Research on Ontology-Based Usable User Interface Layout Approach

Shengyang Luo, Yinglin Wang*, Jianmei Guo
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai 200240, China
*Corresponding author, ylwang@sjtu.edu.cn

Abstract—**Usability has become a key factor in the success of a software product, and constructing an appropriate user interface layout is an important way to enhance the software usability because a well-organized user interface is needed to support the task associated with it. This paper proposed an automatic user interface layout approach for the ontology-based reconfigurable system according to the structural principle of the usable user interface design guidelines. It firstly maps the components in the user interface into the concepts in the ontology, and then divides them into groups based on the concepts' similarities and relationships, finally identifies the position of the components in the user interface according to its significance to the task associated with the user interface. Experiments show that the approach is helpful in the usability design of software systems.**

Keywords-ontology; user interface; layout; usability;

I. Introduction

In an ontology-based system, the domain ontology is usually not hardcoded in the system, but is allowed to change according to the requirements, such as changing the concept s and the type of them [1]. Because the domain ontology can be customized and the related user interface, such as instance-input and instance-search interface, depends on the type of concept in the ontology, so the system must have the ability of automatically generating user interface for new concepts. For example, Protégé [2] automatically generates an instance-input interface for every concept in the ontology. In the process of user interface design, usability is considered as the core of product competitiveness because it directly determines the actual use effect of the product [3]. However, most of the systems based on ontology focus on the ontology modeling; few efforts were made on the user interface's usability, especially in the process of automatic user interface generation.

Larry L. Constantine [4] concluded a series of usable user interface design guidelines, and the structural principle is considered as one of the most basic principles used to guide the user interface layout. Structural principle emphasizes on the clear and consistent user interface model, which should organize the components in a meaningful and useful way, putting the related things together and distinguishing different things. Furthermore, Larry L. Constantine used visual coherence and layout appropriateness to measure the degree of structure. Visual coherence emphasizes that the semantics-related components should be laid together, and layout appropriateness emphasizes that the component which is the most significant to the task associated with the user interface should be put on the place where users concern mostly. However, Larry L. Constantine did not give a computing method to generating user interface layout according to structural principle, which must be done manually by the user interface designer.

In the ontology-based system, user interface can be derived from domain ontology. Macías and Castells [5] created a template for each concept of the ontology. A template defined what attributes must be included in the presentation, and their appearance. The dynamic user interface was generated from those templates. Ben Liu [6] presented an approach to derive user interface from ontology. It automatically generates user interface according to declarative model specifications. Similar to Macías and Castells's, in this paper we defined a component for each attribute of the concept in the ontology. The novelty of our approach is of the user interface layout in the following two aspects:

- We measure the semantic relativity of the components in the interface based on concept similarity and relationship, and then put the semantic-related components into the same group.
- We associate each concept with a task, and then calculate the significance of each attribute of the concept to the task; finally put the component associated with the most significant attribute on the place the users concern mostly.

The motivation of our work is briefly introduced in section 2. The relevant definitions are given in section 3. The implementation algorithm of automatic user interface layout is discussed in section 4. An example is analyzed in section 5. The conclusion and future research directions are discussed finally.

II. Motivation

We have built an ontology-based reconfigurable knowledge management system (ORKMS) [7]. Users can create new type of knowledge in the runtime, whose user interface is automatically generated by the system. Similarly, most ontology editing tools, such as Protégé, generate a default user interface form for every concept in the ontology, based on the

978-1-4244-4738-1/09/$25.00 ©2009 IEEE

attributes of the concept and their attribute value types [2]. For example, if we build a simple ontology about goods management shown in Figure 1, the instance-input form for Invoice can be automatically generated by Protégé or by ORKMS.

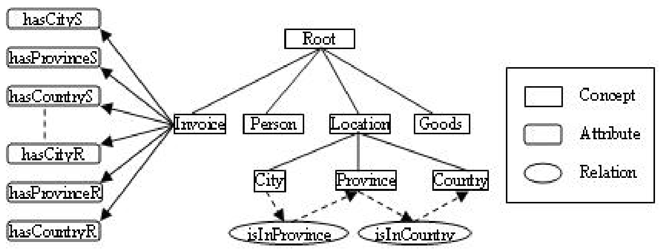

Figure 1. A simple ontology about goods management

However, the automatically generated interface may not be good enough for easy use. E.g., sender's locations represented by *hasCountryS*, *hasProvinceS*, and *hasCityS* may not be grouped together. In order to address this problem of inconvenience, we conceived a new user interface layout approach based on concept similarity and relationship in the ontology. In the following sections, we will elaborate on how to realize the idea.

