Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Reengineering for service oriented architectures: A strategic decision model for integration versus migration

Amjad Umar[a,*], Adalberto Zordan[b]

[a] Department of Information and Communication Systems, Fordham University, Schools of Business Administration, 113 West 60th Street, New York, NY 10023, United States
[b] Via S. Colomba, 2, 31011 Asolo (TV), Italy

ARTICLE INFO

Article history:
Received 13 January 2008
Received in revised form 16 May 2008
Accepted 23 July 2008
Available online 14 August 2008

Keywords:
Enterprise architectures
Enterprise service bus (ESB)
SOA patterns
Application migration
Application integration
Cost-benefit analysis

ABSTRACT

Service Oriented Architecture (SOA) is a popular paradigm at present because it provides a standards-based conceptual framework for flexible and adaptable enterprise wide systems. This implies that most present systems need to be reengineered to become SOA compliant. However, SOA reengineering projects raise serious strategic as well as technical questions that require management oversight. This paper, based on practical experience with SOA projects, presents a decision model for SOA reengineering projects that combines strategic and technical factors with cost benefit analysis for integration versus migration decisions. The paper identifies the key issues that need to be addressed in enterprise application reengineering projects for SOA, examines the strategic alternatives, explains how the alternatives can be evaluated based on architectural and cost-benefit considerations and illustrates the main ideas through a detailed case study.

1. Introduction

The fiercely competitive and ever fluctuating market has forced several companies to streamline operations and reduce product delivery time through flexible and adaptable systems. For needed flexibility and adaptability, service oriented architecture (SOA) provides a loosely coupled architecture which allows business components to discover and communicate with each other over a standards-based infrastructure (see texts Barry (2003), Bieberstein (2005), Stojanovic and Dahanayake (2004) and websites (SOA Portal; IEEE Computer Society Technical Committee on Services Computing; IBM site on SOA; Sun site on SOA) for additional information on SOA). SOA is currently receiving a great deal of industrial as well as academic attention. Unfortunately, SOA is also introducing a great deal of confusion due to new jargon and a diverse array of standards and commercial products. The main challenge in introducing SOA is that many enterprise applications – some old and some new and based on numerous vendor products – collectively need to be reengineered to conform to the SOA features. For example, a typical online purchasing system may involve several applications such as an order processing system, a payment and billing system, a purchasing system, an inventory management system, a shipping system, a customer relationship management (CRM) system, and possibly a supply chain management (SCM) system. These applications belong to different vintages (some are very old while others are relatively new), use different technologies (some use legacy 3270 screens, others use the latest voice over IP interfaces), and have different business value (some are mission critical, others are peripheral). To reengineer these closely interrelated applications for SOA, a wide range of questions arise:

- Should some of these applications be integrated by wrapping them with SOA interfaces *without* changing the applications internally? If so, which ones and how?
- Is there a need to migrate some of these applications by decomposing then into reusable components? If so, which ones and how?
- Which applications require integration plus migrations? Is it better to replace these applications with pre-packaged SOA compliant application packages? If so, which vendor products should be used?
- What are the strategic, architectural and technical tradeoffs in making the aforementioned decisions? What are the cost-benefit considerations in making these decisions?

In our consulting practice, we have found that these important reengineering questions are not being addressed adequately even though a great deal of information is becoming available to

* Corresponding author. Tel.: +1 212 636 6134; fax: +1 732 424 2023.
E-mail addresses: umar@amjadumar.com (A. Umar), adalberto@adal.it (A. Zordan).

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2008.07.047

develop SOA-based systems from scratch. This paper attempts to answer these questions by presenting an overview of the application reengineering decisions in SOA, identifying the key issues involved in each decision, and proposing a decision model for application reengineering for SOA (see Sections 2 and 3). The stages of the proposed decision model (strategic analysis, architectural issues and cost-benefits of the major alternatives) with focus on integration versus migration tradeoffs are analyzed in detail in Sections 4–6, respectively. Section 7 illustrates the main ideas through a case study, Section 8 compares this paper with related approaches and highlights the main contributions, and Section 9 contains the concluding comments.

2. Reengineering for SOA: integration versus migration issues

The main idea of SOA is that the applications should be decomposed into *reusable* components that deliver the business services. Thus as more services are needed, new components with the needed services can be added easily. For example, a bank provides a set of services (e.g., deposits, withdrawals, fund transfers) and these services are provided through components that can be combined into banking applications.

