Using Abstract State Machine in Architecture Design of Distributed Software Component Repository*

Yunjiao Xue, Leqiu Qian, Xin Peng, Yijian Wu, and Ruzhi Xu

Department of Computer Science and Engineering, Fudan University,
Shanghai Key Laboratory of Intelligent Information Processing, Fudan University,
PO Box 200433, 220 Handan Road, Shanghai, China
{yjxue, lqqian, pengxin, wuyijian, rzxu}@fudan.edu.cn

Abstract. Recently many enterprises have established their own software component repositories. Because of the physical isolation to each other and independent decision on the classifying and specification mechanisms, the repositories form a distributed and heterogeneous system, hindering the reusability of component resource. Due to the infeasibility of integrating all the repositories physically, it is necessary to use a collaborative way to integrate such repositories and form a logically uniform architecture with consistent user interface and retrieval mechanism, to improve the reuse of software components. In this paper we propose an architecture of integrated system based on intelligent agent, and make use of ASM in its architecture design, which can be validated formally. This architecture implements the required fundamental functionality.

1 Introduction

The increase of software components has led to a remarkable amount of enterprise repositories [1-3][5]. For sharing software components among enterprises, a high level, regional repository beyond enterprises is needed, such as a city- or country-level ones, which is named as a centralized, integrated component repository.

In general, enterprises tend to forbid the external access due to security. The direct access is inaccessible, which needs a middle layer. Ideally, distributed enterprises can easily submit their queries to the centralized repository through a uniform portal. However, there exist some problems, which are outlined in the following.

(1) When numerous retrieval requests are sent to the centralized one, they will form remarkable workload to the server (due to the bottleneck of the system performance), thus significantly decreases the efficiency.
(2) In reality, the enterprises are not directly access to avoid secret-keeping due to high business competition.
(3) To achieve requested information on the centralized server and their own systems could influence data integrity and redundant work.

Therefore, a system with a collaboration mechanism is urgently needed to integrate the distributed information available at various enterprise repositories, while

* Partially supported by NSFC Grant No. 60473062; National 863 High-Tech Research and Development Program of Chin, a under Grant No. 2002AA114010.

H.T. Shen et al. (Eds.): APWeb Workshops 2006, LNCS 3842, pp. 861 – 868, 2006.
© Springer-Verlag Berlin Heidelberg 2006

sustaining data isolation. Such a system should provide a uniform logical view and retrieval interface to the outer world. The retrieval requests sent to the system will be transferred to the inner repositories, and ultimately a single set of results upon on each request can be fetched to the end users.

Unfortunately, a distributed architecture is very complicated. During the constructing process of each repository, the enterprises may employ different specification mechanisms, leading to many heterogeneous systems. This tremendously increases the complexity of the overall system. The intelligent agent-based technique introduced in [13-16] can be an alternative solution. It is beneficial to the integration of component repositories.

This paper reviews the related work on component repository. It presents a general idea of the Abstract State Machines (ASM) method and its architecture design. A discussion of its validations is also presented, followed by a conclusion.

2 Related Work

The major topics in repository construction include types of reusable software components, classification methods, storage, and search and retrieval mechanisms [6]. People are still working hard to develop the theory of component repository. For example, some systems use REBOOT (Reuse Based on Object-Oriented Techniques) [7] and PCTE (Portable Common Tool Environment) in Europe [8], STARS program in the USA that proposes a reference model of component repository, and an instance named ALOAF (Asset Library Open Architecture Framework) [9]. These programs focus on the component models and specification paradigms. The JBird project at Beijing University [3] deployed the component repository concept to come up with a long time, and advised a facet model for the component specification. The government office of Shanghai in China also started a project of component repository research to explore its application for enterprises [4]. Other projects, such as ComponentRank [10], Codebroker [11], and the Ohsugi et al's system [12] dedicated to the exploration of component retrieval in a repositary system. Tangsripairoj et al [6] introduced an interesting attempt, SOM (Self-Organizing Map) to construct component repositories.

