Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Parallel Iterative Reengineering Model of Legacy Systems

Xing Su, Xiaohu Yang, Juefeng Li
College of Computer Science Zhejiang University
Hangzhou, China
Email: {suxing, yangxh, lijuefeng}@zju.edu.cn

Di Wu
State Street Technology (Zhejiang)
Hangzhou, China
Email: dwu2@statestreet.com

***Abstract*—Great improvements have been made in methodologies on reengineering of legacy systems to enhance the migration process. Unfortunately, taking into account the large scale of these systems, the duration of reengineering process can still be quite long which will make an impact on the business the systems supported. In this paper we propose a parallel iterative reengineering model to reduce the reengineering process duration. The model uses the method of Formal Concept Analysis to process the complex access relationship between legacy components and shared data. An algorithm named "Bottom-Up", which consists of two parts, is introduced to build the parallel schedule for the reengineering process. This model shortens the reengineering duration and has been proven to be successful in improving the transition system's performance in a comparative experiment.**

***Index Terms*—Legacy System Reengineering, Parallel Iterative Reengineering Model, Formal Concept Analysis, Bottom-Up Algorithm, Performance**

I. Introduction

Legacy systems are old computer systems or application program that continues to be used, typically because it still functions for the users's needs, even though newer technology is available. After years of evolution, legacy systems have suffered from many problems, such as outdated technologies, decayed architectures, lack of documents and so on. These problems make the maintenance cost higher and higher. Methods are promoted for reengineering legacy systems such as "Chicken Little Strategy" [1] and "Butterfly Methodology" [2]. These methodologies all have deficiencies in data migration and access [3]. Alessandro in [4] proposed an iterative reengineering method to overcome these problems. This method suggested reengineering work to be iteratively executed on various components in different phases; during execution of the process there will be coexistence of legacy components, components currently undergoing reengineering, reengineered components, and new components, added to the system to satisfy new functional requests [5].

Although the iterative reengineering method improves the procedure of software reengineering in a large scale, the migration process of legacy systems is still suffering deficiencies. Firstly, the quick change of business environment for legacy systems nowadays requires a shorter reengineering cycle, while the heavy workloads makes it difficult to shorten project duration. Secondly, frequent development and maintenance of the code for transition systems will cause resources waste and increase the complexity, especially in some ultra-large legacy system. Thirdly, the cost on format change between old and new data module in the interactive operation will impair the performance of transition system. These unsatisfactory aspects will finally impact the customer's confidence.

Due to these problems, we proposed an enhanced iterative model: the parallel iterative reengineering model. This model is attractive because it allows several different components to be reengineered in one cycle. Formal Concept Analysis, which has strong mathematical capability of processing a group of elements with properties [6], is used in our approach to formalize the access relationships between legacy components and shared data. We proposed "Bottom-Up" algorithm to sort the concepts in the lattice of access relationships. Finally an Iterative Cycle Reference Table is obtained to help schedule concurrent development in reengineering process. The parallel reengineering model can decrease the complexity of data access and maintenance as well as make a full use of developing resources, thus enhance the transition system's performance.

The rest of the paper is organized as following. In section II, the formal method used to deal with access relationships of a legacy system is introduced. In section III, the parallel iterative reengineering model is presented in detail, including two parts of the "Bottom-Up" algorithm and the output in each stage. Section IV outlines a comparative experiment to prove the enhancement this method made in the performance of a transition system. The last section is the conclusion.

II. Former method

A. Formal Description of a Legacy System

The following gives the formal definitions of elements in the reengineering process of a legacy system. The legacy system is represented by S_0, while S' represents the target system. Then the parallel iterative model can be described by Equation (1), in which is the transition system after the nth iteration [7]. All systems of different states execute the same function as the original legacy system.

$$S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \cdots S_n \Rightarrow S' \qquad (1)$$

In system analysis work, the legacy system is divided into two parts: the Legacy Components (LC) and the shared Legacy Data (LD). To begin with, we will define the two factors that will appear in every cycle [8].