There are several definitions of SOA. After reviewing several definitions, we have decided to use The Open Group 1.1 version of SOA definition: service-oriented architecture (SOA) is an architectural style that supports service orientation (The Open Group website). We translated this definition to emphasize that a service-oriented architecture is based on the following three fundamental features (Bieberstein, 2005):

- *Reusable components:* It is important to decompose business applications into business components (BCs) in such a fashion so that as many components as possible are general purpose (i.e., reusable) and as few as possible are special purpose. It is highly desirable to create common services and components that can be reused to serve many different requests.
- *Web-services enablement:* The components must have well defined service interfaces that can be stored in a directory so that service clients (SCs) can query an interface directory to discover and invoke the needed service providers (SPs). Although older technologies (e.g., CORBA, DCOM) can be used for service definition and discovery, web services (WS) is the favored enabling technology at present (Barry, 2003). WS provides a widely accepted mechanism for service definition through web services definition language (WSDL) that can be defined and discovered through a universal, description, discovery and integration (UDDI) directory by exchanging XML messages over the Internet by using HTTP. Specifically, WS-based application that is currently being used inside an organization can be easily outsourced or moved to an external site without having to re-engineer the communication platform. In addition, WS with a UDDI service can allow http access to applications and business components that could be geographically distributed, thus leading to a global information system architecture.
- *Enterprise service bus (ESB):* Instead of point to point communications between participants, a loosely coupled common middleware infrastructure must be used for communications, brokerage, security, directory and administration services needed throughout the enterprise. Although such an infrastructure can be provided by the existing enterprise application integration (EAI) platforms, the SOA patterns strongly suggest WS-enabled enterprise service busses (ESBs) for SOA (Bieberstein, 2005). We will discuss EAI and ESBs later.

Even though a longer list of SOA features can be found in the literature (Erl, 2005), the aforementioned three features embody the main requirements of an SOA from a practical point of view. These somewhat naïve requirements lead to a very powerful architecture which can support and promote highly flexible and reusable business services for the current and future enterprises. However, these requirements are nontrivial and expensive to comply with – organizations have to decompose applications into reusable components, define WSDL interfaces and buy, install and manage the ESBs.

2.1. Quick overview of application re-engineering strategies

Reengineering of applications has been an active area of work for quite a while (see Brodie and Stonebraker, 1995; Lithicum, 2000; Ruh et al., 2002; Umar, 1996, 2004a,b). In reality, most of the work that deals with legacy applications revolves around reengineering (Dedene and DeVreese, 1995; Morgenthal, 2000). Different reengineering approaches have been proposed by different researchers over the years. Brodie and Stonebraker (1995), for example, primarily concentrate on migration of legacy applications that include gradual migration (“chicken little”) and complete migration (“cold turkey”) approaches. Lithicum (2000) and Ruh et al. (2002) focus on different integration strategies that include data integration, process integration, and user-view integration. Umar (1996, 2004a) presents a more complete discussion of the topic by proposing integration-in-place, complete-rewrite, gradual migration, data warehouses, and ‘do nothing’ as the main strategies. Instead of focusing on legacy applications, Land and Crnkovic (2006) divert their attention to merging of similar and duplicate applications that may exist due to mergers or close collaboration of organizations. They discuss options such as “Merge” (which is subdivided into “evolutionary merge” and “instant merge”), “Choose One”, “Start from Scratch”, and “No Integration”. Although all these approaches are viable, we want to concentrate on a few main strategies that are of vital importance due to SOA. Since SOA is an architectural style, we propose that reengineering for SOA can be examined in terms of integration-in-place and migration strategies (and their subdivisions), discussed below. This proposal is based on the literature reviews and first hand insights gained through consulting assignments.

2.2. Integration-in-place strategies

The main idea of integration-in-place strategy is that the applications themselves are not touched (i.e., not internally modified and decomposed into reusable components) but are instead externally interconnected by using WS and possibly an ESB. Specifically, this strategy involves two options:

1. Integration-partial (point-to-point): In this case, a WS interface is added to existing applications (i.e., the applications are ‘wrapped’ by WS interfaces) and applications communicate with each other by using SOAP or other WS protocols.
2. Integration-full: The communication between applications is handled by the ESB instead of point to point communication of option 1.

