Design and Implementation of Universal Web-Tree Service Component for Large-Scale Data Maintenance

Zhidong Wang*
Computer Academy, Beihang University, Beijing, China
Email: wzd2ff99@126.com

Lichao Ye
Beijing audit firm, Beijing, China
Email: ylc_1989@163.com

Wenfa Li
Beijing Union University, Beijing, China
Email: liwenfa@buu.edu.cn

***Abstract*—At present the third-part open source web-tree components and related algorithms have the shortcomings of poor compatibility, small data-scale and low running efficiency. The paper aims at design and implementation of a universal web-tree service component (WTSC) for larger-scale data maintenance to overcome these deficiencies mentioned above. By comparing and analyzing the existing algorithms, an improved algorithm named M-C was put forward to meet the demands of large-scale web-tree data loading, adding, removing and dragging. Then pseudo-code descriptions of M-C algorithm were given in detail. The paper gave a comprehensive formal description of WTSC to illustrate its abstract logic model. By using SCA, SOA, SSH, and UML technologies, software development framework named SOA-SSH, service access interface model named SCA and UML diagrams were given to develop and implement the application prototype of WTSC. To verify its performance, a comparative analysis was made among three different schemas. From the data test results, conclusions can be drawn that WTSC designed by this paper had better performance at large-scale data loading and maintenance, at the same time as a result of SCA model and SOA-SSH framework, it had friendlier UI and better compatibility and extensibility. Finally, future research work was prospected.**

***Index Terms*—SCA, SOA-SSH, large-scale, web-tree data**

I. INTRODUCTION

During a large number of B/S (Brower/Server) model MIS (management information system) software design and development, the problem of web-tree data maintenance is frequently encountered. Management of such data had common characteristics of frequently accessing to large amounts of data, complex maintenance operation (loading, querying, adding, deleting, modifying, dragging, dropping, moving, etc.). For example, in the development of web-based online material cataloging system, we must describe the classification relationships of thousands of materials through a directory tree. When the data-scale increased rapidly, due to the low running efficiency of data-loading, we had to wait too long for the directory tree building and displaying. Therefore, it is particularly critical for us to choose an appropriate data description method, realization algorithm and development framework by which we can improve the running efficiency and enhance the system scalability.

Now the widely used third-part open source web-tree components include DTree, XTree and ZTree. DTree is an open source tree component developed by JS (Java Script).It is particularly simple for the user to use the tree component according the label of the open new links and display icons. DTree is simple and practical, but it does not support the checkbox choosing and dynamic nodes adding, deleting and dragging. XTree is a menu of supporting properties and functions API based on Ajax [1]. It can customize icons and links object-oriented. Because of being described by XML, its structure is relatively simple. Compared with DTree, we can add, delete or drag nodes by calling API directly. ZTree is a multi-functional "plugging and playing" component based on jQuery. ZTree can supply the functions of selecting check-box, changing icon dynamically and providing a wide range of incident response calling back, but also can synchronize JSON[2] format data.

The technology system architecture, data format (JS, JSON, XML), and running environment were significantly different among these components mentioned above. It is difficult for us to exchange and share data received from these web-tree components.

In terms of data management, the main class web-tree data structure can be divided into two types, the static and dynamic. The former is applicable for the small scale and content-fixed tree-level data management. The advantage

This work was supported by the High Technology Research and Development Program of China (2007AA01Z416).Corresponding Author, Zhidong Wang.

doi:10.4304/jsw.9.7.1781-1789

of the static is that it is simple for us to program and disadvantage is difficult to reflect data changes rapidly. The latter can dynamically maintain the web-tree data and realize the real-time operation and centralized storage by the method of a certain database [3]. In terms of data loading, the static adopted "one-loading and many-running" policy. Its performance and efficiency completely depends on the data scale. When the scale reached a certain size (10^4 above), the time consuming for web-page loading became long. The dynamic commonly uses depth-first recursive algorithm. This algorithm is suitable for the small-scale web-tree data management. When the data size reached up to 10^5, the loading time is relatively long too. Jian Wang, Hongying Fang and Changchun Chen put up binary tree-based prefix code conversion algorithm [4]. Dejun Chen, Yingzhe Ma and Zhude Zhou put forward breadth-first traversal code-based algorithm [5]. The two algorithms had relatively good efficiency in data loading, but when it comes to data maintenance such as changing hierarchy, a large number of nodes involved being moved and running efficiency is low. When the data size of 10^6 or more, data-loading efficiency is extremely low.

To cope with these issues, we designed a universal platform-independent WTSC to realize the large-scale data loading and maintenance by an improved algorithm named M-C, SOA-SSH software development framework and its related technologies.

II. Algorithm Analysis and Design

A. Analysis of Existing Algorithms

The existing tree traversal algorithm mainly includes pre-order, in-order and post-order. We usually use pre-order. Common tree node coding algorithm include Inheritance-Coding (referred I-C), Relationship-Coding (referred R-C), and Left and Right Coding (referred L&R-C). A case study in cataloging materials tree shown as Fig.1, we can design database tables as TABLE I, TABLE II and TABLE III respectively.

TABLE I.
I-C

ID(PK)	TITLE
10	Commodity
1010	Food
101010	Meat
10101010	Pork
101011	Vegetable
10111110	Cabbage
1011	Electric
101110	Television
101111	Refrigerator
...	...

TABLE II.
R-C

ID(PK)	PID	LID	OID	TITLE
0	NULL	0	1	Root
1	0	1	1	Commodity
2	1	2	1	Food
3	2	3	1	Meat
4	3	4	1	Pork
5	2	3	2	Vegetable
6	5	4	1	Cabbage
7	1	2	2	Electric
8	7	3	1	Television
9	7	3	2	Refrigerator
...	...	...	...	...

