Usage–Centric Adaptation of Dynamic E–Catalogs

Hye-young Paik, Boualem Benatallah, and Rachid Hamadi

School of Computer Science and Engineering, The University of New South Wales
Sydney, NSW, 2052, Australia
{hpaik,boualem,rhamadi}@cse.unsw.edu.au

Abstract. Although research into the integration of e-catalogs has gained considerable momentum over the years, the needs for building adaptive catalogs have been largely ignored. Catalogs are designed by system designers who have a priori expectations for how catalogs will be explored by users. It is necessary to consider how users are using catalogs since they may have different expectations. In this paper, we describe the design and the implementation of a system through which integrated product catalogs are continuously adapted and restructured within a dynamic environment. The adaptation of integrated catalogs is based on the observation of customers' interaction patterns.

1 Introduction

In recent years, integration of e–catalogs has gained considerable momentum because of the emergence of online shopping portals, increasing demand for information exchange between trading partners, prevalent mergers and acquisitions, etc [10]. In approaches that address the problem of e–catalogs organisation and integration, a product catalog is usually structured in a category–based hierarchy [10,7]. Catalogs are designed in a "one–view–fits–all" fashion, by a system designer who has a priori expectations for how catalogs will be "explored" by customers. However, the customers may have different expectations. Therefore, it is necessary to take into consideration how the customers are using the catalogs to continuously minimise the gap between expectations of the system designer and customers. For example, in a catalog for computer parts, assume that it is repeatedly observed that many users always use product category `RAM` right after using category `CPU`. If the administrator merges the two categories and creates a new category `CPU&RAM`, users now only need to visit this new category once for information of both products.

In this paper, we describe the design and the implementation of a system, called $WebCatalog^{Pers}$, through which existing online product catalogs can be integrated and the resulting integrated catalogs can be continuously adapted and restructured within a dynamic environment. The catalogs integration framework used in this paper originates from a previous project on integration of Web data, called *WebFINDIT* [4]. Based on this framework, we propose a usage–centric

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 344–360, 2002.

technique for transforming catalogs organisation. It should be noted that the focus of this paper is not on catalogs integration. The objective is to *continuously* improve the organisation of catalogs by being responsive to the ways customers navigate them in searching for products. The proposed approach offers the following features: (i) *Catalog navigation and access model* – this model provides a set of actions, called catalog interaction actions, that users would perform while accessing catalogs, (ii) *Catalog transformation operations* – these operations are used to transform the structure and organisation of catalogs, and (iii) *Predefined sequences of catalog interaction actions* – these sequences represent pre–identified interaction patterns of users. They can be considered as heuristics for catalog transformations. Discovery of these patterns help administrators decide what kind of transformations would be desirable to improve the organisation of catalogs. Transformations of a catalog over time, result in offering improved alternative of its organisation based on user interaction patterns.

The remainder of this paper is organised as follows. Section 2 overviews the design of $WebCatalog^{Pers}$. Section 3 presents a formal model for integrated catalogs and user interaction actions. The catalog reorganisation operations, predefined interaction sequences (PISs), and the confidence of a PIS are introduced in Sect.4, Sect.5, and Sect.6 respectively. Section 7 presents the results of simulation studies. Finally, Sect. 8 discusses related work and concludes the paper.

2 $WebCatalog^{Pers}$: Design Overview

In this section, we give the intuition behind the main concepts that are used in $WebCatalog^{Pers}$, namely, catalog communities and eCatalogs–Net. The formalisation of these concepts will be presented in the next section.

2.1 Catalog Communities

A catalog community[1] is a container of catalogs which offer products of a common domain (e.g., community of `Laptops`). It provides a description of desired products without referring to actual sellers (e.g., `a seller of IBM Laptops`). We illustrate catalog communities with *computers and related services* domain (see Fig.1).

There are two types of relationships defined between catalog communities: *SubCommunity–Of* and *PeerCommunity–Of*. SubCommunity–Of relationships represent *specialisation* between domains of two catalog communities (e.g., `Printer` is a sub–community of `Peripherals`). We assume that, each catalog community has at most one super–community. PeerCommunity–Of relationships are viewed as a referral mechanism in that when the user can not find (or is not satisfied with) information from a catalog community, s/he can refer to other communities that the catalog community consider as its peers (e.g., community `Display` is a peer community of `VideoCard`). It should be noted that,

[1] We use the terms *catalog community* and *community* interchangeably.

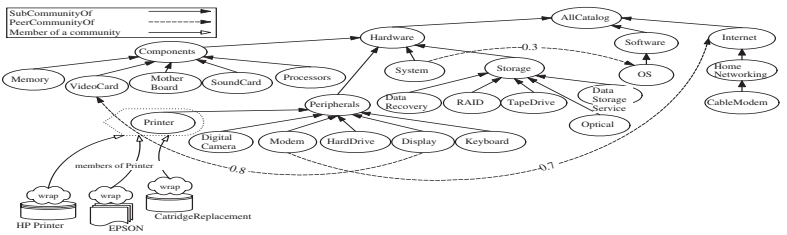

Fig. 1. eCatalogs–Net: Organising catalog communities

we do not assume that the opposite (i.e., `VideoCard` is a peer community of `Display`) systematically holds. A weight (a real value between 0 and 1) is attached to each PeerCommunity–Of relationship to represent the degree of relevancy as a peer. Note that communities can also forward queries to each other via PeerCommunity–Of relationship. We call this organisation of catalog communities *eCatalogs–Net*. Any catalog community that is not a sub–community of any other community is related to `AllCatalog` via SubCommunity–Of relationship.

Each catalog community has a set of attributes that can be used to query the underlying catalogs. We refer to the set of attributes as *community product attributes*. For example, catalog community that represents "CD–Readers and Writers" would have community product attributes such as `Maker`, `Read-WriteSpeed`, `Price`, etc.