These two options can be used to gradually transition existing applications to SOA, i.e., start with a simple WS interconnection and then switch over to an ESB. Integration-in-place as a whole has several benefits. First, it extends the useful life of the existing systems by making them useful to new SOA-enabled applications. Second, it hides the unnecessary internals so that if an SP is transitioned internally, then the SCs are not impacted. Third, it leverages the investment in existing systems by taking advantage of their capabilities. Finally, it minimizes the risks and costs involved in application conversion – applications are not broken up into smaller pieces, they stay intact and just use SOA as a means of communications. However, integration-in-place does have a few pitfalls. The main problem is that this strategy does not change the fundamental characteristics of the applications that are being integrated – it just adds glues between these applications. If the applications being integrated are old, impossible to reuse, difficult to maintain, and hard to upgrade, integration-in-place will not solve these problems. In other words, it does not satisfy the 'reusable component' requirement of SOA. Due to these tradeoffs, in some cases, it may be better to look at migration instead of integration-in-place for existing systems. For example, instead of patching an old system with new interfaces ("a face lift"), it may be wiser to migrate the old system into a decomposable system with reusable components first and then integrate it.

2.3. Application migration strategies

Migrations take place from an existing (source) system to an intended (target) system. The source systems, typically monolithic legacy applications, are internally restructured and modified into target systems, in our case SOA-compliant reusable business components with well defined interfaces preferably through web services. The primary motivation for migration is to internally modify the source applications that are inflexible and hard to reuse/maintain to the newer service-oriented components that can be assembled to form different business applications quickly. Migration, within the context of SOA, involves the following two options:

1. Gradual migration: In this case, parts of an application are converted to reusable components one or few at a time.
2. Complete (Sudden) replacement: The source application is completely and suddenly replaced ('big bang' or 'cold turkey') typically with a commercial package that is hopefully SOA compliant.

Migration of applications (gradual or sudden) is an expensive and risky undertaking with costs and benefits that must be carefully weighed (Brodie and Stonebraker, 1995; Chen, 2005). This strategy is very useful if the application is inflexible and expensive to modify but is supporting a long range business function that requires frequent modifications and functional enhancements. In these cases, the strategies of surrounding these applications with WS interfaces are not enough (remember surrounding an old application with a new interface does not restructure it for flexibility). Migrations should not be undertaken if the applications serve a business function that is being phased out (i.e., the time to migrate may be longer than expected life of the application). Once migrated, the applications could use, preferably, the ESB to communicate with other applications.

Thus, there is a need for a decision model to support the strategic decisions in re-engineering for SOA. In particular, the tradeoffs between migration versus integration summarized in Table 1, need considerable analysis. The decision model introduced in the next section combines strategic and technical factors with cost benefit analysis for decisions on integration versus migration.

Table 1
Tradeoffs between integration and migration strategies

	Integration	Migration
Main plusses	• Reduces risk of failure • Protects investment in existing applications • Reduces time for implementation • Good for numerous mergers and acquisitions that keep bringing new applications to be linked to existing ones • Can be used as a transition to full migration	• Very good if the target applications are too inflexible and expensive to maintain (recall that access/integration only surrounds the old system, it does not replace it) • Takes advantage of very good COTS solutions that are increasingly becoming available • Allows a company to eliminate outdated functionality that is not needed anymore
Main minusses	• Does not work if the target applications are too inflexible and expensive to maintain (access/integration approach only surrounds the old system, it does not replace it) • Delays migration to very good COTS solutions that are increasingly becoming available • Outdated and old functionality in existing systems is perpetuated	• Can be risky (what if the new system does not work) • Investment in existing applications is lost • Can be time consuming and/or expensive • Not good for numerous mergers and acquisitions that keep bringing new applications to be linked to existing ones • Staff may need to be retrained in case of replacement with a COTS solution

3. Decision support model for SOA reengineering

To aid decision-making concerning the strategies for enterprise application re-engineering following the SOA approach, Fig. 1 shows a decision support model. The major decisions in this model are (a) strategic analysis to make integration versus migration decisions based on enterprise factors, (b) architecture analysis based on intangible cost benefit analysis of the strategic decisions, and (c) solution development based on detailed system specifications and tangible cost-benefit analysis of reengineering projects. These three decisions are discussed in detail in the next three sections.