III. Definitions

Ontologies were originally developed in Artificial Intelligence to facilitate knowledge sharing and reuse. An ontology is a shared conceptual representation of domain knowledge that provides a common understanding of a domain [8]. A knowledge base based on domain ontology is a structure

$$KB := (C, S, R, I, \prec_C, \sigma_S, \sigma_R, \ell_C, \ell_S, \ell_R)$$

consisting of

- four sets C, S, R and I whose elements are called concepts, signatures, relations, and instances, respectively,
- a partial order $\prec_C$ on C called concept hierarchy. For example, $C_1 \prec_C C_2$ indicates that C_1 is a subconcept of C_2,
- a function $\sigma_S : S \rightarrow C\times C$ called concept signature,
- a function $\sigma_R : R \rightarrow C\times C$ called concept relation,
- a function $\ell_C : C \rightarrow \xi(I)$ called concept instantiation,
- a function $\ell_S : S \rightarrow \xi(I \times I)$ called signature instantiation,
- a function $\ell_R : R \rightarrow \xi(I \times I)$ called relation instantiation.

To facilitate description, denote $\pi_i(\sigma)$ as the *i-th* concept in $C\times C$ in function σ. For a signature $s \in S$, we define its domain and its range by $dom(s) := \pi_1(\sigma(s))$ and $range(s) := \pi_2(\sigma(s))$, where $\pi_1(\sigma(s))$, $\pi_2(\sigma(s)) \in C$. Take the goods management ontology in Figure 1 for example. We define a concept signature σ_S : *hasCityS* $\rightarrow$ *Invoice* $\times$ *City*, whose domain and range are *Invoice* and *City* respectively, which means that every invoice has a sending city attribute.

For a relation $r \in R$, we define a direction from $\pi_1(\sigma(r))$ to $\pi_2(\sigma(r))$, which means that there is a semantic link from $\pi_1(\sigma(r))$ to $\pi_2(\sigma(r))$. Take the goods management ontology in Figure 1 for example too. We define a concept relation σ_R : *isInProvince* $\rightarrow$ *City*$\times$*Province*, which means that every city is located in some province. The difference between concept signature and concept relation is that, the former is used to describe a concept by other concepts, and the latter is used to describe the relationship between two concepts.

In the system, we define T_C as a type of task associated with a concept C, such as inputting and displaying concept instances. In order to complete task T_C, the attributes of C will be used, which are also concepts too. Therefore, the concepts used by task T is a set

$$Set_{Tc} := \{range(s) \ : for\ s \in S,\ where\ dom(s)\ equals\ to\ C\}$$

From the point of the task T_C, some of concepts in Set_{Tc} are more significant, while others are relatively less significant. So we define three types of significances: *Unique, Required, Optional,* which indicate that for each task of type T_C, the instantiated value of the concept C' in Set_{Tc} is unique, required, and optional, respectively. We define a property $P_{C'}$ for each C' in Set_{Tc}, whose value is one of the following: *{Required, Unique}, {Required}, {Optional, Unique}, {Optional}.*

Finally, we define the elements on the user interface for each task as a set of components: $UI_{Tc} := \{Component(C') : for\ all\ C' \in Set_{Tc}\}$, where *Component(C')* indicate the component associated with concept *C'*.

IV. Implementation

From the point of the task associated with the user interface, a well-organized user interface should put the most relevant content at the place where users can easily access, so that users can quickly complete their tasks. Layout appropriateness emphasizes that the arrangement of the components should minimize the time needed to complete the task. In this paper, the components are divided into groups based on the relevance of concepts, including concept similarity and relationship.