- *Legacy Components*: legacy system operating on a set of recorded data
- *Shared Legacy Data*: the data set operated by legacy system, used by various legacy components

LC and LD are expressed respectively in Equation (2) and (3):

$$LC = \{lc_1, lc_2, \cdots lc_i\} \tag{2}$$

$$LD = \{ld_1, ld_2, \cdots ld_j\} \tag{3}$$

B. Access Relationship Table

In a legacy system, dependencies between components and data are quite complex and can influence the performance of the transition system when components access shared data files through Data Banker [4]. The first step to analyze a legacy system involves identification and analysis of the usage relationship between data files and various components. In this paper we use the method introduced in [9] to establish the usage relationship. A table is created to show the access dependencies between legacy components and shared data. Those table cells filled with an "X" indicate an access relationship between the corresponding component and the shared data. The paper will give an example of a simplified legacy system, named System E, to make all conceptions easy to understand. Table I is the Access Relationship Table of System E.

TABLE I
ACCESS RELATIONSHIP TABLE OF SYSTEM E

		Shared Data					
		ld_1	ld_2	ld_3	ld_4	ld_5	ld_6
Legacy Components	lc_1	X	X				X
	lc_2	X	X				
	lc_3			X	X		X
	lc_4		X			X	
	lc_5			X			
	lc_6		X	X	X	X	X
	lc_7			X			

C. Building the Concept Lattice of Components and Data

Formal Concept Analysis (FCA) is a branch of lattice theory used to identify meaningful groupings of elements that have common properties [10]. The concept, defined as a pair of sets: a set of elements and a set of properties (X, Y), is used to represent a relationship between a component and a shared data set in this paper. The set of all the concepts of a given context forms a complete partial order, thus it can constitutes a concept lattice [11]. Concept Lattice reflects relations of super-concept and sub-concept. Every node in the lattice represents a concept and the node at a higher level is the super-concept of underside nodes which have edges to it. So the top node has all objects of discussion, and the bottom node has the all attributes. Assuming there's an access relationship between lc_i and ld_i, we can call (lc_i, ld_i) a concept. Based on the access relationships in Table I, we can build the concept lattice of system E using the Lattice Build Algorithm [10], as shown in Fig.1.

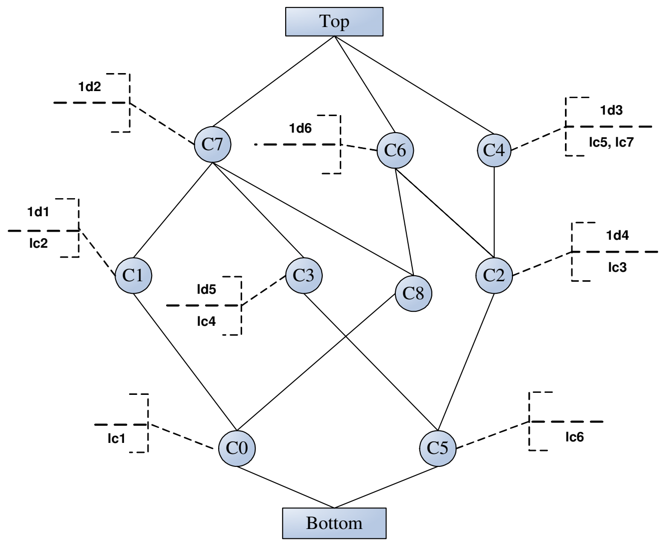


Fig. 1. Concept Lattice of System E

The theory of formal concept analysis and concept lattice guarantees the uniqueness of access relationships between legacy components and shared data in the access relationship lattice. Therefore we can avoid extra work on determining the optimal one among kinds of possible solutions.

III. PARALLELED ITERATIVE REENGIEERING MODEL

Parallel method allows various components and data to be operated in one cycle during the reengineering process. Its main idea is to analyze the concepts in a relationship concept lattice applying "Bottom-Up" algorithm. Based on the concept lattice obtained in last scenario, developers can already draw the schedule plan according to project's resources. However the workloads can be very heavy providing a complex lattice. Hence the relationships in concept lattice need further process. "Buttom-Up" algorithm is designed to meet this requirement. The algorithm consists of two parts: the Concept Partition Algorithm and the Recursive Sorting Algorithm. An Iterative Cycle Reference Table (ICRT) is delivered at the end to help schedule the iterative cycles.There are four stpes in total to fulfill the parallel iterative reengineering model. The whole process, including access relationship analysis and conception lattice building, is shown in Fig.2.

