Simon Khoury, Peter Kropf, and Gilbert Babin. "Resource Warehouses: a Distributed Information Management Infrastructure." in High Performance Computing Symposium 2002 (HPC 2002). April 2002. Submitted.

Resource Warehouses: a Distributed Information Management Infrastructure

Simon Khoury[*], Peter Kropf[*], and Gilbert Babin[†]

[*] University of Montreal
Department of Computer Science and Operations Research
Montreal, Quebec, Canada
{elkhous,kropf}@iro.umontreal.ca

[†] HEC – Montreal
Department of Information Technologies
Montreal, Quebec, Canada
Gilbert.Babin@hec.ca

KEYWORDS

Distributed Systems, Communication Protocol, Middleware, Resource Management, Web Operating System (WOS), Resource localization.

ABSTRACT

This paper presents work related to the design of distributed systems, which is useful for emerging Internet applications. We propose algorithms for searching and managing distributed information about resources and services using locally available warehouses. The concept of warehouses has been introduced in the Web Operating System (WOS) (Kropf 1999). Warehouses have the ability to decide which information should be stored, replaced or removed without any intervention of the user. We present a tree structure for WOS warehouses, an attribute/value scheme used for describing resources, and the algorithms to look up information about resources. Among other things, warehouses take into account the capacity limitations of the devices that the WOS is using. Moreover, in order to share locally available information, WOS warehouses need to communicate with each other. We present an approach which allows for profitable exchange of information between the various warehouses. The advantage of our approach is the use of a simple method to describe what is being looked for (i.e., the intent), instead of specifying where to find it (i.e., the extent). We have implemented our warehouse structure in Java taking advantage of its portability.

INTRODUCTION

The Web Operating System (WOS) was developed to provide a user with the possibility to submit a service request without prior knowledge about the service (where it is available, at what cost, under which constraints, etc.) and to have the service request fulfilled within the user's desired parameters (time, cost, quality of service, etc.). In other words, the WOS is designed to enable transparent use of network-accessible resources, whenever a user requires a service, provided the service is available. These services may be specific hardware or software, or a combination of both. A user needs only to understand the WOS interface and does not need to know how the service request is fulfilled. Therefore, the WOS provides a computation model and the associated tools to enable seamless and ubiquitous sharing, and interactive use of software and hardware resources available on WOS enabled systems over the Internet. There exist numerous other projects with similar goals, for example JINI (Waldo 1998), SLP (Guttman *et al.* 1998), INS (Adjie-Winoto *et al.* 1999), and JXTA (Gong 2001). Each project has its own characteristics (service directories, resources discovery and localization, etc.). The WOS uses distributed databases, called warehouses, which allow open access and search procedures. The work presented in this paper proposes algorithms to efficiently search and manage the resources available in the system. In particular we present the warehouse structure, the resource request and look up algorithms, and warehouse management approach.

This paper is organized as follows: the next section presents the search approach in WOS; in Section "Warehouse," we describe the architecture and the functionality of the warehouses. Following that, Section "Implementation" briefly presents our implementation, while Section "Related Work" discusses other applications similar to WOS. Finally, we present future work planned for warehouses and draw some conclusions in the last section.

THE SEARCH APPROACH

One of the greater challenges for the WOS is the implementation of an efficient algorithm to look up available services or resources in the WOSNet, which is a set of nodes running a WOS software and knowing at least one other WOS enabled node (i.e., there is an entry in the node's warehouses referencing directly or indirectly that other node). The WOS supports a search algorithm for finding the WOS node providing the requested services with the best offer. The search algorithm currently used in WOS to look for

available services continues until every WOS node is contacted (Kropf 1999), or until the WOS node providing the requested services with the best offer is found. This concept does not mandate when the request should stop. We can envision a number of approaches for solving this problem. For example, all requested messages can use a TTL (time-to-live) counter, which states how many times the user request can be forwarded to other WOS nodes. This approach prevents forwarding the request over all the WOSNet but brings about other problems as pointed out in (Annexstein *et al.* 2001). Further work is needed in this area.