2.2 Catalog Registration

In order to be accessible through a community, product sellers need to register their catalogs with the community. A catalog provider is known to a community by providing (i) a wrapper, (ii) an exported interface, and (iii) a mapping between exported interface and community product attributes. The wrapper translates $WebCatalog^{Pers}$ queries to local queries, and output of the local queries are translated back to the format used by $WebCatalog^{Pers}$. The exported interface defines the local product attributes for querying information at the local catalog. A local catalog supplier also should provide operations, such as ordering or payment for the products. However, the focus of this paper is not on specifying transactional operations. Detailed description on provisioning such operations in the context of Web services is presented in [1]. Users may use a community to express queries that require extracting and combining product attributes from multiple underlying product catalogs (e.g., price comparison). We refer to this type of queries as *global queries*. Global querying is achieved by using community product attributes which do not directly correspond to product attributes. Therefore, when a product catalog is registered with a community, the catalog provider should also define mapping between local product attributes and community attributes. We call this mapping *Source–Community mapping*. Note

that a community can be registered with another community. By doing so, the members of the first community also become members of the second community.

2.3 Searching and Querying Product Information

Users in $WebCatalog^{Pers}$ will typically be engaged in two–step information–seeking activity: (i) navigating communities for product catalogs location and semantic exploration (e.g., get communities that are relevant to selling laptops) and (ii) querying selected communities or catalogs for products information (e.g., compare product prices). Users would have a specific task to achieve (e.g., product items they wish to purchase, a category of products they want to investigate) when using product catalogs. We assume that they use the following strategy[2] :

1. Start at the root (i.e., `AllCatalog`), or at a specific community (if they know the location of the catalog community).
2. While (current community C is not the target community T) do
   (a) If any of the SubCommunity–Of relationships of C seems likely to lead to T, follow the relationship that appears most likely to lead to T.
   (b) Else, if any of the PeerCommunity–Of relationships of C seems likely to lead to T, follow the relationship that appears most likely to lead to T.
   (c) Else, either backtrack and follow SuperCommunity–Of relationship of C, or give up.

Once the user has reached the target, s/he will submit a query to the target. If the user ends up in the same community again in step 2(a) or 2(b), s/he will follow a different relationship, since her/his reasoning of which relationship is likely to lead to the target has changed by then.

3 Modelling Catalog Communities and User Interaction

In this section, we present a model for formally representing communities, eCatalogs–Net, and consistency of eCatalogs–Net. The model also identifies a set of actions that users can perform when interacting with the eCatalogs–Net. The proposed model forms the basis for defining catalog restructuring operations and user interaction patterns (see Sect. 4 and Sect. 5).

3.1 eCatalogs–Net

We give the definition of a community first and then eCatalogs–Net.

Definition 1 (Catalog Community). *A catalog community C is a tuple* $\mathtt{C} = (\mathtt{NameC}, \mathtt{GeneralInfo}, \mathtt{CommunityProductAttr}, \mathtt{Members})$ *where:*

– `NameC` *is the name of the community* `C`,

[2] [13] uses a similar strategy for browsing and searching Web documents.

- `GeneralInfo` *is a set of pairs* $(\mathtt{p}, \mathtt{v})$*, where* `p` *is a property of the community and* `v` *is a value of* `p` *(e.g., (Domain, "CD Writers")),*
- `CommunityProductAttr` *is a set of attribute-type pairs* $(\mathtt{att}, \mathtt{type})$*, where* `att` *is a community product attribute (i.e., a global attribute) and* `type` *is the type of* `att` *(e.g., ("ModelNumer", Integer)),*
- `Members` *is a set of members. A member can be either a product catalog or another catalog community and is defined as a pair* $(\mathtt{mid}, \mathtt{map})$ *where* `mid` *represents the identifier of the member and* `map` *contains the Source–Community mapping.* □

Definition 2 (eCatalogs–Net). *An eCatalogs–Net is a labelled directed graph* $\mathtt{G} = (\mathtt{N}, \mathtt{E}_1, \mathtt{E}_2, \mathtt{W}, \ell)$*, where:*

- $\mathtt{N}$ *is a finite set of nodes. A single node represents a catalog community,*
- $\mathtt{E}_1 \subseteq \mathtt{N} \times \mathtt{N}$ *is a finite set of directed edges (representing SubCommunity–Of),*
- $\mathtt{E}_2 \subseteq \mathtt{N} \times \mathtt{N}$ *is a finite set of directed edges (representing PeerCommunity–Of),*
- $\mathtt{W} : \mathtt{E}_2 \rightarrow [0, 1]$ *is a weighting function (initially each edge in* $\mathtt{E}_2$ *receives a neutral weight of 0.5), and*
- $\ell : \mathtt{N} \rightarrow \mathcal{C}$ *is a naming function where* $\mathcal{C}$ *is a set of catalog community names.* □

To be *consistent*, an eCatalogs–Net must satisfy the conditions given in the definition below:

Definition 3 (Consistent eCatalogs–Net). *The eCatalogs–Net* $\mathtt{G} = (\mathtt{N}, \mathtt{E}_1, \mathtt{E}_2, \mathtt{W}, \ell)$ *is consistent if and only if the following conditions are satisfied:*

1. *The naming function* ℓ *is injective (that is, there will not be two communities with the same name),*
2. *The graph* $\mathtt{G}'_1 = (\mathtt{N}, \mathtt{E}_1^{-1}, \ell)$ *(generated from the sub-graph* $\mathtt{G}_1 = (\mathtt{N}, \mathtt{E}_1, \ell)$ *of* $\mathtt{G}$ *by inverting the edges) is a tree. The root of the tree is* `AllCatalog`*,*
3. $\mathtt{E}_2 \cap (\mathtt{E}_1 \cup \mathtt{E}_1^{-1})^+ = \varnothing$ *(where* $\mathtt{E}^+$ *denotes the transitive closure of E, i.e.,* $(\mathtt{i}, \mathtt{j}) \in \mathtt{E}^+$ *iff there is a directed path from* `i` *to* `j` *in* $\mathtt{E}$*).* □

3.2 Permissible User Actions

The permissible actions, noted $\mathcal{A}$, for exploring eCatalogs–Net are listed in Table 1. By modelling user interaction actions, the system can capture them for future use.