A. Concept similarity in the ontology

In an ontology, the similarities of concepts can be measured according to the taxonomy. The similarity measures based on the path distance between two concepts in the tree is the most common approach [9]. The Palmer's measure can be used as an example of the approach [10]:

$$Sim(C_1, C_2) = \frac{2\times H}{N_1 + N_2 + 2\times H}$$

where *N1* and *N2* are the numbers of edges from *C1* and *C2* concepts to the most specific common concept *C*, and *H* is the number of edges from *C* concept to the root of the ontology. From the formula we can see that: (1) the longer distance path is between C_1 and C_2, the less similarity in taxonomy is between them; (2) the longer distance path is between the root of the ontology and the concept *C*, the more similarity in taxonomy is between C1 and C2 concepts. Take the ontology in section 2 for example; we calculate the similarity between *Country* and *City* concepts, and *Country* and *Person* concepts:

$$Sim(Country, City) = \frac{2\times 1}{1+1+2\times 1} = 0.5$$

$$Sim(Country, Person) = \frac{2 \times 0}{2 + 1 + 2 \times 0} = 0.0$$

It shows that the similarity between the concepts *Country* and *City* is greater than the similarity between the concepts *Country* and *Person*. From the intuitive, both *Country* and *City* are *Location* concepts, but *Country* and *Person* only have a common concept at the root of the ontology, so the fact holds. Specially, while C_1 and C_2 are the same concept, we conclude that $Sim(C_1, C_2)=1$.

B. *Concept relationship in the ontology*

In the ontology, besides the similarity between concepts in taxonomy, there are other relations between them. For example, a concept may consist of, depend on, or include another concept. We describe such relations between concepts by function $\sigma : R \rightarrow C \times C$, and $\ell_R : R \rightarrow \xi(I \times I)$ after concept instantiation.

In the ontology-based system, the establishment of such relations is not only necessary to maintain the consistency of ontology, but also helpful for building usable user interface. Take the concept *Invoice* in section 2 for example. It has two attributes *hasCountryS* and *hasProvinceS*, representing the country and province of the sending adress, respectively. On the instance-input user interface associated with concept *Invoice*, we map these two attributes into combobox. Suppose that there are 10 countries, and there are 30 provinces in each country, then there will be 10 items in the country's combobox and 300 items in the province's combobox. On one hand, assume that if we haven't built the following relations:

$\sigma : isInCountry \rightarrow Province \times Country$

$\ell_{isInCountry}: R \rightarrow Invoice.hasProvinceS \times Invoice.hasCountryS$

After users select a country from the sending country combobox, there are only 30 provinces are legal to select, but there are still 300 provinces in the sending provinces combobox! Users must be very careful to select the correct one from these 300 provinces, which will cost them much time. On the other hand, if such relation exists, we can remove the other 290 irrelevant items from the sending provinces combobox after the user select one country from the sending country combobox, which will be convenient for use.

C. *Algorithm*

In our approach, first of all, we divide the components into groups based on the concepts' similarity and relationship, and each component only belongs to one group. The relations between concepts are stronger than the similarity between concepts, because the relation between concepts is more specific after it is instantiated. Therefore, our strategy of division consists of two steps. In the first step, the components are divided based on concepts' similarity, and in the second step, the components are re-arranged based on concepts' relationships. Then, the groups of components are arranged on the user interface based on the importance of the groups to the task.

In the first step of the division, we set a similarity threshold λ_{TS}. If the similarity between concepts is equal to or greater than λ_{TS}, they will be divided into the same group. The value of λ_{TS} is set according to the specific task. The similarity threshold λ_{TS} should not be too large or too small in case that every group only consists of a component or all the components are divided into the same group. Since the components have been roughly divided into groups based on concept similarity, in the second step, we only consider relationships between concepts. Consider that there is a relation between two concepts. If the two concepts were in different groups before, we remove the two concepts from the original groups and then make a new group for them, or else we do nothing. We should note that the relation is transitive. All the direct and indirect related concepts are divided into the same group. Finally, for all the relations which are from C_1 to C_2, we put C_1 in front of C_2.

```
Algorithm: Layout
Input:  Task T and the concept similarity threshold $\lambda_{TS}$.
Output:  User interface $UI_{Tc}$
G ← GroupedByTS(T, $\lambda_{TS}$)      // division based on concept similarity
G' ← GroupedByRS(T, G)       // division based on concept relationship
S[1…|G'|] ← 0
foreach g in G'              // calculate significance of group of component
    $Score[g] \leftarrow \sum_{\forall c \in g} Score(c) / |g|$
end for
sort all the $g \in$ G' in non-ascend order according to S[g].
foreach g in G'
    add Component(g) to $UI_T$     // add group component
    foreach c in g
        add Component(c) to Component(g)   // add component
    end foreach
end foreach
return $UI_T$
```