The bootstrap problem

The WOS system uses a generic protocol, WOSP (Babin *et al.* 1998) to communicate information between nodes. A particular instantiation of the protocol is called a *version*. When a new WOS node is added to the WOSNet, all it knows is an initial list of WOSP versions it understands. It knows nothing about other nodes in its neighbourhood. The first order of business for the node is therefore to locate other WOS nodes in its neighbourhood. We refer to this situation as the *bootstrap problem*. In the original description of WOSP, the bootstrap algorithm resolved this situation as follows. The new WOS node broadcasts a message requesting information about any WOS node to its local network. If no answer is received (i.e., no WOS nodes are located on the local network), the new WOS node, in the worst case, broadcasts to the next network level (Kropf 1999). This process continues until at least one WOS node is found or until every machine on the Internet is visited (Babin *et al.* 1998). This approach produces a high load on the network, and might also be impeded by firewalls.

We propose an alternative bootstrap algorithm to resolve these problems. In the new bootstrap algorithm, we use a normal WOS node, which offers a *bootstrap service*. A node offering this service is referred to as a *bootstrap node*. The bootstrap service populates the warehouses of the nodes requesting it by sending back information about other bootstrap nodes it knows. Bootstrap nodes allow for an isolated WOS node to enter the WOSNet. Each WOS node must therefore have initial information about at least one bootstrap node. It is conceivable that some web sites or their equivalent will be devoted to providing information of well-known bootstrap nodes.

Figure 1 illustrates how an isolated WOS node could enter the WOSNet. When a new WOS node is being installed, it broadcasts to the local network, asking for new WOS nodes; if there is at least one WOS node on the local network, it will respond with a positive answer. In this case, the isolated WOS node will know another WOS node that is already a member of the WOSNet, and then becomes a member of WOSNet as well. Otherwise, it sends a request

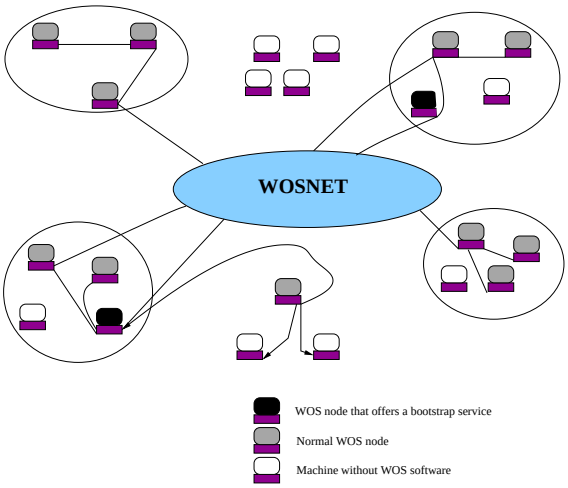

Figure 1: Bootstrap.

to a bootstrap node, requesting the bootstrap service. The bootstrap node's answer is used by the isolated WOS node to populate its warehouses. The approach we propose here will not flood the Internet, since we broadcast only to the local area network, instead of broadcasting everywhere at one time.

WAREHOUSES

Most resource management systems are built for use in small and midsize distributed computing environments (Abdennadher *et al.* 2001). They are often represented by a global catalogue containing all the resources located on the available nodes of the distributed system. This approach cannot be applied to large distributed operating systems since the highly dynamic traffic as well as the quickly changing structures of a large heterogeneous system like the Internet require steady and efficient update of such resource information. Instead of a global catalog, WOS uses limited knowledge, kept in local warehouses which contain all resources managed locally. Every WOS node also maintains warehouses about remotely available resources. The node does not keep all the information it receives. A trade-off should be found between two opposite criteria: acquisition of new information; and control/update of existing data entries. The first criterion guarantees that all information requested by the local user is available without accessing other warehouses, which would cause an additional communication overhead. The second criterion ensures that obsolete information is deleted so as to limit the size of the warehouse (Unger 2000). In fact, a WOS warehouse is more than just a static database with limited storage capacity. Indeed, each warehouse in the WOS must have the ability to decide without any additional user activity which information should be:

- stored in which place in the warehouse,
- replaced or deleted,
- obtained from another warehouse and which one.