Careful cost-benefit analysis is an important aspect of these decisions. Tangible as well as intangible costs and benefits are needed to make sound decisions. In particular, application reengineering projects must take into account the tangibles (e.g., maintenance and operation costs) plus intangibles (e.g., customer satisfaction, internal improvements, flexibility) before deploying a system. Determining the intangible aspects of IS and evaluating them against tangibles has been an elusive goal for many years. Murphy and Simon (2002) discusses the various approaches to intangibles for IT projects and use a case study methodology to show how intangibles can be included in ERP projects in different levels of decision making. We use a similar approach, as shown in Fig. 1, in which the intangibles are in-

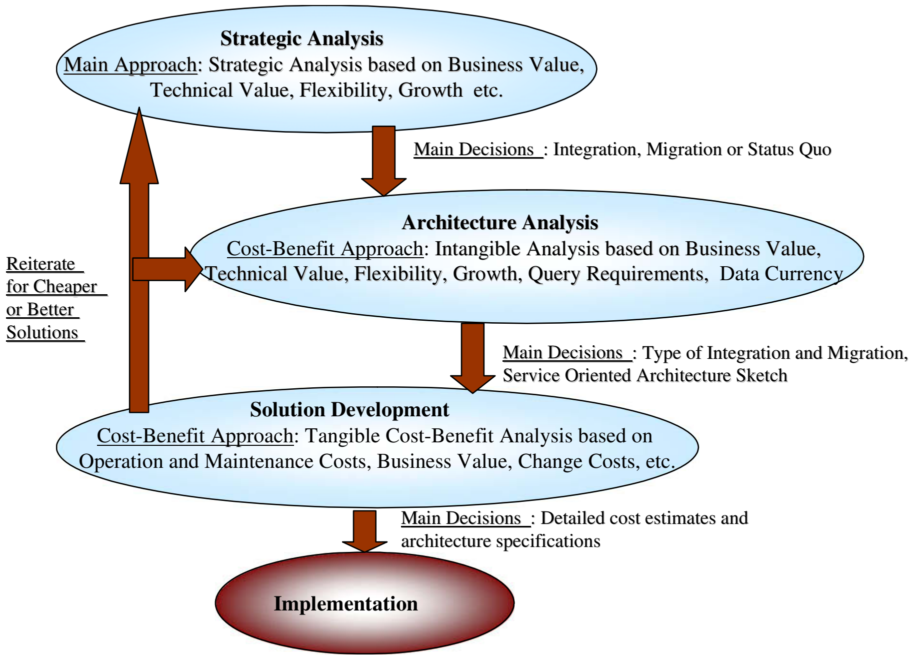
Strategic Analysis
Main Approach: Strategic Analysis based on Business Value, Technical Value, Flexibility, Growth etc.
Main Decisions : Integration, Migration or Status Quo
Architecture Analysis
Cost-Benefit Approach: Intangible Analysis based on Business Value, Technical Value, Flexibility, Growth, Query Requirements, Data Currency
Reiterate for Cheaper or Better Solutions
Main Decisions : Type of Integration and Migration, Service Oriented Architecture Sketch
Solution Development
Cost-Benefit Approach: Tangible Cost-Benefit Analysis based on Operation and Maintenance Costs, Business Value, Change Costs, etc.
Main Decisions : Detailed cost estimates and architecture specifications
Implementation

Fig. 1. Application reengineering decision support model.

cluded in the strategic and architecture analysis steps and the tangible cost-benefit analysis in the solution development step because many tangibles are difficult to determine in high level decisions. Specifically:

- Strategic Analysis should answer the question: "Should the system be reengineered or not?"
- Architectural analysis should refine strategic analysis results and suggest a reengineering strategy, based on intangibles.
- Solution development should validate the suggested strategy by exploring costs and benefits of available alternatives.

The process shown in Fig. 1 is highly iterative. The users may go through several iterations (2–4, typically) to determine solutions with better cost and technical fit. In particular, each iteration identifies and eliminates unavailable or unacceptable solution platforms, excessive cost configurations or solutions characterized by disproportionate implementation time. The number of iterations depend on the number of viable candidate solutions (typically top 2–3 solutions drive the iterations).This simple approach significantly differs from the SOA reengineering approaches discussed in the literature so far. Most approaches reported in the literature, e.g. (Bisbal, 1999; Chen, 2005; Land and Crnkovic, 2006; Serrano, 2002; Tilley, 2004; Zou et al., 2001) are partial reengineering efforts – these do not meet the 3 requirements for SOA we proposed earlier in this section. A detailed analysis of the related efforts is given in Section 8.