TABLE III.
L&R-C

ID(PK)	LC	RC	TITLE
1	1	18	Commodity
2	2	11	Food
3	3	6	Meat
4	4	5	Pork
5	7	10	Vegetable
6	8	9	Cabbage
7	12	17	Electric
8	13	14	Television
9	15	16	Refrigerator
...	...	...	...

I-C: For using hierarchical coding values to indicate the parent-child relationships between tree nodes, child node code-value can be inherited from the parent. Advantages are as follows: the parent-child relationship expressed visually, with only one query we can get preorder a root node and all its descendants. By eliminating the recursion, when loading large-scale data, we can get high ruining efficiency. Furthermore, the layer number can be directly obtained from the operation of code value, using less storage fields, if the coding-bit is b and coding-value length is w, and then the layer number equals w/d. Disadvantages are as follows: the same layer node capacity is limited to the coding-digit, up to a maximum number of nodes in the same layer is 10^b-1. Poor reliability of the algorithm, when a node coding error prone fault. When the parent-child relationship was changed due to data dragging or moving, the associated child nodes should be re-encoded. As the number of nodes increases, computational complexity increases suddenly. Therefore, I-C algorithm can not be adapted to large-scale of web-tree data management.

R-C: The algorithm expressed parent-child relationship through a unique identifier. A node recorded itself ID, the parent node PID, the number of levels LID as well as its sibling node order OID at the same time. Advantages are as follows: for doing not have to follow the encoding rules, code values can be set to self-growth and parent-child relationships changes are simple, we just only change the node's PID, LID and SID value. Because of the smaller impact on other nodes, we can get the higher running efficiency. Disadvantages are as follows: when the data needed to be loaded or removed, recursive algorithm had to be involved. When the data-scale increased, the efficiency became lower especially the data size reached 10^4 or more.

L&R-C: We can set the left and right encoding-value for each node by a specific rule, which can show the parent-child relationship among tree nodes. By using the

preorder algorithm to traversal the tree nodes, the parent is visited firstly, then the child. Each node can be visited at twice. We can use increment marks (such as 1, 2, 3 and n) to tag the first traveling by left-value and the second right-value (shown in Fig.5). Advantages are as follows: we can achieve the unlimited classification by eliminating recursive. Adding, modifying, deleting, changing and other conventional operation had high efficiency. Because the query is based on the plastic figures, the large-scale data loading efficiency is high. Disadvantages are as follows: due to L&R-C is different from H-C or R-C, the parent-child relationship is not intuitive. The left and right code values of one node must be calculated by the preorder operation, the parent-child relationship is not obvious.

Considering the advantages and disadvantages of the three coding algorithms, the paper adopts a mixed coding algorithm (referred M-C) to implement the creating, loading and maintenance of large-scale web-tree data. We can create the data table as TABLE IV.

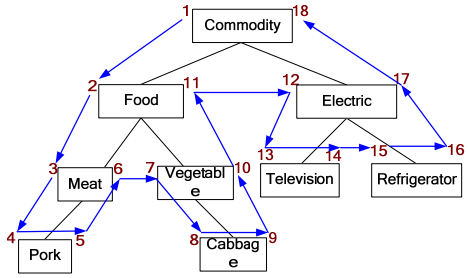

Figure 1. The left and right coding tree

TABLE IV.
MIXED-CODING

ID(PK)	PID	LID	LC	RC	TITLE
1	NULL	0	1	18	Commodity
2	0	1	2	11	Food
3	1	2	3	6	Meat
4	2	3	4	5	Pork
5	3	4	7	10	Vegetable
6	2	3	8	9	Cabbage
7	5	4	12	17	Electric
8	1	2	13	14	Television
9	7	3	15	16	Refrigerator
...	...	...	...	...	...

We can be seen from the TABLE IV: ID and PID, for building intuitive parent-child relationship, to make up for the shortcomings of L&R-C algorithm. We can set left and right code-value by pre-order traversal algorithm. The left-value of all child nodes of a specific parent node must be between its left-value and right-value. The number of all child nodes is equal to (rvalue-lvalue+1)/2. According to this law, we can query large-scale data, display or remove the unlimited hierarchical tree nodes quickly, eliminating the traditional recursive algorithm to make up the low operating efficiency of R-C.

B. Related Definitions and Terminologies

a) Definition 1: Tree Data Structure

Tree is a very important non-linear data structure, and it is generally suitable for describing the hierarchical structure, which can be defined as follows formula (1):

Tree includes n ($n \geqslant 0$) data elements (for each data element in the tree named a node) of a finite set. When n=0, we call it an empty tree; otherwise non-empty tree.

$$T = \begin{cases} \phi & n = 0 \\ \{R, T_1, T_2 ..., T_m\} & n > 0 \end{cases} \quad (1)$$

R is a special node. We call the node as a root with no precursor. Other nodes (T_1, $T_{2...}T_n$) as disjoint subsets, each subset is also a tree called a sub-tree. Fig.1. shows the tree of goods. Each data element of the tree is called a child node and the number of nodes of a sub-tree is called degree, terminal node called leaf. Detail content about tree structure can be found in [6].

b) Definition 2: Node of WTSC

According to Table IV, A node of WTSC can be defined by a six-tuple:

$$Node =< ID, PID, LFT, RGT, LID, TITLE > \quad (2)$$

According to definition 1, the node i can be denoted as $Node_i =< ID_i, PID_i, LFT_i, RGT_i, LID_i, TITLE_i >$.

