Enterprise Application System Reengineering: A Business Component Approach

Shi-Ming Huang, National Chung Cheng University, Taiwan
Shin-Yuan Hung, National Chung Cheng University, Taiwan
David Yen, Miami University, USA
Shing-Han Li, Tatung University, Taiwan
Chun-Ju Wu, National Chung Cheng University, Taiwan

ABSTRACT

Due to the rapid changes in information technology (IT), the life cycle to develop an information system has become significantly shorter. Furthermore, companies worldwide have invested much time and effort in developing their existing enterprise application systems in order to maintain their own competitive advantages. Specifically, they are facing a similar problem of preserving the investment. Information system reengineering technology (ISRT) is one way to solve this problem. However, most of the systems reengineered by using traditional ISRT may lack flexibility and reusability and also have some quality concerns. The component-based software engineering can provide some remedy by providing a flexible and reusable enterprise application system. Many companies could benefit from thinking more in terms of building their own business components for their information system areas. This article considers a reengineering approach in order to migrate legacy systems to component-based information systems with a quality control mechanism. Companies can build their own business component libraries in the reengineering process. One additional benefit is to build information systems from business components rather than reinventing the wheel in the future. This study proposed a system architecture and also implemented the prototype system to validate that our mechanism is technically practical. Additionally, we used an example to evaluate that our approach is effective and practicable in the real world.

Keywords: business component; component-based software engineering; enterprise application systems; information system reengineering technology (ISRT)

INTRODUCTION

The variable information technology (IT) environments bring new challenges for legacy systems in the modern world. This need unlikely will be fulfilled by redeveloping legacy systems, because legacy systems are crucial for companies, and therefore, they cannot afford to write them off. However, legacy systems cannot continuously adapt themselves easily to changes in technology. A case in point
is that HP announced plans to stop selling the HP3000 product line within two years and to cease support at the end of 2006. Another one is that Microsoft posted that all support for Windows NT Server 4.0 was scheduled to end on December 31, 2004.

When information technology changes, companies always clash over whether to redevelop or to sustain their legacy systems. The reasons behind discarding or sustaining legacy systems are shown in Table 1 (Umar, 1997).

Legacy systems represent the investment that companies have made previously. Redeveloping legacy systems not only wastes current software investments but also causes organizations to lose competitiveness as a result. Thus, how do companies preserve the value of legacy systems and prevent the current software investments from disappearing?

Nowadays, leading software component standards have evolved from nascent definitions of simple architectures to mature specifications of complete runtime environments. Some standards, such as Microsoft’s .NET and Sun Microsystems’s EJB, stress the advantages of flexibility and reusability. This, indeed, is the solution to solving the problems of legacy systems. Companies will benefit greatly by using encapsulating information systems with software components. IT professionals can decompose the legacy system into small parts and wrap them with technology of software components. Then, a company can introduce adjustments by adding or removing software components instead of redeveloping new information systems. They also can alter partially the legacy systems in order to adapt to rapid changes in IT environments.

The object-oriented (OO) technique, because of its strength of encapsulation and inheritance, is suitable to build a component-based information system (Brazier, Jonker, & Treur, 2002; Hughes, 1999; Morisio, Tully, & Ezran, 2002). It offers a powerful model and de-

<table>
<thead>
<tr>
<th>Discard Reasons</th>
<th>Sustain Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy systems cannot satisfy the flexibility and growth requirements of modern enterprises</td>
<td>Legacy systems have support staff that is trained to operate these systems</td>
</tr>
<tr>
<td>Many off-the-shelf client/server packages with nice GUI are becoming available</td>
<td>Legacy systems are very reliable and perform very well</td>
</tr>
<tr>
<td>New employees do not want to work on legacy systems created before they are employed</td>
<td>Some emotional attachment to legacy applications among senior staff</td>
</tr>
<tr>
<td>Legacy systems cannot exist in the new platform such as Web platform</td>
<td>The administrative support of legacy systems (e.g., backup recovery, disaster handling, change management, security) has matured over the years</td>
</tr>
<tr>
<td>Legacy systems are becoming increasingly expensive to maintain and operate</td>
<td>Legacy systems provide vital services that are very risky to discard</td>
</tr>
</tbody>
</table>

Table 1. Legacy application reengineering dilemma
Predicting Software Abnormal State by using Classification Algorithm
Yongquan Yan and Ping Guo (2016). Journal of Database Management (pp. 49-65).
www.igi-global.com/article/predicting-software-abnormal-state-by-using-classification-algorithm/165162?camid=4v1a

Fuzzy Imputation Method for Database Systems
www.igi-global.com/chapter/fuzzy-imputation-method-database-systems/20379?camid=4v1a