3 Abstract State Machines

3.1 Introduction

The terminology of Abstract State Machines (ASM) was defined in [17], captures in mathematically rigorous yet transparent form of some fundamental operational intuitions of computing. This allows the practitioner to work with ASMs without any further explanation, viewing them as "pseudocode over abstract data" which comes with a well defined semantics supporting the intuitive understanding. Based on Börger's work, ASM is a practical design method that can solve the fundamental problems in software development [19]. With ASMs, one can elaborate the informally presented requirements of a desired system, and turn them into a satisfactory ground model, i.e. a functionally complete with abstract description of sufficient but not more than necessary rigor. The model can (a) be read and understood by and justified to the customer as solving his problem, (b) defines relevant system features for user

expectation, and (c) only contains the logic of the problem requirements for the system behavior (i.e. does not rely upon any further design decision belonging to the system implementation).

3.2 Basic Definition

ASMs are systems with transition rules.

if *Condition* **then** *Updates*

which transforms abstract states. The *Condition* (so called guard) under which a rule is applied is an arbitrary first-order formula without free variables. *Updates* is a finite set of function updates (containing only variable free terms) of form

$$f(t_1, \ldots, t_n) := t$$

whose execution is to be understood as *changing* (or defining, if there was none) the value of the (location represented by the) function f at the given parameters. Hereby we employ Gurevich abstract state machine theory, which was formerly known as *evolving algebras* or *ealgebras* and introduced in [18], as our approach's fundation. In its definition, the states of an ASM are simply the structures of first-order logic, except that relations are treated as Boolean-valued functions.

We introduce rules for describing changes to states. At a given state S whose vocabulary includes that of a rule R, R gives rise to a set of updates; to execute R at S, fire all the updates in the corresponding update set. An *update instruction* R has the form

$$f(t_1, t_2, \ldots, t_n) := t_0,$$

where the f is an r-ary function name and each t_i is a term. A *block rule* R is a sequence $R_1, \ldots, R_n$ of transition rules. To execute R at S, execute all the R_i at S simultaneously. A *conditional rule* R has the form

if g then R_0 else R_1 endif,

where g (the *guard*) is a term and R_0, R_1 are rules. The meaning of R is the obvious one: if g evaluates to *true* in S, then the update set for R at S is the same as that for R_0 at S; otherwise, the update set for R at S is the same as that for R_1 at S. A *choice rule* R has the form

choose v satisfying $c(v)$
  $R_0(v)$
end choose

where v is a variable, $c(v)$ is a term involving variable v, and $R(v)$ is a rule with free variable v. This rule is nondeterministic. To execute R in state S, choose some element of a of S such that $c(a)$ evaluates to *true* in S, and execute rule R_0, interpreting v as a. If no such element exists, do nothing.

4 Architecture Design

4.1 Integration Requirements

In the Internet computing environment, the following requirements must be satisfied to integrate various component repositories. They are: (1) It is unnecessary to

explicitly integrate all available repositories into a physically centralized repository; thus avoiding the great cost of integration and performance bottleneck in a central server. (2) It must support at least one central portal, while most of independent enterprise repositories are responsible for their own components management. (3) The retrieval requests from users must be sent to the central portal, such as via a Web site; thus the end users do not need to know the existence of other repositories. (4) Each repository must access retrieval requests consistently with maintaining interoperability and compatibility. The retrieving results must be sent to the end users in a unified format.

4.2 Overall Design

To implement the collaboration among enterprises, a collaboration model with a formal framework is developed to represent the interactions of agents. The top-level view of the whole architecture is shown in Fig. 1.