Every time a user invokes one of the permissible actions at a catalog community, $WebCatalog^{Pers}$ keeps that event in the system log file. The log file is, later, organised into *sessions* and for each `SubmitQuery` action in a session, all of the product attributes selected by the query are identified. A session in $WebCatalog^{Pers}$ is an ordered sequence of actions performed by a single user, where the time difference between any two consecutive actions in the sequence should be within a time threshold, $\mathtt{T}_{\mathtt{threshold}}$ defined by an administrator.

Table 1. Permissible User Actions $\mathcal{A}$ in eCatalogs–Net

Action Name	Description
`NavigateToSub(Community c)`	The user goes from the current catalog community to one of its sub–communities `c`.
`NavigateToSuper()`	The user goes from the current catalog community to its super–community.
`NavigateToPeer(Community c)`	The user goes from the current catalog community to one of its peer communities `c`.
`LeaveCatalogCommunity()`	The user leaves the current catalog community. The user is taken to `AllCatalog`.
`ShowMembers(Constraint s)`	The user requests to show members of the current catalog community satisfying the constraint `s`.
`SubmitQuery(Query q)`	The user submits the query `q` to the current catalog community. It could be a global query which uses the community product attributes, or a source query which concerns one member of the community.

4 Restructuring eCatalogs–Net

We now describe a set of restructuring operations on eCatalogs–Net. These operations are used, for example, to change the relationships between catalog communities, remove a catalog community, or merge catalog communities. They can be performed at an administrator's own discretion. In the next section, we will introduce predefined interaction sequences which provide means to observe the user's interaction patterns. The observation will help decide which operation to perform in order to improve the organisation of the eCatalogs–Net.

An operation is applied to a consistent eCatalogs–Net $\mathtt{G} = (\mathtt{N}, \mathtt{E}_1, \mathtt{E}_2, \mathtt{W}, \ell)$ and produces a consistent eCatalogs–Net $\mathtt{G}' = (\mathtt{N}', \mathtt{E}'_1, \mathtt{E}'_2, \mathtt{W}', \ell')$. For space reasons, we do not give detailed description of each operation. We only describe operations for merging and splitting catalog communities. It should be noted that each high level operation is defined as a sequence of primitive operations. The primitive and high level restructuring operations are summarised in Table 2.

Moving a Community. The operation `moveCatComm()` moves a community `c` from one place to another, by changing its super–community. This operation is used, e.g., when an administrator is convinced that the current super–community of c does not represent the domain of products in c properly. For example, in Fig.1, assume that the community `HardDrive` is sub–community of `Peripherals` and the user navigation behaviour shows that community `Storage` is more suitable super–community for `HardDrive`. This may suggest that it is beneficial to move `HardDrive` to `Storage`. When a community `c` is moved, all of its sub–communities are moved with it. Having this assumption creates less overhead,

Table 2. eCatalogs–Net Restructuring Operations

Primitive Operations
`setCatalogName(Community c, String n)` : Set the name of `c` to `n`.
`addPeer(Community` c_i`, Community` c_j`)`: Add PeerCommunity–Of from c_i to c_j.
`delPeer(Community` c_i`, Community` c_j`)`: Delete PeerCommunity–Of from c_i to c_j.
`updatePeer(Community` c_i`, Community` c_j`, Weight w)`: Update the weight of PeerCommunity–Of from c_i to c_j by `w`.
`addSub(Community` c_i`, Community` c_j`)`: Add SubCommunity–Of from c_i to c_j.
`delSub(Community` c_i`, Community` c_j`)`: Delete SubCommunity–Of from c_i to c_j.
`createCatComm(Name n, GeneralInfo gi, Members m, CommunityProductAttr gs)`: Create a new catalog community with the information given. `create-CatComm` must be followed by `addSub` operation.
`superCatComm(Community c)`: Return the super–catalog community of `c`.
`subCatComm(Community c)`: Return a set of catalog communities which directly have SubCommunity–Of relationship with `c` (direct sub–communities).
`indSubCatComm(Community c)`: Return a set of catalog communities which, directly or indirectly, have SubCommunity–Of with `c` (indirect subcommunities).
High Level Operations
`mergeCatComm(Community` c_i`, Community` c_j`, Name n)`: Merge two existing communities c_i and c_j and set the name of the new catalog community to `n`.
`splitCatComm(Community c, GeneralInfo gic, Name n, GeneralInfo gi, CommunityProductAttr cpa, Query q, setOfCommunities sub)`: Split catalog community `c` into two separate communities. `gic` contains new specification of `GeneralInfo` for `c`. `n`, `gi`, and `cpa` contain specification of the new community (`Name`, `GeneralInfo`, and `CommunityProductAttr` respectively). `q` is a query which will be used by the operation to select members to be moved from `c` to the new community. `sub` is a set of sub–communities to be moved to the new one.
`delCatComm(Community c)`: Remove the catalog community `c` from eCatalogs–Net. Used, for example, when a community becomes obsolete (e.g., has no useful existence inside the eCatalogs–Net).
`moveCatComm(Community` c_i`,Community` c_j`)`: Move c_i to new super–community c_j.

since sub–communities of `c` do not get affected by the change. The effects of this operation are described in Fig.2.

The community `E` is moved from its super–community `B` to the new super–community `C`. `E`'s sub–communities, i.e., `G` and `H` remain as sub–communities of `E`. However, since a catalog community cannot be a peer of its super–community, the PeerCommunity–Of relationship from `H` to `C` has to be deleted.

Merging Communities. The operation `mergeCatComm()` merges two communities `c` and c' which have the same super–community[3]. It is used, e.g., when

[3] Note that, in this paper, we only consider merging of two communities, but the operation can be generalised to more than two communities.