Its parent can be expressed as:

$$Node_{i_p} =< ID_p, PID_p, LFT_p, RGT_p, LID_p, TITLE_p >$$

$$(ID_p = PID, LEF_i, RGT_i \in (LEF_p, RGT_p)). \quad (3)$$

c) Definition 3: Value of Tuple

We can get the value of the tuple by the index. The formula is as following:

$$ID_i = Node_i.get < Index_i > \quad (4)$$

d) Definition 4: Child Nodes of WTSC

We can denote a collection of a certain node and all its child nodes as following:

$$C_i = \{Node_k \mid LFT_k \in [LFT_i, RGT_i], k = 1,2...n,$$

$$n = (RGT_i - LFT_i + 1)/2\}. \quad (5)$$

e) Terminology 1: HQL

HQL (Hibernate Query Language) is Hibernate database query language based on the object-oriented database, which can package an object as a physical table [19].

f) Terminology 2: DAO

DAO (Data Access Object) can do a variety of operations such as adding, deleting, changing, checking by object-oriented design ideas package to access the physical database tables.

C. M-C Algorithm Design of WTSC

API functions involved in M-C algorithm are listed in TABLE V.

TABLE V.
M-C API

API Name	Function Description
1.batchUpdate_Nodes(String hql)	Batch updates data by hql.
2.get_Nodes (String hql)	Get nodes list by hql
3.get_Node (Integer i)	Get node by ID.
4.load_AllNodes (Integer i)	Load its all Childs.
5.show_Node(Integer i)	Output the Node
6.add_Node(Node node)	Add new node.
7.remove_Node (Integer i)	Remove all Childs.
8.drag_Node(Integer f, Integer t)	Drag one node to another.

Due to the limited length of the article, we only gave java pseudo-code descriptions of the node loading, adding, removing, and dragging operations.

a) Node-Loading Algorithm

We can load a commodity tree by the order numbers marked by blue arrows shown in Fig.1. By definitions of section B, the node-loading algorithm is summarized as algorithm I.

Algorithm I: load_AllNodes (Integer i)

1: *Node* $node_i$*= get_Node(i);*(Get $Node_i$ by M-C API 3)
2: *Integer* lft_i*=* $node_i$*.get*<2>*;*(Get left value of $Node_i$)
3: *Integer* rgt_i*=nodei.get*<3>*;*(Get right value of $Node_i$)
4: *String hql="from WebTree A where A.lft>="+* lft_i*+" and A.rgt<="+* rgt_i*;*(By Terminology I, create HQL)
5: *List<Node>* C_i*=get_Nodes(hql);*(Call M-C API 2)
6: ***for****(Integer k=0;k<* C_i*.size();k++){*
7: *show_Node(k);* (Call M-C API 5 to show node data)
}

b) Node-Adding Algorithm

For example, we add a child node beef to meat. Since a node occupies two values(the left and right), so by definitions, we can find rules as following:

$$LFT_i = LFT_{i_p} + 1 \quad (6)$$

$$RGT_i = LFT_{i_p} + 1 \quad (7)$$

In Formula (6) and formula (7), LFT_i is the left value of the added node, RGT_i is the right value and LFT_{i_p} is the left value of its parent. After the adding operation the result is shown as Fig.2. The data-adding algorithm is summarized as algorithm II.

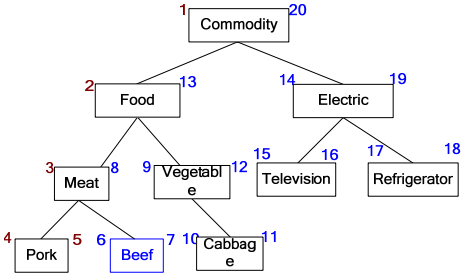

Figure 2. adding tree node result

Algorithm II: add_Node(Node node)

1: *Integer* pid_i*= Nodei.get*<1>*;*(Get PID value of $Node_i$)
2: *Node* $node_{ip}$*= get_Node(*pid_i*);*(Get Parent $Node_{ip}$)
3: *Integer* lft_p*=* $node_{ip}$*.get*<2>*;*(Get Left value of $Node_{ip}$)
4: *Integer* rgt_p*=*$node_{ip}$*.get*<3>(Get Right value of $Node_{ip}$)
5: *Integer* lid_p*=* $node_{ip}$*.get*<4>(Get Level value of $Node_{ip}$)
6: *Integer* lft_i*=* lft_p *+1;*(Formula (6))
7: *Integer* rgt_i*=*rgt_p *+2;* (Formula (7))
8: *Integer* lid_i*=* lid_p *+1;*
9: *save(*$node_i$*);* (Save node value to database)
10: *String hql="update WebTree wt set wt.lft =wt.lft+2 where wt.lft >="+ rgtp;* (Create HQL to update left value of the affected nodes blue arrows marked)
11: *batchUpdate Nodes(hql);*(Batch update nodes by hql)

c) Node-Removing Algorithm

When deleting a node we will also delete its all descendants. By definition 3, the number of $Node_i$ to be deleted is equal to $(RGT_i - LFT_i + 1)/2$.Any node has a pair of unique left and right values, so after deleting one node other appropriated nodes should be adjusted. The changing amplitude should be $RGT_i - LFT_i + 1$.

For example, after deleting the node Vegetable the changed result marked in blue is shown in Fig.3. The node-removing algorithm is summarized as algorithm III.

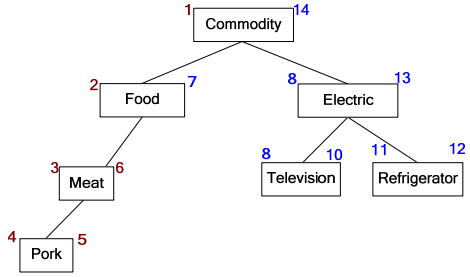

Figure 3. Removing tree node result

Algorithm III: remove_Node (Integer i)