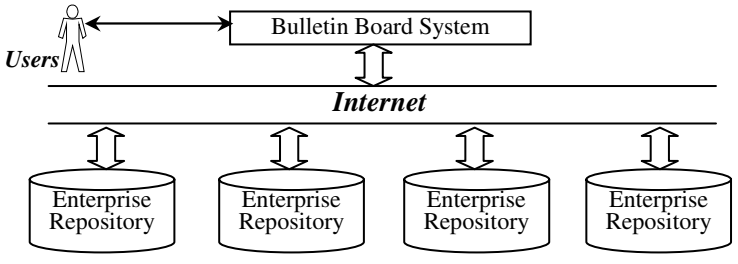

Fig. 1. The top-level view of the architecture

In the architecture, many enterprise repositories can be connected to Internet, and a centralized Bulletin Board System (BBS) acts as a mediator of the integrated behavior. The users will interact with BBS, for sending their retrieval requests and receiving the retrieving result from the same board. Each repository runs several agents that take charge of getting requests from the board, retrieving the object component in a local repository, and of returning the results to the board (if find some).

The BBS can be further partitioned into three regions. Some sharable component specifications that the enterprises do not want to keep secret can be submitted to the *Region of Sharable Component Specifications* of the Bulletin Board System. User retrieval requests are pooled in the *Region of Component Retrieval Requests*. After each request is performed, if there are any satisfied results, they will be sent to the *Region of Returned Retrieval Results*. Users observe the results of their requests from the *Region of Returned Retrieval Results* of the Bulletin Board System.

For each region, there is an intelligent agent running on an enterprise repository that takes charge of interacting with it.

4.3 ASM Design

As an abstracted machine, the state of the system is a 4-arity tuple $\Sigma = (S, R, A, U)$, where S, R, A and U respectively abstracts the state of the whole system, the retrieval

requests, the agents running on the enterprise repositories, and the results of the agent performing the requests. Here the sharable component specifications are not mentioned because they are not involved in the parallel retrieving process.

To simulate the architecture, we abstract the system into a set ***S*** of states,

S = {*WaitingforRequest, WaitingforAgent, RequestPerformed*}, with the meaning of each state as follows,

- *WaitingforRequest*: No any request in *the Region of Component Retrieval Requests* is un-performed.
- *WaitingforAgent*: A new request is sent to the system by some user, but there is no any agent from the enterprise repository that fetches it. So the system is waiting for some agent to perform the new one.
- *RequestPerformed*: A new request is performed by at least one agent and there is no any request keeping un-performed.

A denotes the set of all agents, $A = \{a_1, a_2, \ldots, a_n\}$, where each a_i is an agent and n is the amount of the repositories that join this system. n will increase if there are new repositories becoming the members of the system, making the system extensible.

R is a finite set of all the requests, notated as r. $R = \{r_1, r_2, \ldots r_{max}\}$, where max denotes the maximal amount of the requests that the region could contain (the expired requests will be archived). The state of each request is defined as a tuple rs with an indefinite arity. A mapping function $Ret: R \times A \rightarrow 2^A$ will compute the new state of each request and change its state into a new one, denoting which of all the agents have performed it. Correspondingly, the state of each agent is defined as a tuple with an indefinite arity. A mapping function $Fet: A \times R \rightarrow 2^R$ will compute the new state of each agent and change its state into a new one, denoting which of all the requests it has performed.

U is a finite set of all the results returned after the requests were performed by some agents, together with corresponding requests. $U = \{u_1, u_2, \ldots u_{max}\}$, where u_i is a result and max is the maximal amount of the results that the region could contain. In the same way, the expired results or the results that users have accessed could be archived. Each u_i has the form of (r, a, c), where r means a retrieval request, a means an agent that has performed r, and c is a set of the component specifications that match the request according to a. That is, a may find one or more component which specification(s) are compatible with the one(s) appearing in the request r.

The ASM of the system is a set T of transition rules, including the following ones,

Rule 0*: *SysStat* := *WaitingforRequest

This rule defines the system an initial state specification.