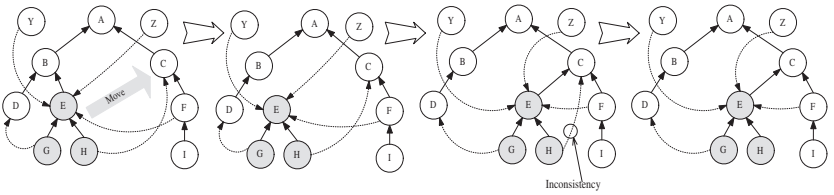

Fig. 2. Moving a catalog community

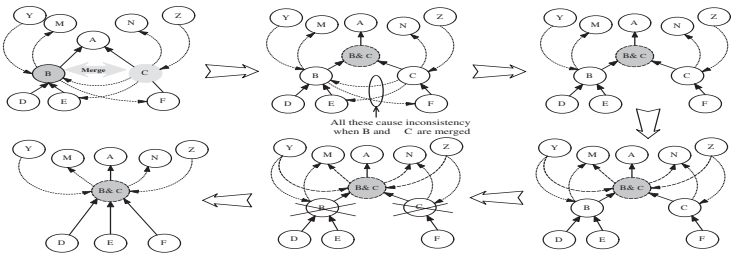

Fig. 3. Merging two catalog communities of the same super–community

it is observed that the two catalog communities `c` and `c′` are always accessed together. Hence, it is beneficial that these two catalog communities are merged, so that the majority of users do not have to visit two separate communities each time. Figure 3 illustrates the effects of `mergeCatComm()`.

It shows that a new community is created from merging communities `B` and `C`. The super-community of the new community is the super-community of `B` and `C` (i.e., `A`). All sub–communities of `B` and `C` (i.e., `D`, `F` and `G`) are sub-communities of the new community. All PeerCommunity–Of relationships between `B` and `C`, as well as between `B` and all of `C`'s sub–communities, `C` and `B`'s sub–communities should be deleted to maintain the consistency of the eCatalogs–Net. Also, all PeerCommunity–Of relationships coming from other communities into `B` and `C` need to be updated, i.e, the PeerCommunity–Of relationships would refer to the name of the new community, instead of `B` and `C`.

Splitting a Community. The operation `splitCatComm()` splits an existing catalog community into two separate communities. This operation is used, e.g., when it is observed that the community represents a domain (described by community product attributes) which can be divided into smaller sub-domains. This situation is illustrated in Fig.4. Note that as a result of split, one new community is created out of an existing one. The definition of the existing community is updated to reflect this change (e.g., remove community product attributes, or members that have been moved to the new community).

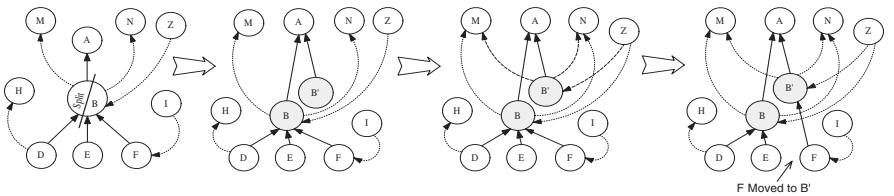

Fig. 4. Splitting a catalog community

Figure 4 illustrates that when the community `B` is split, a new community `B′` is created out of `B`. All incoming and outgoing PeerCommunity–Of relationships of `B` are inherited by `B′`. Also, if it is necessary, some of the sub communities of `B` can be moved to `B′`.

The split of an existing catalog community needs careful consideration about "how" each element in the community should be treated. It is an administrator's responsibility to decide how the attributes `GeneralInfo, CommunityProductAttr, Members` and SubCommunity–Of relationships should be initialised. This information is specified via the operation parameters (see Table 2).

5 Predefined Interaction Sequences

Predefined interaction sequences represent foreseeable user's interaction behaviour, therefore can be predefined. In our approach, we use these sequences of actions to help identify situations where the organisation of an eCatalogs–Net may be improved through restructuring operations. Any particular sequence of actions with prevalent occurrences should be recognised as a recurring user interaction pattern. Each interaction pattern identified suggests a restructuring operation. A *Predefined Interaction Sequence* (PIS) is formally defined as follows:

Definition 4 (Predefined Interaction Sequence (PIS)). *A predefined interaction sequence PIS of length n* $(n > 0)$ *is a vector of ordered user actions* $\texttt{PIS} = \langle a_1, a_2, ..., a_n \rangle$ *where* $a_i \in \mathcal{A}$ *(see Table 1)* $(i = 1, .., n)$. □

For a given PIS, there may exist a session s such that the exact order of actions in PIS can be found in s. A predefined interaction sequence is matched against each session in the processed log file to check whether the sequence exists in the session. We refer to the number of occurrences of a PIS in the log file as *Frequency* (see Sect. 6).

In the following subsections, we present a set of predefined interaction sequences. We describe some of the PISs in details. The rest are listed in Tables 3 and 4. We use the action `SubmitQuery` as the most appropriate action in indicating user's strong interests in a community. However, an administrator may decide to choose other actions (or define new ones, e.g., PurchaseItem) for the same purpose.