1: *Node* $node_i$*= get_Node(i);*(Get $Node_i$ by M-C API 3)
2: *Integer* lft_i*=* $node_i$*.get*<2>*;*(Get Left value of $Node_{ip}$)
3: *Integer* rgt_p*=*$node_i$*.get*<3>(Get Right value of $Node_{ip}$)
4: *Integer span=* rgt_i *-* lft_i *+1*(Get adjustment margin)
5: *String hql="delete from WebTree wt where wt.lft>="+*lft_i*+" and wt.rgt<="+*rgt_i*;//(lft,rgt) ∈[*lft_i*, *rgt_i*]*
6:*batchUpdate_Nodes(hql);*(Batch update nodes by hql)
7:*hql="update WebTree wt set wt.lft = wt.lft –"+ span +"where wt.lft >"+* rgt_i*;*(Create HQL to update Left value of the affected nodes(Blue arrows mark)
8:*batchUpdate_Nodes(hql);*(Batch update nodes by hql)
9:*hql = "update WebTree wt set wt.rgt = wt.rgt –"+ span +"where wt.grt >" +* rgt_i*;*(Create HQL to update Left value of the affected nodes(Blue arrows mark)
10:*batchUpdate Nodes(hql);*(Batch update nodes by hql)

d) Node-Dragging Algorithm

The node-dragging operation follows the following rules as shown in formula (8):

$$N = RGT_i - LFT_i + 1 \quad (8)$$

N is the digit number occupied by the tree node. RGT_i is the right value of the node. LFT_i is the left value of the node. For example, when we move up Cabbage to Meat, the result is shown in Fig.4.

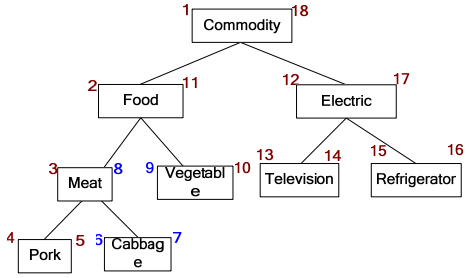

Figure 4. Move up tree node result

The coding values of sub-tree cabbage were changed based on the original right value of the new parent node

meat. As shown in Fig.1, the former right value of meat is 6 and the new left value of cabbage is 6.

Re-iterating sub-tree cabbage is equivalent to the each node updated by formula (9)-(10):

Algorithm IV: drag_Node(Integer f, Integer t)

1: *Node node$_f$=get_Node(f);*(Get moved from Node$_f$)
2: *Node node$_t$=get_Node(t);*(Get moved to Node$_f$)
3: *Integer lft$_f$= node$_f$.get<2>;*(Get Left value of Node$_f$)
4: *Integer rgt$_f$=node$_f$.get<3>*(Get Right value of Node$_f$)
5: *Integer lft$_t$= node$_t$.get<2>;*(Get Left value of Node$_t$)
6: *Integer rgt$_t$=node$_t$.get<3>*(Get Right value of Node$_t$)
7: *Integer lid$_f$= node$_f$.get<4>;*(Get Level value of Node$_f$)
8: *Integer lid$_t$=node$_t$.get<4>*(Get Right value of Node$_t$)
9:*Integer span=rgt$_f$-lft$_f$+1;*(Get adjustment margin)
10: *String hql="update WebTree b set b.pid ="+ t +" where b.id =" + f;*
11:*batchUpdate_Nodes(hql);*(Batch update nodes by hql)
12: *if(lft$_f$>lft$_t$){*(When Move up)
13: *Integer offset= lft$_f$-rgt$_t$;*
14:*hql="update WebTree b set b.lft = b.lft -"+offset+",b.rgt = b.rgt -"+offset+",b.lid=b.lid-"+(lid$_f$ - lid$_t$ -1)+",b.flag=1 where b.lft >=" + lft$_f$ +" and b.rgt<="+ rgt$_f$;* (Create HQL to update left and right value of the moved nodes)
15:*batchUpdate_Nodes(hql);*(Batch update nodes by hql)
16: *hql = "update WebTree b set b.lft = b.lft + " + span + " where b.lft >=" + rgt$_t$ +" and b.lft<"+ lft$_f$ +" and b.flag=0";*(Create HQL to update left value of the affected nodes blue arrows marked)
17:*batchUpdate_Nodes(hql);*(Batch update nodes by hql)
18:*hql="update WebTree b set b.rgt = b.rgt + " + span + " where b.rgt >=" + rgt$_t$ +" and b.rgt<"+ lft$_f$ +" and b.flag=0";*(Create HQL to update right value of the affected nodes blue arrows marked)
18: *batchUpdate_Nodes(hql);*(Batch update nodes by hql)
}
19: *if(lftf<lftt){*(When Move down)
20:*Integer offset= rgt$_t$-rgt$_f$-1;*
21:*hql="update WebTree b set b.lft = b.lft -"+offset+",b.rgt = b.rgt -"+offset+",b.lid=b.lid-"+(lid$_f$ - lid$_t$ -1)+",b.flag=1 where b.lft >=" + lft$_f$ +" and b.rgt<="+ rgt$_f$;* (Create HQL to update left and right value of the moved nodes)
22:*batchUpdate_Nodes(hql);*(Batch update nodes by hql)
23: *hql = "update WebTree b set b.lft = b.lft - " + span + " where b.lft >" + rgt$_f$ +" and b.lft<"+ rgt$_t$ +" and b.flag=0";*(Create HQL to update left value of the affected nodes blue arrows marked)
24: *batchUpdate_Nodes(hql);*(Batch update nodes by hql)
25: *hql = "update WebTree b set b.rgt = b.rgt - " + span + " where b.rgt >" + rgt$_f$ +" and b.rgt<"+ rgt$_t$ +" and b.flag=0";*(Create HQL to update right value of the affected nodes blue arrows marked)
26: *batchUpdate_Nodes(hql);*(Batch update nodes by hql)
27: *hql = "update WebTree b set b.flag =0 where b.flag =1";*
28: *batchUpdate_Nodes(hql);*(Batch update nodes by hql)

$$LFT_{new} = LFT_{old} - (LFT_{root(old)} - LFT_{root(new)}) \quad (9)$$

$$RGT_{new} = RGT_{old} - (LFT_{root(old)} - LFT_{root(new)}) \quad (10)$$

LFT_{new} and LFT_{old} are the new and old left values of the node to be moved. $LFT_{root(old)}$ is the left value of the old root node to be moved. $LFT_{root(new)}$ is the left value of new target root node. For example, the node cabbage left value equals 8-(8 - 6) and right value equals 9 - (8 - 6).