ICRT contains all optimized information of the access relationships in the legacy system. Each role within it covers the essence information of a node in the concept lattice, including elements of priority, name of the access concept, the legacy components and data that need reengineering. In the following paragraphs we will introduce the detailed theory of how "Bottom-Up" algorithm creates ICRT.

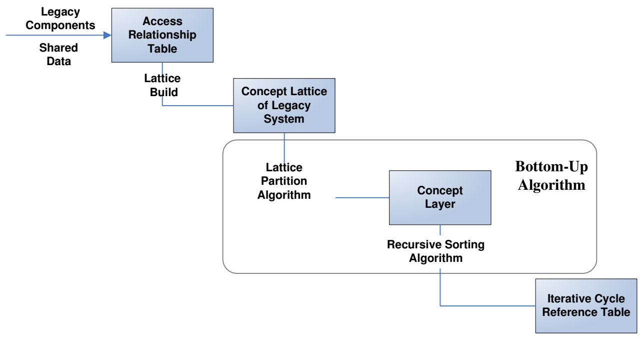

Legacy Components
Shared Data
Access Relationship Table
Lattice Build
Concept Lattice of Legacy System
Lattice Partition Algorithm
Concept Layer
Bottom-Up Algorithm
Recursive Sorting Algorithm
Iterative Cycle Reference Table

Fig. 2. Four Steps in the process of Parallel Iterative Reengineering Model

A. Concept Partition Algorithm

Based on the concept lattice of a legacy system, Concept Partition Algorithm works on partitioning the access concept into different layers according to their distance from the bottom. In this method, L_i is a set of concept, L_0 presents the bottom of the concept lattice, and $L_1, L_2, \cdots, L_i$ respectively present the layers above in order. Edge $\{C_j \rightarrow C_i\}$ stands for the relationship C_j that is the sub concept of C_i .There are three rules in detail for the partition algorithm.

- C_i is in L_1 if it directly connect to the bottom;
- C_i is at least in L_{k+1} if C_j is in L_k and edge $\{C_j \rightarrow C_i\}$ exists;
- If C_i has several sub concepts in different layers, the layer of C_i depends on the one with the highest layer.

The detailed Concept Partition Algorithm is described in Fig.3. Table II shows the partition result of the access concept lattice in System E

TABLE II
CONCEPT LAYER OF SYSTEM E

Layer	Concept Set
L_1	$\{C_0, C_5\}$
L_2	$\{C_1, C_1, C_3, C_8\}$
L_3	$\{C_4, C_6, C_7\}$

B. Recursive Sorting Algorithm

Once the table of concept layer is generated, we will use the Recursive Sorting Algorithm to form ICRT. The concepts are sorted uniquely according to four rules:

- A concept in lower layer has a priority in sorting;
- Once a concept is sorted, those who share the same super concept has a priority in sorting;
- A concept has a priority in sorting when all its sub concepts are sorted;
- A concept should be sorted after all its sub concept

Because L_i is a set, concepts in L_i are out-of-order. However, sorting concepts in the same layer can help to decide the priority in the reengineering cycle. The rule is that the one in the front of the queue will be reengineered first. Recursive sorting Algorithm is as shown in Fig.4. Table III is the ICRT of system E after sorting.

```
Concept Partition Algorithm
C is the set of concepts in lattice except for bottom and top
$c_i \in C, i = 1, 2, \ldots N$
1. for all $c_i \in C$ do
2.     If $\exists\{bottom \rightarrow c_i\}$
3.         $L_1 \leftarrow L_i \cup \{c_i\}$
4.         $C \leftarrow C - \{c_i\}$
5.     End if
6. End for
7. Repeat:
8. for all $c_j \in L_x$ do
9.     for all $c_i \in C$ do
10.        If $\exists\{c_j \rightarrow c_i\}$
11.            $L_{x+1} \leftarrow L_{x+1} \cup c_i$
12.            If $((c_i \in L_k)\&\&(k < x+1))$
13.                $L_k \leftarrow L_k - \{c_i\}$
14.            End if
15.        End if
16.    End for
17. End for
18. If $(L_x = \emptyset)$
19.    goto Exit
20. End if
21. x = x+1
22. Goto Repeat
23. Exit
```

Fig. 3. Bottom-Up: Concept partition algorithm.

