, 2008.
12. McCarey, F., Cinnéide, M.Ó., Kushmerick, N.: Recommending library methods: an evaluation of the vector space model (VSM) and latent semantic indexing (LSI). In: Proceedings of the International Conference on Software Reuse (ICSR'06), LNCS 4039, pp. 217-230, Springer-Verlag, 2006.
13. Schürmann, K.B., Stoye, J.: Counting suffix arrays and strings. In: Proceeding of the String Processing and Information Retrieval (SPIRE'05), LNCS 3772, pp. 55-66, Springer-Verlag, 2005.
14. Ferraro, P., Godin, C.: An edit distance between quotiented trees. *Algorithmica*, vol. 36, no. 1, pp. 1-39, Springer New York, 2008.
15. Khambatti, M., Ryu, K.D., Dasgupta, P.: Structuring Peer-to-Peer networks using Interest-based communities. In: Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'04) on VLDB 2004, LCNS 2944, pp. 48-63, Springer-Verlag, 2004.