Warehouse structure

The warehouses are the central data structure in the WOS. They achieve expressiveness and are based on a hierarchy of attributes and values. An attribute is a category in which an object can be classified, for example its color. A value is the object classification within that category, for example, red. Together, an attribute and its associated value form an attribute-value pair or av-pair. The hierarchy of av-pairs allows WOS nodes that provide a service to precisely describe what they provide and consumers to easily describe what they require. Characterizing resources and services using attributes and values has been suggested before in other contexts (Adjie-Winoto *et al.* 1999) and we draw upon previous work in this area in designing our warehouse structure. While several complex request languages exist in the literature, our approach, based on attributes and values, is particularly simple. It follows a lightweight scheme and is easy to implement even on impoverished devices such as PDA or other small devices. We also design the warehouse structure to be independent of the specific language used to perform requests, so that it can also be used in the context of other service description languages.

The structure of a warehouse is a hierarchical arrangement of av-pairs, such that an av-pair is a descendent of another av-pair in the hierarchy when it is dependent on it (see Figure 2). There can be multiple values per attribute and each value can in turn be refined by multiple attributes. Each value-node in the warehouse has pointers to all the services it describes. This allows a user to find compatible services if the requested service is not available.

User's request

Users make requests to identify services that fulfill their needs. Requests are built using arrangements of attributes and values, related by relational operators $(=, <, >, \leq, \geq, \neq)$, which we call *av-relations*. For example, av-relation [Attribute: Price $\leq$ Value: 10 dollars] would request services that cost less than 10 dollars. A request consists of two predicates, P_u and P_c, which are combinations of av-relations using logic operators. Predicate P_u describes the user-specific characteristics of the service requested, while P_c corresponds to characteristics of the context in which the request is made. Predicates P_u and P_c are described using the following grammar:

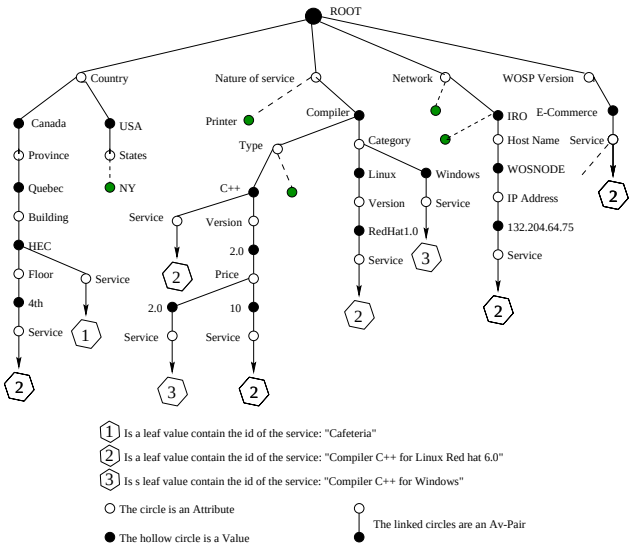

Figure 2: Warehouse Structure.

```
<predicat>    ::= <expression>
                  *[ ['and' | 'or'] <expression>  ]* ;
<expression>  ::= '(' <predicat> ')' |
                  'not' <predicat> |
                  <av-relation>;
<av-relation> ::= attribute <op> value;
<op>          ::= '=' |
                  '<=' |
                  '>=' |
                  '<' |
                  '>' |
                  '<>' ;
```

Predicates P_u and P_c are used as parameters to the request processing algorithm described below.