4. Strategic analysis of application reengineering

As stated previously, the strategies needed to deal with application reengineering for SOA fall into the following categories:

- Integration in place – consolidate them into the current and future applications by using integration technologies (e.g., WS interfaces over point-to-point or an enterprise service bus).
- Migration – transition gradually to new applications with reusable components or completely replace ("cold turkey") with SOA compliant packages.
- Do nothing – postpone the decision to a later date

The purpose of strategic analysis is to decide the most appropriate approach before implementation. In many large scale problems involving many applications, a mixture of approaches is used (i.e., some are integrated, some are migrated, and others are left alone). The exact mix and sequence of these approaches to form a strategy is based on a combination of business drivers, the technical status of the application being considered, the flexibility and growth requirements, the corporate attitude towards IT reengineering, and several other business as well as technical issues. A high level understanding of the strategies to be used can be gained by focusing initially on three key factors (see Exhibit 1 for guidelines on these three and additional factors):

Business value of the existing application. To justify any reengineering, the existing applications must support significant current as well as future business processes (Sneed, 1995). The business value can be measured in terms of contribution to profit, type of business processes supported, and market value. If needed, business value can be measured in terms of a score between 0 and 10 to indicate the importance of this application to the corporation (see Exhibit 1A).

Technical status ("friendliness") of the existing application. If the application is a very old non-decomposable and monolithic system with outdated file support, then it may be better to rewrite it at some point (Brodie and Stonebraker, 1995). For the purpose of this paper, applications can be categorized in terms of the three SOA features described previously (i.e., reusable components, well defined interfaces, and ESB). A high technical value application is 100% SOA compliant (it possesses the 3 main SOA features) and is the easiest to integrate with other SOA applications while a monolithic legacy application (it possesses no SOA requirements) requires the highest effort. An application of medium technical value only partially possesses the three features. See Exhibit 1A for details.

Flexibility and growth requirements. If the application does not need to be modified extensively for flexibility and growth, then minimal effort to integrate with new applications may be appropriate. In other words, these applications could be easily surrounded by WS interfaces because there is no need to decompose them into reusable components. Flexibility and growth requirements can be measured in terms of the number of functional and performance enhancements needed in the next few years (typically 2–3 years). See Exhibit 1A for details.

Fig. 2 shows how these three factors contribute towards a first cut understanding of what strategy may be most appropriate to reengineer applications. In general, the applications that have low business value should not be considered for reengineering (these are represented by the plane AEFD in Fig. 2). If needed, the operation and maintenance of these applications could be outsourced. On the other extreme, a great deal of time and effort should be spent on applications with high business value (the plane BCGH in Fig. 2). The applications that have low technical value (the plane ABCD in Fig. 2 are the hardest to deal with and are very expensive to reengineer. These applications are typically very old and monolithic. On the other extreme, the applications with high technical value (the EFGH plane) are relatively easy and inexpensive to reengineer and interface with newer applications. Applications in this category are, as mentioned previously, 100% SOA compliant. Lastly, the flexibility dimension also plays a key role in this analysis. Applications supporting businesses and processes that do not need to grow and change dramatically (the AEHB plane) are good candidates for integration-in-place with new applications while the applications with high flexibility and growth requirements (the CDFG plane) should be decomposed and migrated.

Table 2 shows general strategies related to BCGH plane, with applications characterized by a high business value and the notes associated with Fig. 2 indicate which points on this cube favor what strategy. Strategic analysis of this nature is extremely valuable to determine what strategies should be favored when. These strategies can be further refined by adding additional factors, as we will see below.

Notes: Point A favors do nothing strategy, Point B favors partial integration strategy, Point C definitely favors migration (Sudden), point D favors do nothing strategy (business value is too low for migration), Point E favors partial integration or do nothing strategy (no need to migrate), Point F favors full integration strategy, Point G favors full integration, and Point H favors full or partial integration strategy (no need to migrate).

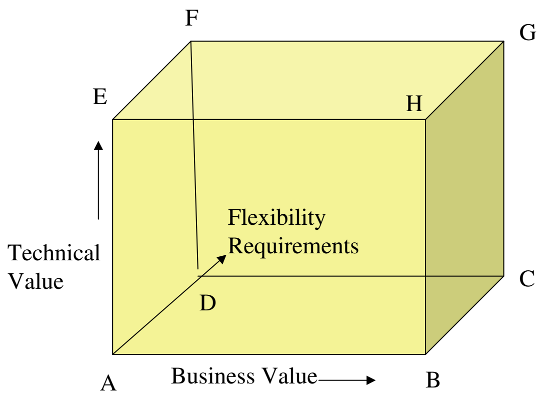

Fig. 2. Framework to analyze application reengineering strategies.