In addition to cabbage, the other nodes changes in a certain range between $RGT_{root(new)}$ and LFT_{old} . Because cabbage is to be moved forward, the involved nodes need to be moved back and need to be added the digit number (cabbage occupied). Therefore, we can summarize a law:

$$RGT_{root(new)} = ([LFT, RGT] + (RGT_{move} - LFT_{move} + 1)) < LFT_{move} \quad (11)$$

In cabbage moving backward case, it is similar to the forward, the laws is as following:

$$RGT_{move} <= ([LFT, RGT] - (RGT_{move} - LFT_{move} + 1)) < LFT_{root(new)} \quad (12)$$

The node-dragging algorithm is summarized as algorithm IV.

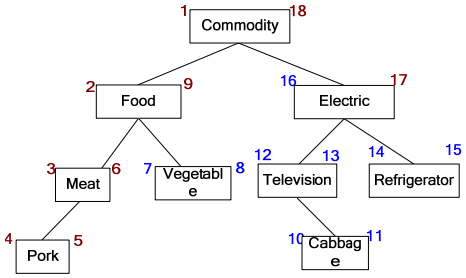

Figure 5. Move down tree node result

III. Architecture Description and Design

A. Formal Description of WSTC

In the field of computer science and software engineering, the formal method is a special technique that is suitable for description, development and validation of the software systems. Different mathematical foundations of the formal methods are different. In this section, we used the appropriate mathematical logic analysis to design the abstract model for improving the design reliability and robustness.

WTSC is a common SOA-based user interface component. We can access the function services through the service interface supplied by WTSC. WTSC had the ability of enabling platform-independent deployment for heterogeneous data management. We can donate WTSC as a seven-tuples *WTSC=<ID, TE, TIR, TU, TP, TS, TNS>*. Among them, ID represents a unique identifier of WTSC and others elements descriptions are as follows:

a) TE(Tree node Elements,TE)

TE= {E_0, E_1, $E_{2,...}E_n$}, (n>0), E_i is a node or element of the tree. E_i's data structure is satisfied with formula (1) and formula (2).

b) TIR(Tree node Information Resource)

TIR can be represented as five-tuples, *TIR=<RI, RN, RD, RW, RS>*.

RI (Resource Identification) represents a global information resource node identifier. RN (Resource Name) indicates the name of the resource.

RD (Resource Description) expresses the description information of the node. RD can be represented by five tuple $<R_d,R_m,R_l,R_w,R_s>$, $R_d= \{d_1, d_2, d_3,...,d_i\}$ statements the node descriptor collections. $R_m= \{Et, Ea, El\}$ represents the information of the metadata. Tree node data type is a mapping:

$Rd \rightarrow \{simpleType, complexType\}$.Ea is the identifier of attribute information. El describes the hierarchy of identifiers. R_l said for the position information of the resource nodes.R_W represents the resource weights, $w = \{0,1,2,...,100\}$, which is based on the importance of requirements determined by the priority level of the processing of information. For example, we assume that 0 is the lowest level and 100 is the highest priority level, the other and so on. RS denotes the level of resource sharing. $R_S \rightarrow \{Rs_1, Rs_2,...,Rs_n\}$.

According to information resources confidentiality requirements, for any $Rs_i = <LID, LN> (i =1,2,...,n)$, LID represent the sharing level, LN indicates the sharing level name.

c) TU(Web-Tree Service Component User,TU)

The person who accesses the tree node information can be regarded as a web-tree service component user and can be represented as $TU=<UID, UN, UT, UL>$. UID is a global identifier. UN is the user name. UT is the type set $UserType=\{user,group,\cdots\}$, user refers to a single entity, group refs a complex one. It also includes other types. UT $\in$UserType. UL is for the user location. According the movement state, UL can be divided into two types: the static and dynamic.

d) TP(Web-Tree Service Component Private,TP)

According to the different user of WSTC, TP can be divided into browsing, adding, modifying, deleting. TP can be represented as four-tuples $TP=<ID,N,T,O>$ ID is the global identifier of the privilege. N is the name of the permission, including a simple type and complex. T can be represented as $T= \{loading, adding, updating, removing,dragging,moving...\}$.O is an operating type. It is a mapping: $O = TU \times TIR \rightarrow T$.

e) TS(Web-Tree Service Component State,TS)

Component state can be defined as two-tuples $TS = <TID,HS>$.TID indicates the component global identity. HS is the hardware utilization status such as CPU usage, memory usage, etc. It is a mapping of the hardware set $\{HS_1,HS_2, ..., HS_x\} \rightarrow [0,1]$.

f) TNS(Web-Tree Service Component Net State,TNS)

Network environment state is represented as a four-tuples $TNS = <PID,NT,ND,NB>$.NT stands for the network type. ND is network latency.NB is the network bandwidth.

B. Related Design and Develop Technologies of WTSC

a) Java EE and SSH

Java EE, Java Platform Enterprise Edition, is Sun's standard platform for enterprise applications launched [10]. The platform is divided into three main editions Java EE, Java SE and Java ME. SSH (Struts, Spring, Hibernate) is the most popular open source lightweight development framework of Java EE platform. It includes Struts, Spring and Hibernate which are integrated in accordance with their respective characteristic. Spring is regarded as a core role of which is responsible for integrating Struts and Hibernate. SSH architecture is shown as Fig.6. Struts are highly configurable framework based on the MVC [15] design model. Spring provides a lightweight business logic solution of a "quick assembly-business components" and supplies declarative transaction managements, including RMI or web services. Hibernate is an excellent lightweight open source middleware of data persistence [16].