TABLE III
ITERATIVE CYCLE REFERENCE TABLE OF SYSTEM E

Priority	Access Concept	Legacy Component	Shared Data
1	C_0	lc_1	$\emptyset$
2	C_1	lc_2	ld_1
3	C_5	lc_6	$\emptyset$
4	C_3	lc_4	ld_5
5	C_8	$\emptyset$	$\emptyset$
6	C_7	$\emptyset$	ld_2
7	C_2	lc_3	ld_4
8	C_6	$\emptyset$	ld_6
9	C_4	lc_5, lc_7	ld_3

C. Build the Iterative Schedule Table

During a project process, allocating the development resources will be influenced by project manager's experience as well as the recognition of the resources and legacy components. Also, four rules are described to conduct the resource allocation:

- The legacy components in ICRT should be transformed from top to bottom;
- When a legacy component was arranged to be reengineered within an iterative cycle, the related shared data should be migrated in the corresponding cycle;
- Legacy components in the same access concept should be arranged in the same iterative cycle for the parallel reengineering; If it is the case that the remainder development resources can not satisfy the reconstruction requirements

```
**Recursive Sorting Algorithm**
$L$ is the set of all layers
$L_i \in L, i = 1, 2, \dots N$; N stands for the total number of layers
$L_i.F$ stands for the first concept in $L_i$
$L_i.c$ is the concept in $L_i$
ICRT stands for Iterative Cycle Reference Table
1. For (i = 1toN)
2.     While($\exists L_i.F$)
3.         $L_1 \leftarrow L_1 - L_i.F$
4.         UpSort($L_1.F, 1$)
5.     End while
6. End for
7. Exit
function UpSort(c,i) // c is a concept, i is the layer number of c
1. if (i+1>N)
2.     return
3. End if
4. For (all $L_{i+1}.c$ that $c \rightarrow L_{i+1}.c$) do
5.     DownSort($L_{i+1}.c, i+1$)
6.     UpSort($L_{i+1}.c, i+1$);
7. End for
8. return
9. End function
function DownSort(c,i) // c is a concept, i is the layer number of c
1. if (i-1<1)
2.     return
3. End if
4. For (x = i:2)
5.     For (all $L_{x-1}.c$ that $L_{x-1}.c \rightarrow c$) do
6.         If (x==2)
7.             add $L_{x-1}.c$to ICRT
8.             $L_{x-1} \leftarrow L_{x-1} - L_{x-1}.c$
9.         Else
10.            DownSort($L_{x-1}.c, x-1$)
11.        End if
12.    End for
13. End for
14. Add c to ICRT
15. $L_x \leftarrow L_x - \{c\}$
16. End function
```

Fig. 4. Bottom-Up: Recursive sorting algorithm.

of all legacy components in an access concept, two options can be taken for the task scheduling:(1)put all reconstruction work of these legacy components in next iterative cycle (2)Continue the data migration work, and defer the reconstruction of the legacy components with lowest priority until next iterative cycle;

- At the end of a iterative cycle, if the next access concept in next cycle contains only shared data, two options can be taken for the data migration: (1) Migrate the data in current cycle if the concept is the super set of former concept; (2) Otherwise, delay the migration until next iteration cycle

Applying these rules on the ICRT of system E in table III, the iterative schedule can be generated, as shown in TableIV.