Processing a request

The request processing algorithm selects the answers that best fit the user's requested service, as expressed by P_u and P_c. This algorithm is at the heart of the warehouse search algorithm. It returns all characteristics corresponding to all the services that match P_u and possibly P_c. The algorithm first starts by searching services corresponding to P_u and P_c. If the number of services found is larger or equal to the number of services requested, noted q, the process stops; this is the best case situation. Otherwise, the algorithm removes one av-relation from P_c until it finds a sufficient number of answers; this is the intermediate case. If the request was not fulfilled, i.e., the number of services found is smaller than q, it continues removing av-relations from P_c until either a suitable number of services is found or until all av-relations are removed from P_c; this latter situation is the worst case situation.

Formally, we define the request algorithm as follows:

Algorithm 1 *Request* (P_u, P_c, q).

Let P_u : the selection criteria defined by the user u
Let P_c : the selection criteria representing the current context c
Let q : the number of services that should meet the selection criteria
Let S_r : the set of services meeting the selection criteria
Let S : the set of all services
Let $s \in S$: a service
Let P_c', P_c'' : intermediate selection criteria
Let n : number of av-relations to remove from P_c
Let *AV-Rel* (P) : the number of av-relations in predicate P
Let *Serv* (P) : the set of services matching predicate P; *Serv* (P) is defined as $\{s \in S \mid P\}$

A-Best case

$S_r \leftarrow$ *Serv* $(P_u \& P_c)$
If *Card* $(S_r) \geq q$
  return S_r
End-If

B-Intermediate case

For each $n \in \{1, ..., \textit{AV-Rel}\ (P_c) - 1\}$
  $S_r \leftarrow$ *Serv* $(P_u \& P_c')$ where
    AV-Rel $(P_c') + n =$ *AV-Rel* (P_c) and
    $\nexists P_c'' \neq P_c' \mid$ *AV-Rel* $(P_c') =$ *AV-Rel* (P_c'') and *Card* (*Serv* $(P_u \& P_c')) <$ *Card* (*Serv* $(P_u \& P_c''))$
  If *Card* $(S_r) \geq q$
    return S_r
  End-If
End-For

C-Worst case

$S_r \leftarrow$ *Serv* (P_u)
return S_r

Warehouse cooperation approach

The request processing algorithm, named hereafter *Request*, specifies how local warehouses are searched, but does not indicate how the search is extended to other WOS nodes. For that purpose, we must define two other search algorithms, namely *Locate* and *Bootstrap*. The *Locate* algorithm performs a request over a set of WOS nodes, noted I, for resources specified by predicates P_u and P_c. It basically calls the request algorithm on every WOS node $i \in I$.

Algorithm 2 *Locate* (P_u, P_c, q, I).

Let P_u : the selection criteria defined by the user u
Let P_c : the selection criteria representing the current context c
Let q : the number of services that should meet the selection criteria
Let I : the set of nodes to which the request is sent
Let *Request* (P, P', q) : calls to the request algorithm

For each $i \in I$, in parallel
  Submit *Request* (P_u, P_c, q) to node i
End-For

The *Bootstrap* algorithm refers to the revised bootstrap approach presented earlier.

Algorithm 3 *Bootstrap ()*.

Let *Request* (P, P', q) : calls to the request algorithm
Let *Locate* (P, P', q, I) : calls to the locate algorithm
Let *Timeout* $()$: waits for an arbitrary period of time
Let *ValueOfAttr* (S, a) : returns the set of values of attribute a describing services $s \in S$
Let *LB* : the local broadcast address
Let *BN* : the address of a known broadcast node
Let I : a set of Internet addresses
Let P: a predicate
Let q' : minimal number of hosts to query

Locate (true, true, 1, {*LB*})
Timeout ()
$P \leftarrow$ [Attribute : IP $\neq$ value : LocalHost]
$I \leftarrow$ *ValueOfAttr* (*Request* (P, P_c, q'), IP)
If *Card* $(I) > 0$
  return
End-If
Locate (true, true, 1, {*BN*})
Timeout ()

Search algorithm

The global search algorithm, named *Search*, is a composition of the *Request*, *Locate*, and *Bootstrap* algorithms.