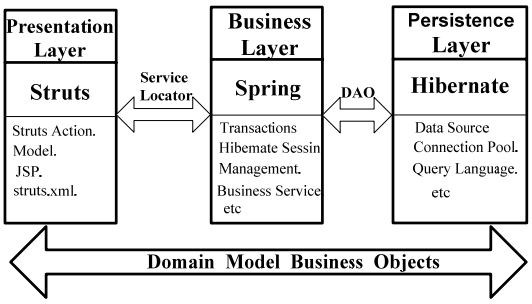

Figure 6. SSH Chart

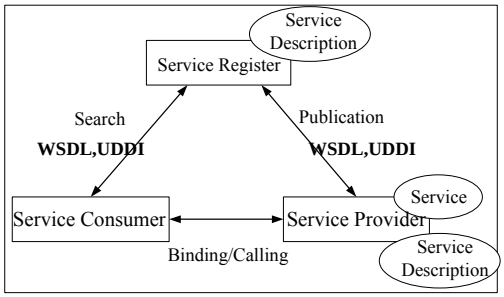

Figure 7. Cooperation diagram of SOA

b) SOA , Web Service and SCA

SOA (Service-Oriented Architecture) aims at maximizing the reuse of application services to enhance IT flexibility and efficiency [8]. SOA as a component model can associate with different functional units (we called them services) through well-defined interfaces and contracts. Interfaces are used to achieve services independent of the hardware platform, operating system and programming language. The independence makes various services can be unified and interacted with each other [10][12].Any service-oriented architecture contains three roles: service requester, provider and registry. Collaborative relationship between them is shown as Fig.7.

Web Service is one of SOA implementations (including Web Service, Session Bean, EJB, JINI, etc.), XML is used to describe and exchange data, SOAP as a protocol service is responsible for providers and consumers communicating each other. WSDL is used to define service interface descriptions. UDDI is used to web service registration and finding [8].

SCA is a component-based and service-oriented programming model proposed by the Open Service Oriented Architecture collaboration and formally donated to the OASIS organization in 2007. SCA provides a service component model for assembling a variety of

packaging and integration of heterogeneous applications. Its goal is to enable the service component binding a variety of transport protocols freedom [17].

C. SOA-SSH Software Development Framework of WTSC

Based on related technical standards introduced above, we build a general service component architecture named SOA-SSH and the overall hierarchical structure is shown in Fig.8.

View Layer: The UI (User Interface) implementation technologies such as JSP, Freemark, and Java Applet are used to generate web blower pages.

Service Agent Layer: Service request processing is used to interact with the user requests of database access such as loading, saving, removing and etc. Service calls are required to be obtained by WSDL description from the ESB, and then the developers should rewrite service agent codes.

ESB Layer: ESB is the middle layer between service requesters and providers. At the client, when a request message is sent to the XML Data-Bus, the proxy service firstly caught it and after a series of processing, the request message is sent to the business service layer. At last it will be further put back forward the request to the external service providers [22].

Service Layer: Service layer is a service provider, generated by a stateless session bean. Usually it is coarse-grained and encapsulated by enterprise business logic.

Business Component Layer: This layer is responsible for implementation of the Java EE business logic components and usually can be completed by the Spring or EJB.

Persistent Component Layer: The layer is a middleware between enterprise applications and database. Business logic data existed as objects in memory, while in relation database as physical table. Communication between them can be carried out by ORM (Object Relation Mapping) component.

Database Layer: A relational database (Oracle, MySQL, SQLServer, etc.), is responsible for enterprise data storage.

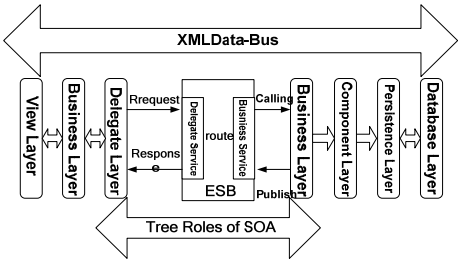

Figure 8. SOA-SSH Software Architecture of WTSC

D. SCA Description of WTSC

Following the standard SCA and SOA, we can design the internal function interfaces of web-tree service component and describe them by XML files. As we all know that the web-tree service component API includes loading, adding, removing, dragging and other operations. These operations will be packaged as a functional component for users to directly calling by URI (Uniform resource identifier) [24]. We can reduce duplication of development and enhance software scalability by SCA. The internal structure of the WTSC is shown in Fig.10.

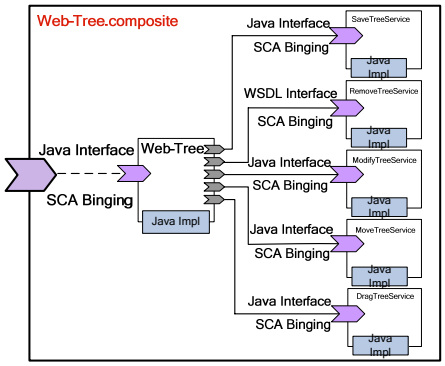

Figure 9. WTSC SCA model

IV. Implementation and Validation

A. Prototype System Implementation

In order to verify the flexibility, efficiency, compatibility and scalability of WSTC, we choose a catalog system as an application prototype. Base on SOA-SSH development framework and SCA stander, we designed and drew class and component diagrams of the prototype system as Fig.10 and Fig.11.

B. Prototype System Peformance Test

To validate performance, we design and realization a prototype system. User interface page is shown as Fig.12.We chose three different tree components dtree, xtree and WTSC implemented by three different algorithms (I-C, R-C and M-C). We named them as schema-1, schema-2 and schema-3 respectively. We change data-scale from 500 to 500000.The test performances of different operations are shown as Fig.13, Fig.14, Fig.15 and Fig.16.