***Rule 1*:**

```
if newrequestcreated(r) & SysStat = WaitingforRequest
   then
    SysStat := WaitingforAgent
    R := AddReq(R, r)
end if
```

Note: *newrequestcreated* is an external event with a parameter r denoting a new retrieval request. *SysStat* is the system controlling state variable. The function *AddReq*(R, r) will append the new request r to R, thus making R changed.

***Rule 2*:**

```
choose r in R and a in A with rs(r) =∅ or a∉rs(r)
   do fetchedbyagent(a, r)
end choose
```

That is, if some request *r* is not performed by any agent, or *r* is not performed by some agent *a*, then *a* will have a chance to perform *r*.

***Rule 3*:**

```
if fetchedbyagent(a,r)& SysStat=WaitingforAgent
  then
    SysStat := RequestPerformed
    rs(r) := Ret(r, a)
end if
```

Note: *fetchedbyagent* is an external event with two parameter *a* and *r* denoting that some agent *a* fetches the request *r*. The function *Ret*(*r, a*) will add *a* into *r*'s state *rs*, specifying that *r* is performed by a new agent *a*.

***Rule 4*:**

```
choose a satisfying retrieve(a, r) ≠ ∅
   U := AddRes(r, a,  retrieve(a, r))
end choose
```

That is, for some request *r*, if an agent *a* performs it and finds some results, then returns this information to the server by appending it to the corresponding region.

***Rule 5*:**

```
for all r do
  if rs(r)≠∅ then SysStat := WaitingforReques end if
end for
```

When there is no any unperformed request, *Rule 5* will reset the state of the system to the initial one.

Validation: We will give an abbreviated validation to the system's functionality. When there is no any user sending retrieval requests, the system will stay in a static state, with *Rule 0* establishing an initial state. If there is any new request, with *Rule 1* the system will change its state and some agent could be activated to perform the new request. With *Rule 2*, an agent will merely perform those requests that it didn't touched before, so the unnecessary repetition and confliction is avoid. Also, a request will not be always neglected by any agent. Through *Rule 3*, for any request, at least one agent will perform it (as we do not require that each agent must perform each request). A distributed retrieval mechanism is implemented in this way. *Rule 4* insures the satisfying results could be sent to the server, and then users may find them via the server's interface. *Rule 5* lets the transitions to be closed and form a cycle. For further elaboration of the design, more rules could be complemented. Figure 2 demonstrates the transition of the system's state under the given rules.

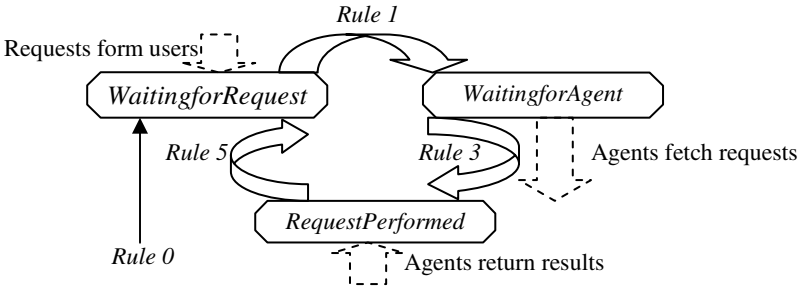

Fig. 2. State transition diagram of the system

5 Conclusion

External processes are forbidden to access an enterprise's internal information because of the security requirements. So, when we fail to find a desired component in current repository, we could not get direct help from other repositories. Otherwise, common users cannot access most of the enterprise repositories due to the unawareness of the address or the access protection. It is the reason that we propose the project of distributed heterogeneous software component repository.

Intelligent agent is an appropriate way to achieve the integration of distributed heterogeneous repositories. In this paper we propose an architecture of the system, and introduce a detailed design based on ASM, which is a good way to simulate the architecture and validate it.