Table 3. Other Predefined Interaction Sequences

Predefined Interaction Sequences
`Deleting a community` : Identify a community from which users are constantly leaving without performing any further action. $\texttt{PIS}_{\texttt{delComm}} = \langle\ \texttt{a}(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j}), \texttt{LeaveCatalogCommunity}(\texttt{c}_\texttt{j})\rangle$, where $\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j} \in \texttt{N}$, $(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j}) \in \texttt{E}_1 \cup \texttt{E}_1^{-1}$, and $\texttt{a} \in \{\texttt{NavigateToSub}, \texttt{NavigateToSuper}\}$.
`Merging communities (Case 1)`: Identify two sub–communities of the same super–community which are always accessed together (not via PeerCommunity–Of). $\texttt{PIS}_{\texttt{merge1}} = \langle\ \texttt{SubmitQuery}(\texttt{c}_\texttt{i}, \texttt{q}_1), \texttt{NavigateToSuper}(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{k}), \texttt{NavigateToSub}(\texttt{c}_\texttt{k}, \texttt{c}_\texttt{j}), \texttt{SubmitQuery}(\texttt{c}_\texttt{j}, \texttt{q}_2)\ \rangle$, where $\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j}, \texttt{c}_\texttt{k} \in \texttt{N}$ and $(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{k}), (\texttt{c}_\texttt{j}, \texttt{c}_\texttt{k}) \in \texttt{E}_1$.
`Merging communities (Case 2)`: Identify a catalog community and its super community are always queried together. $\texttt{PIS}_{\texttt{merge2}} = \langle\ \texttt{SubmitQuery}(\texttt{c}_\texttt{i}, \texttt{q}_1), \texttt{a}(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j}), \texttt{SubmitQuery}(\texttt{c}_\texttt{j}, \texttt{q}_2)\ \rangle$, where $\texttt{a} \in \{\texttt{NavigateToSuper}, \texttt{NavigateToSub}\}$, $\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j} \in \texttt{N}$, and $(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j}) \in \texttt{E}_1 \cup \texttt{E}_1^{-1}$.
`Merging communities (Case 3)`: Same as $\texttt{PIS}_{\texttt{merge1}}$, but uses `NavigateToPeer` $\texttt{PIS}_{\texttt{merge3}} = \langle\ \texttt{SubmitQuery}(\texttt{c}_\texttt{i}, \texttt{q}_1), \texttt{NavigateToPeer}(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j}), \texttt{SubmitQuery}(\texttt{c}_\texttt{j}, \texttt{q}_2)\ \rangle$ where $\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j} \in \texttt{N}$ and $(\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j}) \in \texttt{E}_2$.

5.1 Merging Communities

Here, we introduce a generic sequence that describes situations where merging of communities may be beneficial. We identify some interesting sequences which represent special cases of the generic sequence[4].

Definition 5 ($PIS_{GenericMerge}$)**.** $\texttt{PIS}_{\texttt{GenericMerge}}$ *which represents the situations where two communities are always queried together is:*

$$\texttt{PIS}_{\texttt{GenericMerge}} = \langle\ \texttt{SubmitQuery}(\texttt{c}_\texttt{i}, \texttt{q}_1), \texttt{a}_1, ..., \texttt{a}_\texttt{n}, \texttt{SubmitQuery}(\texttt{c}_\texttt{j}, \texttt{q}_2)\ \rangle$$

where $\texttt{c}_\texttt{i}, \texttt{c}_\texttt{j} \in \texttt{N}$, $\texttt{a}_\texttt{k} \in \{\texttt{NavigateToSub}, \texttt{NavigateToSuper}, \texttt{NavigateToPeer}\}$ $(k = 1, .., n)$, *and* $\texttt{q}_1, \texttt{q}_2$ *are global query attributes (i.e., community product attributes).* □

$\texttt{PIS}_{\texttt{GenericMerge}}$ captures interaction sequences where users, within a catalog community $\texttt{c}_\texttt{i}$, first submit a query then perform several navigation actions to reach a community $\texttt{c}_\texttt{j}$ from where they finally submit another query. Figure 5 presents three particular cases of $\texttt{PIS}_{\texttt{GenericMerge}}$.

5.2 Splitting a Community

A catalog community may be split if a subset of community product attributes are always queried together and the subset can represent a specific domain by

[4] Note that, even though actions in Table 1 do not include source catalog community parameters, we add them when defining PIS for clarity reasons.

Table 4. More PISs (on PeerCommunity–Of relationship)

Predefined Interaction Sequences
`Upgrading the weight of a PeerCommunity-Of`: Consolidates the relevancy of the relationship. Consider that many users navigate from community $\mathtt{c_i}$, via PeerCommunity-Of relationship, to community $\mathtt{c_j}$, and submit a query to $\mathtt{c_j}$. This indicates that the PeerCommunity – Of relationship from $\mathtt{c_i}$ to $\mathtt{c_j}$ positively contributed in finding the target community. $\mathtt{PIS_{upgrade}} = \langle\ \mathtt{NavigateToPeer}(\mathtt{c_i}, \mathtt{c_j}), \mathtt{SubmitQuery}(\mathtt{c_j}, \mathtt{q})\ \rangle$, where $\mathtt{c_i}, \mathtt{c_j} \in \mathtt{N}$, $(\mathtt{c_i}, \mathtt{c_j}) \in \mathtt{E_2}$, and $\mathtt{q}$ is global query attributes.
`Downgrading the weight of a PeerCommunity-Of`: Consider that many users who followed a PeerCommunity–Of relationship and arrived at a community $\mathtt{c_j}$, ultimately leave the community without performing any further action. This may indicate that $\mathtt{c_j}$ is not relevant to these users. To leave $\mathtt{c_j}$, use `LeaveCatalogCommunity` or `NavigateToPeer`. $\mathtt{PIS_{downByLeave}} = \langle\ \mathtt{NavigateToPeer}(\mathtt{c_i}, \mathtt{c_j}), \mathtt{LeaveCatalogCommunity}(\mathtt{c_j})\ \rangle$, where $\mathtt{c_i}, \mathtt{c_j} \in \mathtt{N}$, and $(\mathtt{c_i}, \mathtt{c_j}) \in \mathtt{E_2}$. $\mathtt{PIS_{downByPeer}} = \langle\ \mathtt{NavigateToPeer}(\mathtt{c_i}, \mathtt{c_j}), \mathtt{NavigateToPeer}(\mathtt{c_j}, \mathtt{c_i})\ \rangle$, where $\mathtt{c_i}, \mathtt{c_j} \in \mathtt{N}$, and $(\mathtt{c_i}, \mathtt{c_j}), (\mathtt{c_j}, \mathtt{c_i}) \in \mathtt{E_2}$.
`Creating a new PeerCommunity--Of`: Identify communities that are constantly used as stop-overs. It may be beneficial to create direct PeerCommunity–Of relationship so that users can by-pass them. $\mathtt{PIS_{createPeer}}$ represents a situation where there are one or more navigational actions between NavigateToPeer and SubmitQuery. This suggests the creation of Peer-Community–Of between $\mathtt{c_i}$ and $\mathtt{c_k}$. $\mathtt{PIS_{createPeer}} = \langle\ \mathtt{NavigateToPeer}(\mathtt{c_i}, \mathtt{c_k}), \mathtt{a_1}, .., \mathtt{a_n}, \mathtt{SubmitQuery}(\mathtt{c_j}, \mathtt{q})\ \rangle$, where $a_p \in \{\mathtt{NavigateToSub}, \mathtt{NavigateToSuper}, \mathtt{NavigateToPeer}\}$ $(p = 1, .., n)$, $\mathtt{c_i}, \mathtt{c_j}, \mathtt{c_k} \in \mathtt{N}$, $(\mathtt{c_i}, \mathtt{c_k}) \in \mathtt{E_2}$, and $(\mathtt{c_i}, \mathtt{c_j}) \notin \mathtt{E_1} \cup \mathtt{E_1}^{-1} \cup \mathtt{E_2}$.
`Deleting a PeerCommunity-Of`: No pattern specifically defined. We consider $\mathtt{PIS_{downByLeave}}$, and $\mathtt{PIS_{downByPeer}}$. When it is observed that the weight of a PeerCommunity–Of in a community reaches the lower threshold (given by an administrator), the relationship is considered to be irrelevant and can be removed.