V. Conclusions and Future Work

A. Conclusions

From the performance test results of the prototype system, we can draw at least three conclusions:

1) In terms of functionality

Three schemas all can support web-tree data online loading and display. Only the latter two can support asynchronous data adding, deleting and dragging operation.

2) In terms of performance

In small-scale data (5000), three scenarios in data loading and displaying, the difference is not obvious. When the data size grew rapidly, the third had very obvious advantages. When it came to data adding, dragging and dropping, there had no significant difference between the latter two. Because of the third using combination associated encoding method of parent-son relationship and left-right coding, in the realization of large-scale web-tree data (10^6 or more) deleting, its performance is significantly superior.

3) In terms of others

The third used the SOA-SSH and SCA architecture to design of a universal service component for large-scale web-tree data management. The component can provide direct service interfaces for the user accessing by URI, reducing the duplication of development, reducing the coupling platform, enhancing the versatility and scalability. Meanwhile, it can provide a good reference to user-oriented design, development and management of the web client service component.

B. Feture Work

Authors suggest that, at least three aspects will be further studied in the future [22] [23] [24]:

1) About friendliness

A common service components can be generally accepted by users and promoted for using, to a large extent depends on the user interface of the software is friendly or not, and the operation is convenient or not. Future we will use Ajax asynchronous transfer and rich client technologies to enhance friendliness.

2) About performance

We should continue to optimize the algorithm, such as nodes dragging operation. When loading large-scale data, in addition to using asynchronous loading techniques, we can optimize the design of the database such as building indexes of fields. Meanwhile, in the realization of software development, we can use new concepts such as SaaS to optimize software architecture and improve data exchange and sharing ways.

3) About security

In the future, we should focus on solving the technical difficulties of how to building a reasonable access control model and using client-side encryption technologies to increase security of WTSC.