Figure 2. Algorithm of layout

```
Algorithm: GroupedByTS
Input: Task T and the concept similarity threshold $\lambda_{TS}$.
Output: Array of group of component G based on concepts' similarity.
let C[1…n] be the concepts used by task T
for i ← 1 to n          // If concept similarity is equal to or greater than $\lambda_{TS}$,
    add c to G[i]       // put them in the same group. In this step,
    for j ← i + 1 to n      // In this step, one concept may be put more
        if Sim(C[i], C[j]) ≥ $\lambda_{TS}$ then add C[j] to G[i]  // than one group
    end for
end for
for i ← 1 to n          // If one concept is divided into two groups, calculate
 for j ← i + 1 to n   // the similarities of the concept in these two groups,
    g ← G[i] ∩ G[j]  // and remove it from the group with smaller similarity
    if g ≡ G[i] then delete G[i] from G
    else if g ≡ G[j] then delete G[j] from G
    while g ≠ Ø
        fetch c from g and delete c from g
        $s_i \leftarrow \sum_{\forall c' \in G[i] \wedge c' \notin g} Sim(c, c') / (|G[i]| - |g|)$
        $s_j \leftarrow \sum_{\forall c' \in G[j] \wedge c' \notin g} Sim(c, c') / (|G[j]| - |g|)$
        if $s_i \geq s_j$ then delete c from G[j]
        else delete c from G[i]
    end while
 end for
end for
return G
```

Figure 3. Algorithm of GroupedByTS

```
Algorithm:  GroupedByRS
Input: Task T and array of group of component G based on similarity.
Output: array of group of component G based on concepts' relationship.
1  $S_R \leftarrow$ all relations between concepts used by task T
2  $G_R \leftarrow \emptyset$
3  foreach r in $S_R$            // If there is a relation in two concepts, divide them
4     $c_1 \leftarrow \pi_1(\sigma(r))$         //  into the same group.
5     $c_2 \leftarrow \pi_2(\sigma(r))$
6     if $G_R(c_1) \equiv \emptyset \wedge G_R(c_2) \equiv \emptyset$ then add $\{c_1, c_2\}$ to $G_R$
7     else if $G_R(c_1) \equiv \emptyset \wedge G_R(c_2) \neq \emptyset$ then add $c_1$ to $G_R(c_2)$
8     else if $G_R(c_1) \neq \emptyset \wedge G_R(c_2) \equiv \emptyset$ then add $c_2$ to $G_R(c_1)$
9     else add $G_R(c_2)$ to $G_R(c_1)$ and delete $G_R(c_2)$ from $G_R$
10 end foreach
11 foreach g in G               // Combine $G$ and $G_R$ into $G'$
12    foreach $g_R$ in $G_R$
13       $g' \leftarrow g \cap g_R$
14       if $g' \equiv g$ then add $g_R$ to G'
15       else if $g' \equiv g_R$ then add g to G'
16       else delete g' from g and add both g and $g_R$ to G'
17    end foreach
18 end foreach
19 foreach g in G'   // For each relation from $C_1$ to $C_2$, put $C_1$ in front
20    if exists $c1 \in g \wedge c2 \in g \wedge r(c1, c2) \in S_R$ then  // of $C_2$
21       if $c_1 \rightarrow c_2$ then place $c_1$ in front of $c_2$
22       else place $c_2$ in front of $c_1$
23 end foreach
24 return G'
```

Figure 4. Algorithm of GroupedByRS

After the components are divided into groups, we calculate the significance of each group of component to the task, and then sort them in non-ascending order. Define the significance of a group G as a score: $Score_G = \sum_{\forall C_i \in G} Score_{C_i} / N_G$, where N_G is the number of concepts in group G, and

$$Score_{C_i} = \begin{cases} 1.00 & P_{C_i} = \{required, Unique\} \\ 0.75 & P_{C_i} = \{required\} \\ 0.50 & P_{C_i} = \{Optional, Unique\} \\ 0.25 & P_{C_i} = \{Optional\} \end{cases}$$

The algorithm *Layout* shown in Figure 2 organizes the components in the user interface. The algorithm *GroupedByTS* in Figure 3 and *GroupedByRS* in Figure 4 divide the components into groups based on concept similarity and relationship, respectively.