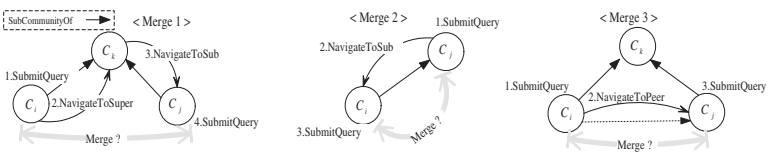

Fig. 5. Three particular cases of $\mathtt{PIS_{GenericMerge}}$

itself. One way to detect this situation is to observe the way the community product attributes are queried. The following pattern is used to identify a subset of attributes that are always queried together. In this pattern, an administrator has a specific catalog community in mind ($\mathtt{c_i}$) that s/he wants to examine for possibility of splitting and a set of attributes s/he predicts to be queried together.

Definition 6 (PIS_{split})**.** $\texttt{PIS}_{\texttt{split}}$ *which represents the pattern for splitting a catalog community is:* $\texttt{PIS}_{\texttt{split}} = \langle\ \texttt{SubmitQuery}(\texttt{c}_\texttt{i}, \text{“}\texttt{attr}_1, .., \texttt{attr}_\texttt{n}\text{”})\ \rangle$*, where* $\texttt{c}_\texttt{i} \in \texttt{N}$*, and* $\texttt{attr}_1, .., \texttt{attr}_\texttt{n}$ *are community product attributes in* $\texttt{c}_\texttt{i}$ *that are likely to be queried together.* □

6 Confidence of Patterns

In this section, we provide two definitions, namely *frequency* and *confidence* of a PIS. They are used to decide whether a PIS can be considered as a pattern for which a restructuring operation is suggested.

Definition 7 (Frequency). *A frequency of a PIS, denoted by* $\texttt{Frequency}(\texttt{PIS})$*, is number of occurrences of PIS in the processed log file.* □

The frequency of a predefined interaction sequence is used to decide whether the result of the match is significant enough to consider performing eCatalogs–Net restructuring operations. We discuss some of the issues that arise from using the patterns.

First, there is an issue of conflicting patterns where discovery of one pattern suggests a certain restructuring operation, whereas another pattern leads to a different operation on the same relationships or communities. For instance, it is possible that the pattern $\texttt{PIS}_{\texttt{upgrade}}$ shows that the weight of PeerCommunity–Of relationship between community A and B needs to be upgraded, but at the same time, the pattern $\texttt{PIS}_{\texttt{downByPeer}}$ may suggest that the same relationship should be downgraded.

On the other hand, there is an issue of knowing patterns that can consolidate each other. We refer to these patterns as consolidating patterns. These patterns, when used together, can reinforce each other's findings. For example, suppose that the pattern $\texttt{PIS}_{\texttt{downByLeave}}$ suggests that PeerCommunity–Of relationship between community A and B should be downgraded. When $\texttt{PIS}_{\texttt{downByPeer}}$ pattern also suggests downgrading of the same relationship, it helps choosing a restructuring operation with much more assurance. Table 5 lists the identified conflicting and consolidating patterns among the predefined interaction patterns presented in this paper.

Definition 8 (Confidence). *A confidence of a PIS denoted by Confidence(PIS) is defined as:*

$$\texttt{Confidence}(\texttt{PIS}) = \frac{\texttt{Frequency}(\texttt{PIS}) + \texttt{A}}{(\texttt{Frequency}(\texttt{PIS}) + \texttt{A}) + \texttt{B}} \quad \textit{where}$$

A is sum of frequency of all consolidating patterns of PIS and B is sum of frequency of all conflicting patterns of PIS. □

For a PIS to be considered, (i) its frequency should be greater than a frequency threshold and (ii) its confidence should be greater than a confidence threshold. Those two thresholds can be defined by an administrator.

Table 5. Conflicting and Consolidating Patterns

Name of PISs	downBy Leave	downBy Peer	create Peer	del Comm	merge1	merge2	merge3	split
upgrade	−	−	+	n	n	n	+	n
downByLeave	·	+	−	+	n	n	−	n
downByPeer	·	·	−	+	n	n	−	n
createPeer	·	·	·	−	n	n	n	n
delComm	·	·	·	·	−	−	−	−
merge1	·	·	·	·	·	n	n	−
merge2	·	·	·	·	·	·	n	−
merge3	·	·	·	·	·	·	·	−
split	·	·	·	·	·	·	·	·

Legend: n=no conflict, −=conflict, +=consolidation

7 Evaluation

The eCatalogs–Net (see Fig.1) used in the experiments represents an integrated view of 27 catalog communities in *computers and related services* domain. Overall, all components in $WebCatalog^{Pers}$ have been implemented using Java, JSP/Servlets, and JDBC. For persistent storage (log data and metadata repository), an XML-supported repository (Oracle 8i database) is used. The metadata repository stores information about community attributes, relationships, members, etc. It should be noted that our initial studies were conducted under simulated scenarios, in which, we restricted the users' interactions in terms of number of moves (e.g., mouse clicks) they can make. We show any measurable improvement by comparing the number of users who find the target (what they were looking for) before restructuring and after restructuring. The primary goal of this simulation study is to demonstrate that given the same constraint (i.e., limited number of moves), more users find targets after restructuring.