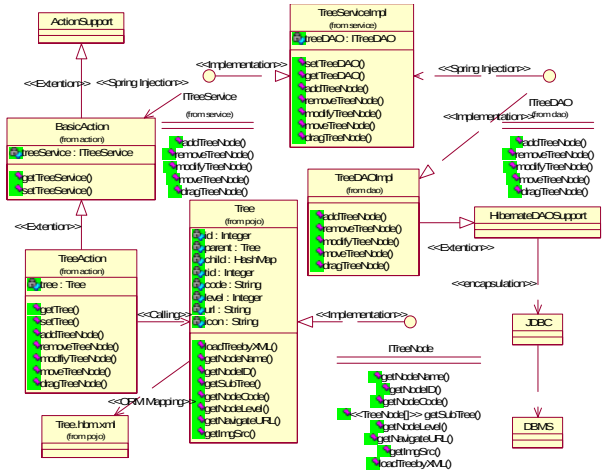

Figure 10. Class Diagram of WSTC

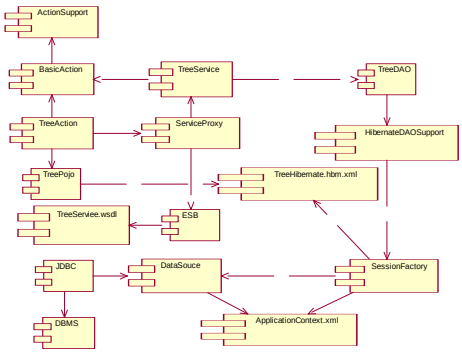

Figure 11. Component Diagram of WSTC

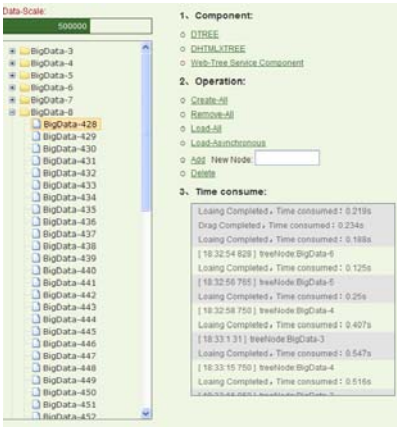

Figure 12. WSTC Performance Compared Test Prototype

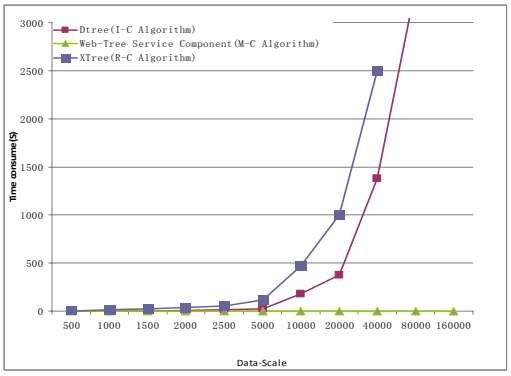

Figure 13. Data-Loading operation performance

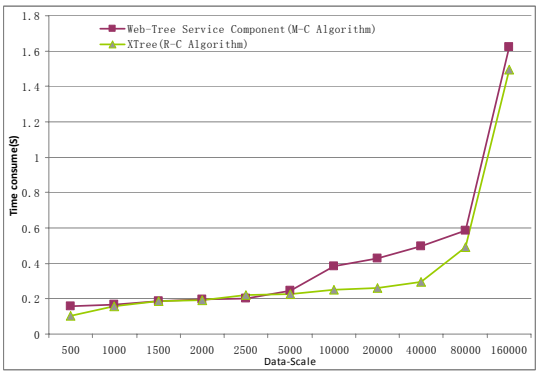

Figure 14. Data-Adding operation performance

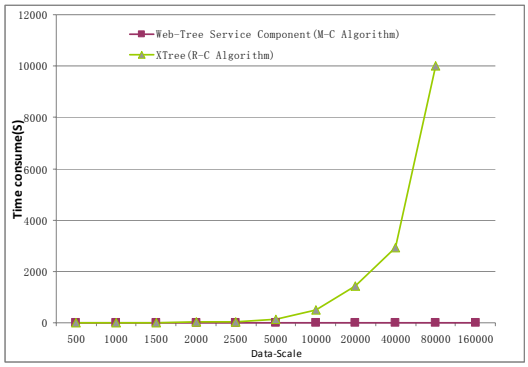

Figure 15. Data-Removing operation performance

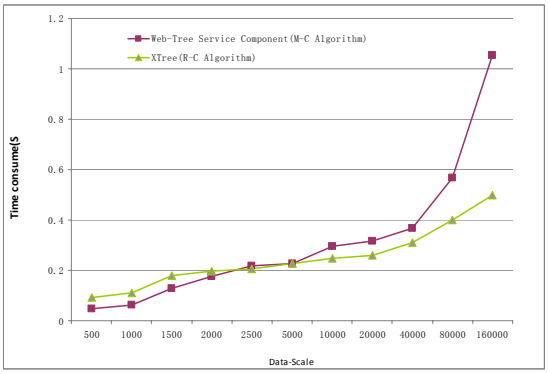

Figure 16. Data-Dragging operation performance

REFERENCES

[1] Compuware AJAX Edition [EB/OL] http://www.compuware.com/en_us/application-performance-management/products/ajax-free-edition/overview.html.

[2] Introducing JSON.[EB/OL]http://www.json.org/.

[3] Wang Suo,You Feng and Zhao Hengyong,Implementation of mass datas fast loading of Web-based tree structure[J],Computer Engineering and Applications,2008,44(27):165-167.

[4] Jian Wang, Hongying Fang and Chuangchuan Chen,An advanced Storage of Tree Based on RDMS[J],Journal of Chongqing Nomal University(Natural Science Edition),Vol.24 No.4,Oct. 2007,pp.2-4.

[5] Dejun Chen, Yingzhe Ma and Zhude Zhou,Celerity Algorithm of Building Dynamic Catalogue Tree[J],Journal of Wuhan University of Technology(Transportation Science&Engineering),Vol.32,No.1,Feb 2088,pp.41.

[6] Renkun Ying,Data Structure based on OO method and C++ Describtion[M], Beijing,TsingHua University Press,2007,pp.186-199.

[7] Huang Liuqing and Wang Manhong, New Software Engineering: A Component-Oriented Approach [M], TsingHua University Press,2006, pp.18-20.

[8] Yang Fuqing and Mei Hong,Design and Implementation of Component-Based Software[M], Beijing,TsingHua University Press,2008,pp.10.

[9] Business motivation model (BMM), version 1.0. Object Management Group, OMG Document formal/08-08-02 (2008)

[10] Business process modeling notation (BPMN) 1.2. Object Management Group, document formal/2009-01-03 (2009)

[11] Hongjun Sun, Shuangxi Huang and Yushun Fan, SOA-based Collaborative Modeling Method for Cross-Organizational Business Process Integration[J],http://www.paper.edu.cn.

[12] Gengli Fu and Baoxiang CAO,Design and Application of SOA-SSH Layered Architecture[J],COMPUTER TECHNOLOGY AND DEVELOPMENT,V01 . 20 No. 1,Jan. 2010,pp.75-77.

[13] Thomas ErI Service-Oriented Architecture Concepts, Technology, and Design, Printed in the United States of America, Prentice Hail Professional Technical Refrence,2005,pp.39-41.

[14] Aihu Liang, Skilled in SOA: Struts+EJB+Web Service Integrated application development Service Bus-based [M], Eletronics Industry Press, 2007, pp.326-327.

[15] Paul C.Brown,Implementing SOA:Total Architecture in practice[M]，China Machine Press,2009,pp.35-39.

[16] SCA is a global hygiene and forest products company. [EB/OL] http://www.sca.com/

[17] Donald Brown, Chad Michael Davis and Scott Stanlic, Struts2 in Action [M], Amecican, Manning Publications Co. 2008, pp.309-313.

[18] Christian Bauer, Gavin King, Java Persistence with Hibernate[M], Amecican,Manning Publications Co. 2005, pp.54

[19] Gary Mark, Spring Recipes A Problem-Solution Approach [M], Appress, 2006, pp.21, 93,135.

[20] Newcomer Eric, Lomow Greg. Uderstanding SOA with Web services[M].Trenton:Addison Wesley,2004,pp.56-58.

[21] Fielding R T. Architecture styles and the desing of network-based software architecture [D]. Irvine:Univ. of California,2000.

[22] Tao Tan, Hongjun Chen, A Personalization Recommendation Method Based on Deep Web Data Query, Journal of Computers, Vol 7, No 7, JULY,2012,pp1601-1602.

[23] Xanjun Li, Gang Ye, Zhongwen Li, Shilong Ma, Study and Implementation of Spacecraft Integration Test Platform Based on Component Technology, Vol 6, No 5, MAY, 2011,pp965-967.

[24] Minghui Wu, Xianghui Xiong, Jing Ying, Canghong Jin, Chunyan Yu，QoS-driven Global Optimization Approach for Large-scale Web Services Composition, Journal of Computers, Vol 6, No 7, 2011,pp1452-1453.

Zhidong Wang received his B.S. degree in computer science from Nanjing Information Engineering University, China in June 2002 and his M.S. degree in military operations research from Nanjing PLA University, China in March 2005. He is currently working towards his Ph.D. degree in computer application technology at Beihang University, China. His current research interest includes service-oriented software architecture and service component design.

Lichao Ye received his B.S. degree in chemical defense engineering command from Beijing Engineering Institute of Chemical Defense, China in June 1998 and his M.S. degree in command automation from Beijing Engineering Institute of Chemical Defense, China in June 2001. His current research interest lies in audit information.

Wenfa Li received his B.S. degree in computer applications from Xi'an University of Posts and Telecommunications, China in June 1994 and his Ph.D. degree in network and information security from Chinese Academy of Sciences, China in June 2010. His current research interest lies in network security and intrusion detection.