V. An Example

In the following, we will give an example for further explanation. Firstly, we consider the detail of the concept *Invoice* in section 2, as shown in Figure 5.

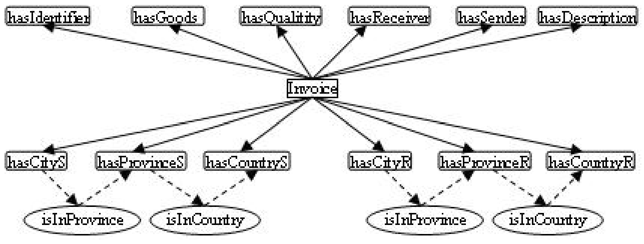

Figure 5. Detail of concept Invoice in section 2

Here, the task *T* associated with concept *Invoice* is to input its instances, and all the attributes of *Invoice* are used by task *T*. The type of each attribute and its significance to task T is as shown in table 1.

TABLE I. Description of each attribute of concept Invoice

Attribute	Type	Sigificance	Description
hasIdentifier	String	Unique, Required	Identifier
hasGoods	Goods	Required	Instance of Goods
hasQuantity	Number	Required	Number of Goods
hasReceiver	Person	Required	Receiver
hasSender	Person	Required	Sender
hasDescription	String	Optional	Remarks
hasCityS	City	Required	Sending city
hasProvinceS	Province	Required	Sending province
hasCountryS	Country	Required	Sending country
hasCityR	City	Required	Receiving city
hasProvinceR	Province	Required	Receiving province
hasCountryR	Country	Required	Receiving country

The user interface is defined using a textual language based on ASP.NET pages, that allows the system to dynamically insert C Sharp expressions (between <%= and >) into HTML code in the runtime. Each component associated with the attribute of concept *Invoice* is represented by a row consisting of two cells in a table. The first cell contains the label of the attribute, and the second cell contains the value of the attribute. For example, attribute *hasSender* is defined as following:

<tr>

*<td><%= [**hasSender**]%></td>*
*<td><input type=<%= TypeFunc([**hasSender**])%>*
*size=<%=SizeFunc([**hasSender**])%></input></td>*
</tr>

TypeFunc is a function mapping the attribute into a real component. For example, we can define a textbox for attribute *hasSender*, as well as a combobox. The former allows users input any legal value into the textbox, and the latter only allows users select one value from the combobox. *SizeFunc* is a function calculating the length of the real component. The length of a component is related to the type of attribute, for example, the length of a textbox associated with an attribute of type of string whose maximum length is 20 will be set to 20, while the length of a textbox associated with an attribute of type of string whose maximum length is 30 will be set to 30.

On the user interface, we organize the components from top to down. The more significant the component is to the task, the more frontal place the component is on the user interface. Finally, the similarity threshold is set to 0.5. According to the algorithms given in section 4.3, after the first step of division based on concept similarity, we get the following groups:

G_1={hasIdentifier}, G_2={hasGoods}, G_3={hasQuantity}, G_4={hasDescription}, G_5={hasSender, hasReceiver}, G_6={hasCountryS, hasCountryR, hasProvinceS,hasProvinceR, hasCityS,hasCityR}

After the second step of division based on concepts' relationship, we divide group G_6 into two new groups G_7 and G_8:

G_7={*hasCountryS, hasProvinceS, hasCityS* }
G_8={*hasCountryR, hasProvinceR, hasCityR*}

Attributs *hasCountryS*, *hasProvinceS*, and *hasCityS* are divided into a new group G_7 because that there is a relation *isInCountry* between *hasProvinceS* and *hasCountryS*, and a relation *isInProvince* between *hasCityS* and *hasProvinvceS*. Attribute *hasCountryS* is put in front of *hasProvinceS* because that the value of *hasCountryS* will restrict the range of the value of *hasProvinceS*. The attribute *hasProvinceS* is put in front of *hasCityS* because of the same reason. The situation is similar in group G_8. Then, there is nothing left in group G_6, so we delete it.