Table 2
General Strategies for reengineering High Business Value Applications (BCGH plane)

Technical value flexibility requirements	Low	High
Low	Point B: Partial integration	Point H: Integrate (Full or Partial)
High	Point C: Sudden migration	Point G: Full integration

Exhibit 1: Guidelines for ranking evaluation factors and their significance

Part A: Guidelines for ranking (low medium, high) of evaluating factors

The following list shows useful and typical evaluation factors that should be captured during the strategic analysis stage. Additional factors could be identified to capture additional details about the business domain or verticality of the project. The factors can be ranked low, medium and high by using the guidelines suggested below (these guidelines can be translated into a questionnaire, if needed). The evaluation factors and their ranking can be determined in a step in the strategic analysis stage in which all stakeholders (managers, users, consultants, and system developers) participate.

Business value of application (this is based on (Sneed, 1995)):
- *Low*: The system is not critical, it supports a business process that is being phased out or supports a low profit, low/impact business process.
- *Medium*: Important but not mission critical, supports an internal medium impact business process.
- *High*: Mission critical, long range, or highly profitable/visible business process.

Technical status of existing application (this is based on (Brodie and Stonebraker, 1995)):
- *Low*: A monolithic legacy system that cannot be decomposed into smaller pieces (i.e., presentation, data and business functionality is intermixed throughout the system). In our case, it does not satisfy any of the 3 SOA requirements.
- *Medium*: Semi-decomposable (some presentation, data and business modules can be separated but not all). In our case, the system only satisfies 1 or 2 SOA requirements.
- *High*: Modern, developed by using SOA principles (decomposable components with well defined interfaces). In our case, the system satisfies the 3 SOA requirements we stated.

Flexibility and growth requirements (Business model, related processes, supporting applications)
- *Low*: The system changes infrequently (functional and performance enhancements are needed once or twice a year or longer, example: payroll).
- *Medium*: The system changes a few time a year (functional and performance enhancements are needed 3–7 times a year).
- *High*: The system needs to change frequently (functional and performance enhancements are needed once a month or once a week).

Corporate pressure towards renovation
- *Low*: No pressure, should be done if possible.
- *Medium*: needs to be done sometime in the near future but is not urgent.
- *High*: Corporation must do this quickly and urgently to compete.

Number of applications accessed by clients for needed data
- *Low*: Less than 5.
- *Medium*: 10–20.
- *High*: More than 20.

Integration needs with other applications (this is based on business patterns (Adams et al., 2001; IBM e-Business Framework; Kalakota and Robinson, 2000)
- *Low*: one to 5 applications.
- *Medium*: 6–15 applications.
- *High*: More than 15 applications.

Part B: Guidelines for choosing evaluation factors and their significance

Each evaluation factor listed in Part A has a specific significance, representing the weight a particular factor carries in decision making. For example, some projects are primarily driven by flexibility and growth requirements. Thus flexibility and growth could be assigned higher significance. Evaluation factors and their significance values can also be assigned during the strategic analysis stage through techniques such as the following:

- Interviews with management and stake holders to solicit which factors are critical for competing in the marketplace.
- Joint application development (JAD) sessions with customers, managers and consultants and formal votes on what is important and what is not.
- Literature surveys that may indicate what is becoming important in the future for particular market segments and application types. For example. flexibility and integration needs with other applications should have very high significance for B2B trade.

5. Architectural analysis

Architectural analysis evaluates the integration and migration choices in more detail and then develops an overall architectural configuration based on SOA and appropriate platforms (e.g., an Enterprise Service Bus). The cube shown in Fig. 2 is an effective tool for "application portfolio" analysis and leads to architectural analysis by addition of other factors. Table 3 shows a decision table for more detailed analysis with additional factors such as corporate pressures towards renovation (some companies, for example, have dictated that they will renovate their application suite to comply with SOA) and integration needs with other applications (for example, payrolls do not need to be integrated with many other applications). These additional factors suggest guidelines in choosing the appropriate approach (other factors can be added to this table; moreover more fine-grained values could be set for evaluation factor). For example, Table 3 shows that complete migration could be chosen if flexibility and growth requirements are high and if corporate pressures for renovation are also high. Significance can be eventually established for evaluation factors, in order to weigh different suggestions to obtain a global strategic suggestion. A set of guidelines and suggestions for ranking (low medium, high) the evaluation factors and assigning significance to each factor are given in Exhibit 1.