Algorithm 4 *Search* (P_u, P_c, q).

Let *BootstrapDone* : indicates whether *Bootstrap* has already been called or not
Let *ValueOfAttr* (S, a) : returns the set of values of attribute a describing services $s \in S$
Let *Request* (P, P', q) : calls the request algorithm
Let *Locate* (P, P', q, I) : calls the locate algorithm
Let *Bootstrap* $()$: calls the bootstrap algorithm
Let *Timeout* $()$: waits for an arbitrary period of time
Let I, I' : sets of Internet addresses
Let P: a predicate
Let q' : minimal number of hosts to query

BootstrapDone ← false
P ← [Attribute : IP ≠ value : LocalHost]

A-Local request

S_r ← *Request* (P_u, P_c, q)
If *Card* $(S_r) \geq q$
    return S_r
End-If

B-Remote request to known WOS nodes

I ← *ValueOfAttr* (*Request* (P, P_c, q'), IP)
If *Card* $(I) > 0$
    Locate (P_u, P_c, q, I)
    Timeout ()
    S_r ← *Request* (P_u, P_c, q)
    If *Card* $(S_r) \geq q$
        return S_r
    End-If
End-If

C-Find new WOS nodes

I' ← *ValueOfAttr* (*Request* $(P$, true, $q')$, IP)
If *Card* $(I') > 0$
    Locate $(P_u$, true, $q, I')$
    Timeout ()
    S_r ← *Request* (P_u, P_c, q)
    return S_r
Else

    D-Find at least one WOS node

    If *BootstrapDone* = true
        $return S_r$
    End-If
    BootstrapDone ← true
    Bootstrap ()
    Go To "A- Local request"
End-If

Best fit

The algorithm shown above chooses the best set of services fulfilling the user's request. The warehouse structure is designed in a flexible way, which allows the request processing algorithm to find the best fit (i.e., the services which match the user's request and most or all of contextual parameters). For instance, if the request is a printer (P_u = [Attribute: service = value: printing]) and the user is currently in the HEC building (P_c = [Attribute: building = value: HEC]), the search algorithm will first try to choose all printers located in the HEC building [value: HEC]. If the printing service was not offered in the HEC building, the search algorithm would provide the user with an alternate list of printing services in other buildings.

Managing resources

Whenever a WOS node receives answers from remote nodes, or at each warehouse modification request, it needs to update its warehouses' content. This update process determines whether information about resources should be inserted, updated, or even removed. The proposed warehouse structure facilitates these manipulations. New information is grafted to the warehouse tree structure, where appropriate. In order to properly manage knowledge about resources (i.e., to be able to remove old information), we keep track, for each av-pair, of its creation date, its last modification date, and its number of access. This information enables a WOS node to properly manage the limited storage capacity allocated to it.

IMPLEMENTATION

We have started the development of a Warehouse Manager. The manager implements the request algorithm, warehouse resource management (warehouse update process), and the warehouse structure. Our implementation is in Java to take advantage of its cross-platform portability. The warehouse structure is developed in XML; access to the warehouse structure is performed in Java using the XML package developed by IBM (Ceponkus and Hoodbhoy 1999; xml 2000). Figure 3 shows a partial warehouse structure in XML.

```
-<?xml version = "1.0"?>
  <!DOCTYPE wos:warehouse (View source for full doctype...)>
--<Root>
    - <Attribute AcN="1" CrD="23/05/2001" LMD="23/05/2001">
        WOSPVersion
        -<Value AcN="1" CrD="23/05/2001" LMD="23/05/2001">
            HP-WOSP
            -<Service> Cafeteria </Service>
              - - - - - - - - - - - - - - - - -
              - - - - - - - - - - - - - - - - -
          </Value>
      </Attribute>
  <Root>
```