7.1 Experiment Framework

We used *task agents* that played the role of customers who wanted to find out information about the products. A Java class called *AgentFactory* was used to create agents. More precisely, the class AgentFactory implements a software component made up of a container and a pool of objects which represent agents. The container is a process that, once created, runs *continuously*, listening to a socket, through which an instantiation message from a predefined script (used to create an agent) is received. An agent interacts with a module called *Community Manager* (implemented as JavaBean) which provides various methods for exploring the community relationships (e.g., getSubCommunityOf(), getPeerCommunityOf() etc.). The agent's search and query behaviour is based on the same search and query strategy which is presented in Sect. 2.3. The

agents are equipped with two kinds of information for autonomous interaction with communities. First, the agents have access to the relationships (i.e., Sub, PeerCommunity–Of) between communities. The second information provided to the agents is called *Likelihood Table*. In the likelihood table, for a given a target community, every community in eCatalogs–Net is assigned a number value, which represents a degree of "closeness" (i.e., relevance) of the community to the target community. Hence, the higher the value, the *more likely* the community will lead the agent to the target. We will refer to this value as a *likelihood* and the list of likelihood values as a *likelihood table*.

Having the likelihood values fixed in the table makes the agents' interaction sequence to be always predictable. Agents should be able to make spontaneous and irregular decisions, resulting in unpredictable behaviour. We introduced a variant factor which would diverge a likelihood value. Each time, when an agent is given the likelihood values, the agent dynamically recalculates all likelihood values according to the factor before starting navigation.

The agent takes the following inputs to run; (1) name of the file that contains likelihood table, (2) name of the target community to find, (3) maximum number of moves an agent can make before giving up. For the purpose that stated earlier in Sect. 7, we limited the `MaxMove` to 14 for all experiments. The parameter `VF` (Variant Factor) represents the value of the variant factor for likelihood table. We asked four people who are familiar with the domain to produce the likelihood tables. The actual likelihood values used in the experiments took the average values of the four. The `VF` has three settings, 5%, 10% and 15%. Higher the VF, bigger the deviation from given likelihood values[5].

7.2 Experiments and Results

We now describe the results of experiments that investigated the effect of two restructuring operations (`addPeer, moveCatComm`) and the experiment parameter `VF`. The experiments carried out were based on two simulation scenarios. In the first scenario, we experimented on a PeerCommunity–Of relationship. For initial runs, 3000 agents were created and given the task of finding the community `CableModem` (see 'Before' in Fig.6). From the initial runs, observation showed that about 28% of the agents who found target followed the PeerCommunity–Of relationship from `Modem` to `Internet`, and `Internet`, `HomeNetworking` were used as stop-overs. We performed `addPeer` operation to create a new PeerCommunity–Of relationship from `Modem` to `CableModem`. Then we ran the 3000 agents again (see 'After' in Fig.6).

Varying VF: We measured the improvement made by the restructuring and study the effect of different values of the variant factor. We varied `VF` from 15% to 10%, and then to 5%. As shown in Fig.6, there were visible improvements in the number of agents found target. Irrespective of creation of the relationship,

[5] Note that other experimental parameters related to likelihood table have been defined, such as number of the tables participated in the experiment, range of likelihood values, etc. However, we only present the experiments with `VF`.

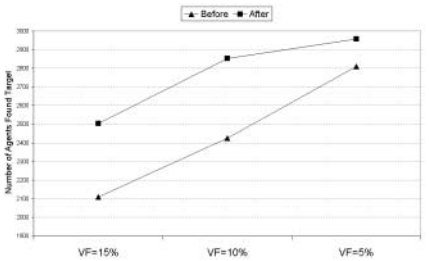

Fig. 6. Varying VF: First Scenario

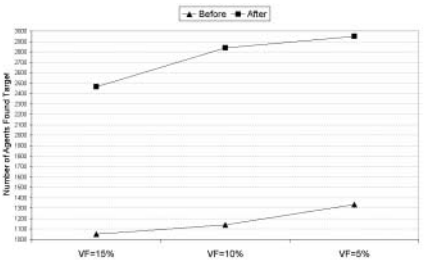

Fig. 7. Varying VF: Second Scenario

as `VF` decreases the more agents were able to find targets. `VF` randomises the likelihood values from a given table. This result demonstrates that agents are likely to find the target if their likelihood values are less deviated from the given likelihood values. Given the fact that the likelihood values used described the relationships of communities in relatively precise manner, this result can be interpreted that the user whose understanding does not deviate much from that of domain experts is more likely to find targets easily. Also, the biggest improvement was made when `VF` was 15 (i.e, having the highest deviation from given likelihood values). This indicates that the restructuring of eCatalogs–Net can benefit the most when the user's understanding deviates much from the expert.

In the second scenario, we experimented on moving a community to a new super–catalog community. In the initial structure of eCatalogs–Net (Fig.1), `HardDrive` is sub catalog community of `Peripherals`. The likelihood values used reflected the `Storage` as the expected location of the target. In the initial runs, 3000 agents were created and given the task of finding the community `HardDrives`. For the second runs, we performed `moveCatComm()` operation to move `HardDrives` from `Peripherals` to `Storage` and ran 3000 agents again. As shown in Fig.7, clear improvements were made after the restructuring.

Overall, we saw obvious improvements made after restructuring of eCatalog–Net across various experiment settings. This demonstrates that adaptive structuring of e–catalogs can help users have more streamlined and easier navigation/search experience. The experiments with `VF` parameter showed that for the users whose understanding deviates very much from the experts, can benefit the most from having communities restructured.