Consider, for example, an application having high business value, low flexibility and growth requirements, medium technical status, and high corporate pressure towards renovation. Table 4 shows the suggested strategy (Integration, having 20 points) based on intangibles. To explain this, let us discuss the first two rows only. In the first row, the business value of the chosen application is high, hence the choice is integrate or migrate as shown in Table 3. Thus 'Do Nothing' is assigned a value of 0 and the other three options are assigned value of 12. In the second row, the flexibility and growth requirements are low, hence the choice is integration as shown in Table 3. Thus the integration choice gets a value of 4, all others have a value of 0. Note that some factors help to differentiate between strategies while others don't. For example, high values in 'Number of applications accessed by clients for needed data' and 'Integration needs with other applications' (last two factors in Table 4), differentiate between partial and full integration strategies while the others did not. This makes intuitive sense because if an existing application interacts with only one other application, a point-to-point interaction may be less expensive and easier to do but an ESB is essential if an application interacts with a large number of other applications. We now take a closer look at the integration versus migration strategies.

Table 3
Decision table for dealing with an existing system

Evaluation factor values and suggested strategy	Sample evaluation factor significance	Low	Medium	High
Business value of application	12	Do nothing	Do nothing	Reengineer (Integration; Gradual or complete migration)
Flexibility and growth requirements	4	Do Nothing/ Integration	Integration	Complete migration
Technical status of existing application	4	Migration	Migration/ Integration	Do Nothing/ Integration
Corporate pressure towards renovation	1	Integration	Migration/ Integration	Complete migration
Number of applications accessed by clients for needed data	1	Integration–Partial	Integration (Partial or Full)	Integration (Full)
Integration needs with other applications	1	Integration–Partial	Integration (Partial or Full)	Integration (Full)

5.1. Integration in place strategy: architectural issues

In many cases, integration in place is the best approach to deal with existing applications. For SOA, this strategy can be subdivided into partial (i.e., point-to-point) or full (i.e., through an ESB) options. Fig. 3a and b illustrate these two strategies.

In case of integration-partial, a WS interface is added to existing applications (i.e., the applications are 'surrounded by' WS interfaces) and applications communicate with each other by using SOAP or other WS protocols. The WS interfaces hide the service provider (SP) details from service clients (SCs) and end-user tools. However, "adapters" are needed to convert the data and protocols between SPs and SCs. Adapters can be simple programs that convert currency or sophisticated protocol converters that convert layers of technologies and semantics between SCs and SPs. In fact, development of adapters is a major cost determinant in most integration projects. For example, if an order processing application needs to communicate with 10 inventory systems with different formats and technologies, then 10 different adapters may need to be developed. We will give a classification of adapters and the cost implications in the next section.

In case of integration-full, the communication between applications is handled by an enterprise service bus (ESB) instead of point to point communication of option. SOA compliant ESBs are the next generation of enterprise application integration (EAI) platforms that emerged in the late 1990s as the key mediators for a wide range of applications (Ovum report, 1999). Fig. 3b shows a conceptual view of ESB platforms (Barry, 2003; Bieberstein, 2005). ESB platforms provide a wide range of capabilities to connect a diverse array of applications by using SOA features. First, ESBs support exchange of messages between SCs and SPs by using a publisher/subscriber model instead of a point to point connection. At the core of ESBs is an information broker that interconnects different applications – the SPs advertise their services to the broker and the sub-

Table 4
Example of Reengineering Strategy based on Intangibles

Evaluation factor values and suggested strategy	Do nothing	Migration		Integration	
		Gradual migration	Complete migration	Partial	Full
Business value of application: High	0	12	12	12	12
Flexibility and growth requirements: Low	4	0	0	4	4
Technical status of existing application: Medium	0	4	4	4	4
Corporate pressure towards renovation: High	0	0	1	0	0
Number of applications accessed by clients for needed data: High	0	0	0	0	1
Integration needs with other applications: High	0	0	0	0	1
Total	4	16	17	20	22

scribers (SCs) invoke the advertised services. Second, the ESB supports a directory service so that SCs can discover and select appropriate SPs and security services to maintain privacy and integrity of messages being exchanged. Third, The ESB provides a wide range of adapters that translate messages between SCs and SPs (many commercially available ESBs support more than 50 adapters). In addition adapter development toolkits (ADTs) are commonly provided with ESBs. An ESB platform is essentially a middleware platform that consolidates many technologies such as communications between applications, directory services, security services and message translations under a single framework. However, ESBs do not directly contribute to reusability business components – they just interconnect whatever are defined by SCs and SPs by the users.