The research issues introduced in this paper are the beginning of an on-going work. Even if the basic results and the main investigation direction have been given, much effort, both theoretical and practical, is needed to achieve a complete treatment of the problem. The future work includes the more detailed design of the structure of 4 or maybe more kinds of agents. The new retrieval methods in the distributed environment will be explored. Most of all, the exchanging of component specification between heterogeneous classifying mechanisms is a vital topic to achieve the integration.

References

1. Basili, V. R., Briand, L C., Melo, W.L.: How Reuse Influences Productivity in Object-Oriented Systems. Communications of the ACM 39 (10) (1996) 104-116
2. Bellinzona, R., Gugini, M.G., Pernici, B.: Reusing Specifications in OO Applications. IEEE Software, March Volume 12, Issue 2, (1995) 65-75
3. Yang, F.Q., Mei, H., and Li,K.Q.: Software Reuse and Software Component Technology. Acta Electronica Sinica, Vol. 27, No.2 (1999)
4. The Web site of Shanghai Component Repository. http://www.sstc.org.cn/. Accessed 5 Aug. (2005)

5. Henninger, Scott: An Evolutionary Approach to Constructing Effective Software Reuse Repositories. ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 2, April, (1997) 111-140
6. Tangsripairoj, S., Mansur, S., Samadzadeh, H..: Application of Self-Organizing Maps to Software Repositories in Reuse-Based Software Development. Proceedings of the 2004 International Conference on Software Engineering Research and Practice (SERP'04), Las Vegas, Nevada, June 2004, Vol. II, (2004) 741-747
7. Faget, J., Morel, J.M.: The REBOOT Environment. Proc. of 2nd International Workshop on Software Reusability (Reuse'93), Lucca, Italy, March (1993)
8. Long, F., Morris, E.: An Overview of PCTE: A Basis for a Portable Common Tool Environment. Technical Report CMU/SEI-93-TR-1, ESC-TR-93-175, Software Engineering Institute, Carnegie Mellon University, March (1993)
9. STARS Technical Committee. Asset Library Open Architecture Framework: Version 1.2, Informal Technology Report, STARS-TC-04041/001/02, August (1992)
10. Inoue, K., et al.: Component Rank: Relative Significance Rank for Software Component Search. In Proceedings of the 25th international conference on software engineering, Portland, Oregon, USA, May 6-8 (2003)
11. Yunwen, Y., Fischer, G.: Information Delivery in Support of Learning Reusable Software Components on Demand. In the Proceedings of the 7th international Conference on Intelligent User Interfaces, California, USA, 2002, ACM Press
12. Ohsugi, N., et al.: Recommendation System for Software Function Discovery. In Proceedings of the 9th Asia-Pacific Software Engineering Conference (2002)
13. Etzioni, O., Lesh, N., Segal, R.: Building Softbots for UNIX. In Etzioni, O., ed., Software Agents — Papers from the 1994 Spring Symposium (Technical Report SS–94–03), AAAI Press, (1994) 9–16.
14. Roesler, M., Hawkins D.T.: Intelligent agents. Online, vol. 18, no. 4 (1994) 18-32
15. Linda Rosen.: MIT Media Lab Presents the Interface Agents Symposium: Intelligent Agents in Your Computer? Information Today, Vol. 10, No. 3 (1993) 9-10
16. Maes, P.: Agents that reduce work and information overload. Communications of the ACM, Vol. 37, No. 7 (1994) 30-40
17. Börger, E., Stärk R.: Abstract State Machines. USA, Springer, 2003.
18. Gurevich, Y., Algebras, E.: An Attempt to Discover Semantics", Current Trends in Theoretical Computer Science, eds. G. Rozenberg and A. Salomaa, World Scienti_c, 266-292, 1993.
19. Börger, E.: High Level System Design and Analysis Using Abstract State Machines. In Hutter, D., Stephan, W., Traverso, P., Ullman, M., eds., Current Trends in Applied Formal Methods (FM-Trends 98). Springer LNCS 1641 (1998) 1-43