8 Related Work and Conclusions

We identify two major areas to discuss related work, namely building adaptive Web sites and navigation mining techniques in Web content personalisation. [11] automatically constructs index pages that supplement an existing organisation by looking at co–occurring pages, so that users can easily locate pages that are conceptually and strictly related to one topic. In [9], a technique that discovers

the gap between Web site designer's expectation and user's behaviour is proposed. The technique uses inter page conceptual relevance vs. inter page access co–occurrence. [13] developed an algorithm to identify "expected locations" of a Web page and create a link from the expected location to the page. [14] extract related pages from a given page, then investigate the relationships between two Web pages based on how each page drives other pages as related page. It is worth noting that, while basic principles of this area are complementary to our work, most approaches only deal with Web pages, which is quite different from the concept of communities we proposed. In our work, communities are individual and autonomous entities (rather than network of Web pages) with which users and members of the community can have various interactions (submitting a query to, invoking operations from, register with, etc.).

In the area of mining access patterns, [6] uses Web usage mining concept to dynamically predict user's next behaviour and to make a recommendation. [3] uses Hypertext Probabilistic Grammar also to predict the user's navigation path. [5,8] discuss issues and processes involved in preparation/transformation of data from Web server logs to a format suited for purpose of mining. In typical sequence or Web usage mining, an access pattern is a sequence of visited Web documents which have a large occurrence frequency. It extracts frequently visited nodes, or nodes that are visited together, but these kind of access pattern does not reflect how users navigate the imposed structure. [12,2] proposed a Web Usage Mining (WUM) system to evaluate effectiveness of the Web site organisation. It uses a concept of g-sequence to model sequence of navigation of users. We use a similar concept to model sequence of user interaction actions.

Another work worth mentioning is [15], in which decision trees are used to automatically construct catalogs based on popularity of product items (i.e., frequency of visits) and weighted product attributes. The algorithm of construction is designed in a way that the depth of product hierarchy (which is a tree) is minimised, pushing the popular product items/attributes to upper levels so that customers can find them easily (with fewer clicks). However, it does not discuss the ongoing adaptivity of the catalogs. Also, *WebFINDIT* [4] considers addition and deletion of links between communities. However, the mechanism is based on link-monitoring agents, which is different from mining user access patterns.

In summary, in this paper, we presented a usage-centric approach for transforming and improving integrated catalogs structure and organisation. We proposed catalog restructuring operations as well as predefined interaction sequences that help decide which operation to perform. We also illustrated the viability of the proposed approach and demonstrated that restructuring increase the chances of the user finding his/her targets through simulated experiments. Ongoing work includes case studies to evaluate $WebCatalog^{Pers}$ in a distributed environment. We also plan to extend the proposed approach by grouping people with similar interaction patterns.

References

1. B. Benatallah, M. Dumas, Q.Z. Sheng, and A.H.H. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In *Proc. of the International Conference on Data Engineering*, San Jose, California, February 2002. 346
2. B. Berendt and M. Spiliopoulou. Analysis of Navigation Behaviour in Web Sites Integrating Multiple Information Systems. *The VLDB Journal*, 9(1):56–75, 2000. 359
3. J. Borges and M. Levene. Data Mining of User Navigation Patterns. In *Proc. of the Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*, San Diego, CA, August 1999. 359
4. A. Bouguettaya, B. Benatallah, L. Hendra, M. Ouzzani, and J. Beard. Supporting Dynamic Interactions among Web-based Information Sources. *IEEE Transaction on Knowledge and Data Engineering*, 12(5):779–801, Sept/Oct 2000. 344, 359
5. R. Cooley, B. Mobasher, and J. Srivastava. Data Preparation for Mining World Wide Web Browsing Patterns. *Journal of Knowledge and Information Systems*, 1(1), 1999. 359
6. A. Datta, K. Dutta, D. VanderMeer, K. Ramamritham, and S. B. Navathe. An Architecture to Support Scalable Online Personalization on the Web. In *Proc. of the 2nd ACM Conference on Electronic Commerce (EC'00)*, Minneapolis, Minnesota, October 2000. 359
7. J. Jung, D. Kim, S. Lee, C. Wu, and K. Kim. EE-Cat: Extended Electronic Catalog for Dynamic and Flexible Electronic Commerce. In *Proc. of the IRMA2000 International Conference*, Anchorage, Alaska, May 2000. IDEA Group Publishing. 344
8. B. Mobasher, R. Cooley, and J. Srivastava. Automatic Personalization Based on Web Usage Mining. *Communications of the ACM*, 43(8), August 2000. 359
9. T. Nakayma, H. Kato, and Y. Yamane. Discovering the Gap Between Web Site Designers' Expectations and User's Behaviour. In *Proc. Of 9th International World Wide Web Conference*, Amsterdam, May 2000. 358
10. S. Navathe, H. Thomas, M. Satits Amitpong, and A. Datta. A Model to Support E-Catalog Integration. In *Proc. of the IFIP Conference on Database Semantics*, Hong Kong, April 2001. Kluwer Academic Publisher. 344
11. M. Perkowitz and O. Etzioni. Adaptive Web Sites. *Communications of the ACM*, 43(8), August 2000. 358
12. M. Spilopoulou. Web Usage Mining for Web Site Evaluation. *Communications of the ACM*, 43(8), August 2000. 359
13. R. Srikant and Y. Yang. Mining Web Logs to Improve Website Organization. In *Proc. of 10th International World Wide Web Conference*, Hong Kong, May 2001. 347, 359
14. M. Toyoda and M. Kitsuregawa. A Web Community Chart for Navigating Related Communitie. In *Proc. of 10th International WWW Conference*, Hong Kong, May 2001. 359
15. D. Yang, W. Sung, S. Yiu, D. Cheung, and W. Ho. Construction of Online Catalog Topologies Using Decision Trees. In *Proc. of Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000)*, Milpitas, California, June 2000. 359