Figure 3: An example representing warehouses in XML

RELATED WORK

Sharing services, the main objective of the WOS effort, is also the goal of many other projects currently under way at several research centers as well as in industry, among them: JINI, INS, JXTA and SLP. JINI (Waldo 1998) handles service lookup and discovery. The Service Location

Protocol (SLP) defined in IETF RFC 2608 (Guttman *et al.* 1998) proposes a service lookup algorithm based on multicast. INS (Adjie-Winoto *et al.* 1999) uses a decentralized network of resolvers to discover names and route messages. Unlike JINI, SLP and INS, the approach proposed in the WOS project is completely decentralized. This is also the case with JXTA (Gong 2001) which can be completely decentralized, completely centralized, or a hybrid of the two. However, JXTA does not mandate how messages are propagated and does not address how to name and bind services and resources; it can benefit from the algorithms we propose in the context of the WOS, especially the *Request* and *Bootstrap* algorithms, and the service description approach.

CONCLUSION AND FUTURE WORK

This paper describes the concept of searching and managing resources in warehouses for the WOS. We believe that our proposition is a suitable approach to enhance the functionality of the Web Operating System (WOS) and the capabilities to find the best answer to any request for a resource. We expect to successfully demonstrate the power of our approach.

In the future, there are many aspects of resource lookup and management issues that we would like to investigate. In the following we list some of them:

- Define a process to limit the size of warehouses by removing unused data according to some priority rule.
- Find a better way to resolve the problem of an isolated WOS node.
- Use a dictionary to make compatible similar requests or to generalize requests.
- Find a relation between the information of resources and services. This relation can be used to provide the user request with the greatest answers

ACKNOWLEDGEMENTS

This work was supported by the Canadian Natural Sciences and Engineering Research Council (NSERC).

References

Abdennadher, N.; P. Kuonen; and G. Coray. 2001. “New trends in high performance computing.” In *IEEE euromicro workshop on network computing* (Jan.).

Adjie-Winoto, W.; E. Shwartz; H. Blakrishnan; and J. Lilley. 1999. “The design and implementation of an intentional naming system.” In *17th ACM symposium on operating systems principles (SOSP’99)* (Dec.).

Annexstein, F.; K. Berman; and M. Jovanovic. 2001. “Latency effects on reachability in large-scale peer-to-peer networks.” In *Thirteenth ACM symposium on parallel algorithms and architectures (SPAA 2001)* (Heraklion, Crete, Greece, Jul.).

Babin, G.; P. Kropf; and H. Unger. 1998. “A two-level communication protocol for a web operating system (WOS).” In *IEEE euromicro workshop on network computing* (Västerås, Sweden, Aug.), 939–944.

Ceponkus, A. and F. Hoodbhoy. 1999. *Applied XML: A toolkit for programmers*. Wiley.

Gong, L. 2001. *Project JXTA: A technology overview*. (Apr.). Available at http://www.jxta.org/.

Guttman, E.; C. Perkins; J. Veizades; and M. Day. 1998. *RFC 2608 : Service location protocol white paper topic*. IETF RFC 2608, (June). Available from http://www.ietf.org/rfc/rfc2608.txt.

Kropf, P. 1999. “Overview of the WOS project.” In *1999 advanced simulation technologies conferences (ASTC 1999)* (San Diego, CA, USA, Apr.), 939–944.

Unger, H. 2000. “Distributed resource location management in the web operating system.” In *High perfomance computing 2000 (HPC 2000)* (Washington, DC, USA, Apr.).

Waldo, J. 1998. *Jini technology*. Sun Microsystems, Inc (June). Available at http://www.sun.com/products/jini.

2000. *Extensible markup language*. Available from http://www.w3.org.