D. Schedule New Components' Development

For most of the time, the development of new components in a reengineering project is based on new business requirements. So their schedule which is defined at the beginning

TABLE IV
ITERATIVE SCHEDULE OF SYSTEM E

Iteration Cycle	Migration of Shared data	Legacy Components to Remold	Development Resource
1	ld_1	lc_1	X
		lc_2	Y
		lc_6	Z
2	$ld_2, ld_3, ld_4, ld_5, ld_6$	lc_4	X
		lc_3	Y
		lc_5	Z
3		lc_7	X

is impossible to adjust with the parallel model. Only when developers has the full right to schedule new components, they could be added into the parallel schedule. The whole process is simple. We can just make a few improvements in ICRT according to the access relationships between new components and shared data. Actually we just take the new components as legacy ones in ICRT. Assuming there're two new components nc_1 and nc_2 to be developed in System E. nc_1 has relationships with ld_1 and ld_5 and nc_2 has the relationships with ld_1, ld_2, ld_6. The ICRT with new components after adjustment is as shown in table V.

TABLE V
ITERATIVE CYCLE REFERENCE TABLE OF SYSTEM E

Priority	Access Concept	Legacy Component	Shared Data
1	C_0	lc_1	$\emptyset$
2	C_1	lc_2	ld_1
3	C_5	lc_6	$\emptyset$
4	C_3	lc_4,nc_1	ld_5
5	C_8	$\emptyset$	$\emptyset$
6	C_7	$\emptyset$	ld_2
7	C_2	lc_3	ld_4
8	C_6	nc_2	ld_6
9	C_4	lc_5, lc_7	ld_3

The time complexity of the "Bottom-Up" algorithm based on the concept lattice is $O(n^3)$. Actually in the practical application, the total number of components in general legacy system is under 100. Thus the schedule result can be calculated in some reasonable time. Therefore we can draw the conclusion that the parallel iterative reengineering model is feasible. A premise to be clear is that the parallel iterative reengineering model mainly focuses on the relationships between components and data. The final goal of components extraction with any methodology is to get a system division in which the components are of strong cohesion inside and week coupling between each other. So the dependencies between components are already weak enough. Even if there is data transmission between components, it is much less uncomplicated than that between components and data.

IV. Comparative experiment

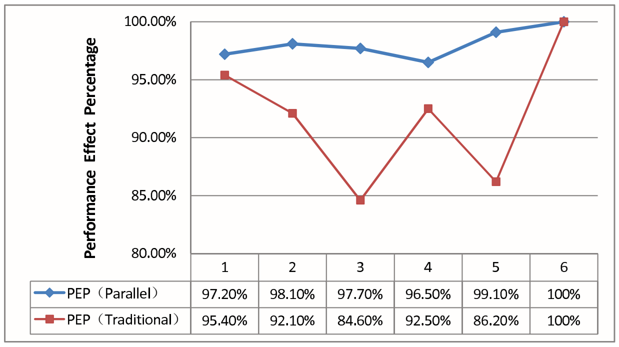
The parallel solution talked above has been applied to a fund accounting system named FE. The system which has a history of nearly 20 years involves more than 1,000,000 lines of source code and is developed by various languages. Parallel iterative reengineering model have been applied to schedule the migrating cycles and control the performance of transition system in the past three years development. To compare the effectiveness of parallel model and traditional model, a formula is introduced to qualify the Performance Effect.

Suppose the performance from legacy system to target system is proportional to the amount of legacy components to be remolded, the effect of transition system by data migration is qualified as Performance Effect Percentage (PEP):

$$PEP_i = \frac{1}{N} \cdot \sum_{j=1}^{N} \frac{PR_{(i,j)}}{(PT_j - PL_j) \cdot \frac{NC_i}{TotalC} + PL_j} \qquad (4)$$

where N represents the total number of all cases in the stress testing, PL_j and PT_j respectively represents the performance of the jth case of legacy system and target system, $PR_(i, j)$ represents the performance value of the jth test case at the end of the ith iterative cycle, NC_i represents the total number of transition components that have finish reconstruction in the ith iterative cycle, $TotalC$ represents the total number of all legacy components in the system.