Finally, we calculate the significance of each group. The result is as follows.

$G_1=1.00,\ G_2=G_3=G_5=G_7=G_8=0.75,\ G_4=0.25$

So, when they are organized on the user interface, group G_1 will be put at the most top position, and G_4 will be at the most bottom position, and G_2, G_3, G_5, G_7, G_8 will be in the middle.

The final user interface is shown in Figure 6. Compared with the user interface generated automatically but without considering the similarities and relationships between concepts and attributes, we can see that 1) the attributes *hasCountryS*, *hasProvinceS*, and *hasCityS*, which represent the sending locations are put together, as well as the attributes *hasCountryR*, *hasProvinceR*, and *hasCityR*, which depict the destination locations; 2) the required and unique attribute *hasIdentifier* is put in the most top position, and the optional attribute *hasDescription* is put at the most bottom position. The former is the most accessible by users, while the latter is the least accessible by users.

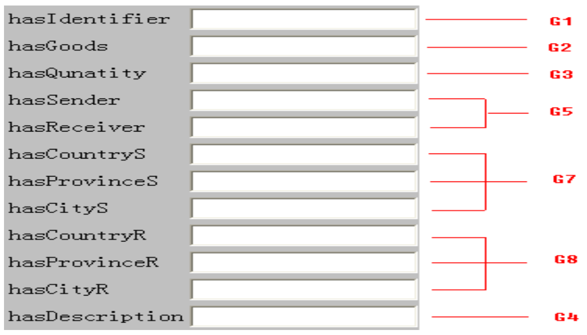

Figure 6. The layout of user interface generated by the algorithms

VI. Conclusion

Structural principle is used to guide the user interface layout. Visual coherence and layout appropriateness are two aspects of structural principle. Visual coherence emphasizes that the semantics-related components should be put together, and layout appropriateness emphasizes the most significant component should be put on the place where users can easily access.

In this paper, we propose an automatic user interface layout approach for the ontology-based system. We realize visual coherence by dividing components into groups based on the concepts' similarity and the concepts' relationship, and realize layout appropriateness based on the significance of the concept used in the task. A drawback of our approach is that the user model is not considered. Next, we will further improve the usability of the user interface with the user model.

Acknowledgment

The paper was sponsored by the National High-tech Research and Development Project of China (863) under the grant No.2009AA04Z106 and the National Science Foundation of China (NSFC) under the grant No.60773088.

References

[1] Y. Wang, J. Guo, T. Hu, et al., "An Ontology-Based Framework for Building Adaptable Knowledge Management Systems," 2nd International Conference on Knowledge Science, Engineering and Management, Springer, Melbourne, Australia, 2007, pp. 655-660.

[2] Protégé 2000, http://protege.stanford.edu/.

[3] Jakob Nielsen. Usability Engineering, Morgan Kaufmann, San Francisco, 1994, pp. 15-45.

[4] Larry L.Constantine, Lucy A.D.Lockwood. Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design, Addison-Wesley Professional, 1999, pp. 32-42, 296, 306-308.

[5] Macías J A, Castells P, "Adaptive Hypermedia Presentation Modeling for Domain Ontologies," 9th International Conference on Human-Computer Interaction, New Orleans, Louisiana, USA, 2001, Lawrence Erlbaum Associates, Publishers. London. Vol. 2, pp. 710-714.

[6] Ben Liu, Hejie Chen, Wei He, "Deriving User Interface from Ontologies: A Model-based Approach," Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence, Hong Kong, 2005, pp. 254-259.

[7] Yinglin Wang, Weidong Wang, Zongjiang Wang, et al., "Reconfigurable Platform for Knowledge Management Based on Ontology," Computer Integrated Manufacturing System, 2003, vol .9, no. 12, pp.1136-1144 (in Chinese).

[8] Gruber, T. R., "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," in Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Press, Boston, 1993, pp. 907-928.

[9] Jie cheng, Zhu-hua Jiang. Concept Similarity Computation for Domain Ontology [A], Computer Engineering and Applications, 2006, vol. 33, pp.163-166 (in Chinese).

[10] Paweł Lula and Grażyna Paliwoda-Pękosz, "An Ontology-Based Cluster Analysis Framework," Proceedings of 1st International Workshop on Ontology-Supported Business Intelligence, 2008.