To generate the simulated the data in traditional iterative reengineering project without parallel model, we made some control in the reengineering process. At the end of each iterative cycle, we made some adjustment to some components to make sure that they still access old database through the data agent. Then three access relationships are randomly picked up after adjustment and the average value of experiment results is obtained to calculate PEP. Fig.5 is the comparative result in graph. The blue line is the PEP of the migrating process in parallel iterative model. The red line shows the PEP of the process in original reengineering model which is simulated by control. The x coordinate stands for performance percentage value and y coordinate stands for the iterative cycle. The total number of iterative cycles in the reengineering process is six.



	1	2	3	4	5	6
PEP（Parallel）	97.20%	98.10%	97.70%	96.50%	99.10%	100%
PEP（Traditional）	95.40%	92.10%	84.60%	92.50%	86.20%	100%

Fig. 5. Results of comparative experiment on PEP in six cycles

In the project above, we can get that the performance of transition system in FE has been improved significantly by 10% around. By analyzing these data, we can draw the conclusion that with the same order of reengineering, the parallel model of iterative reengineering has improved the performance of the transition system.

V. Conclusion

Rapid reengineering process and full use of development resources are main challenges for reengineering of legacy systems. Parallel iterative reengineering model proposed in this paper is designed to satisfy these requirements. It allows several components and shared data to be migrated in one cycle according to their access relationships. Also, the Iterative Cycle Reference Table built by “Bottom-up” algorithm helps developers to allocate developing resources according to the reengineering schedule, which can make a full use of resources in some extent. As shown in the comparative experiment, the performance of transition system has been increased compared to that of iterative reengineering model.

In future, we will take into account the access strength in the access relationships between legacy components and shared data. To gain a more effective solution, threshold of access strength would be one of the main focuses in building the access relationship table. Furthermore, we will focus on the system structure and seek for solutions to enhance the performance of transition system in the reengineering process.

References

[1] M. Brodie and M. Stonebraker, *Migrating Legacy Systems: Gateways,interfaces, and the Incremental Approach*, San Francisco: Morgan Kaufman, 1995.

[2] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, and D. O’Sullivan, *The Butterfly Methodology: A Gateway-Free Approach for Migrating Legacy Information System*, Proc. Int’l Conf. Eng. Complex Computer Systems, pp. 200-205, 1997.

[3] A. Bianchi, D. Caivano, and G. Visaggio, *Method and Process for Iterative Reengineering Data in a Legacy System*,Proc. Seventh IEEE Working Conf. Reverse Eng., pp. 86-96, 2000.

[4] A. Bianchi, Danilo Caivano, Vittorio Marengo and Giuseppe Visaggio, *Iterative Reengineering of Legacy Systems*, IEEE Trans on Software Eng, 2003, 29(3):225-241.

[5] A. Bianchi, D. Caivano, V. Marengo, and G. Visaggio, *Iterative Reengineering of Legacy Functions*, Proc. IEEE Int’l Conf. Software Maintenance, pp. 632-641, 2001.

[6] Juefeng Li, Xiaohu Yang, Tao Huan, Zhijun He, *Abstract high-maintainable classes from legacy systems based on formal concepts analysis*, IEEE International Conference on Systems, Man and Cybernetics, 2004(2): 1249 - 1254.

[7] Gabriela Arevalo, *High Level Views in Object Oriented Systems using Formal Concept Analysis*, PhD thesis, University of Berne, January 2005.

[8] Juefeng Li, Xiaohu Yang, Zhijun He, *Using formal concept analysis for scheduling legacy system iterative reengineering process*, IEEE International Conference on Machine Learning and Cybernetics, 2006, 2351 - 2357.

[9] Igor Ivkovic, Kostas Kontogiannis, *Using Formal Concept Analysis to Establish Model Dependencies*, Proceedings of the International Conference on Information Technology: Coding and Computing, 2005, 2:365-372.

[10] Bernhard Ganter and Rudolf Wille, *Formal Concept Analysis: Mathematical Foundations*,Springer Verlag, 1999 .

[11] Garret Birkhoff. *Lattice Theory*, American Mathematical Society, 1940.