5.2. Migration (gradual or cold Turkey): architectural issues

Migration to SOA involves decomposing of existing applications into reusable components that can be assembled to form different business applications. Migration, within the context of SOA, involves gradual migration (parts of an application are converted to reusable components gradually) or complete replacement of an existing application with a commercial package that is hopefully SOA compliant. Different architectural issues arise for these two options with the migration strategy.

The gradual migration strategy involves several steps over a period of time (could be years) and uses "surround technology" to hide the transition from client applications and end-users. The existing and target systems must coexist during the migration stage. The general approach is to use a 'migration gateway' to isolate the migration steps so that the end-users do not know if the needed information is being retrieved from an old system or a new component. See Brodie and Stonebraker (1995) for an extensive discussion of migration gateways. Development of gateways to facilitate migration is usually an expensive undertaking. Fortunately, the ESB can be used as a migration gateway because the ESB, as shown in Fig. 3b, mediates between different applications and can be used to hide the nature of the interacting applications. Consider for example, a shipping application that supports ground as well as air shipments. If this application is to be gradually migrated into two components (one for ground the other for air shipments), then an interface of this application can receive the shipping requests and send them to one application that supports both shipments or two different components – without the SCs knowing anything about it.

"Cold Turkey" migrations completely replace an existing application with a new one in a single step. This may involve complete re-write or replacement with a new package. Complete re-write is not viable in many practical situations because it is not easy to re-write application systems from scratch (see Brodie and Stonebraker (1995) for a detailed discussion). Fortunately, a large number of commercial-off-the-shelf (COTS) SOA compliant application packages are becoming available from suppliers such as SAP, Microsoft, Oracle and others. This, however, requires that the new package meets the functional, performance, and budgetary requirements of the existing system – not an easy task. The choice between complete rewrite versus buy a COTS package is based on factors such as time available to implement the new system, cost considerations, in-house expertise to develop new products, and availability of new products. For example, if a solution is needed quickly and a suitable COTS package is available, then buying or renting a COTS

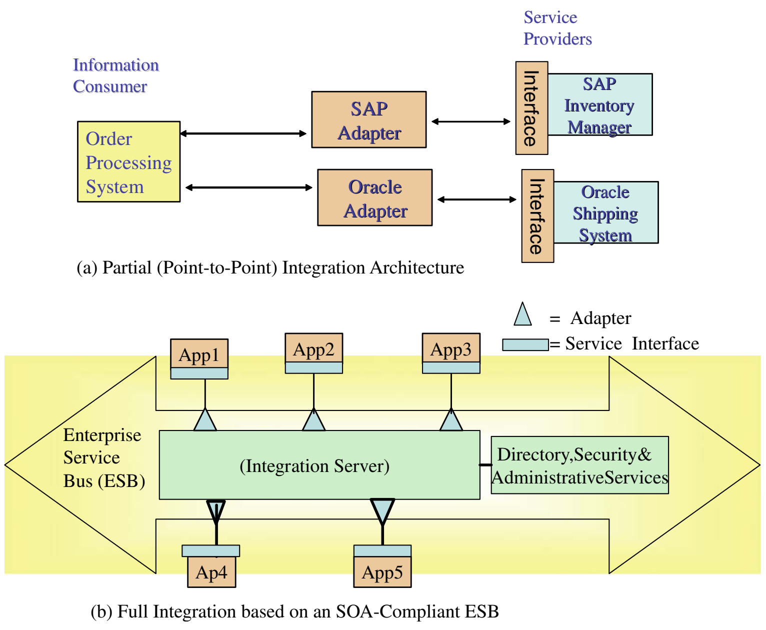

Fig. 3. Integration technologies for integration in place.

package is an obvious choice. In practice, most companies consider COTS first (it is quicker, cheaper, and more flexible in most cases). However, if suitable COTS packages are not available, then outsourcing of application rewrites is a viable option that has been available to many organizations for many years (see Dedene and DeVreese (1995) for an early example). Once again, an ESB can be used to mask the differences between the old and the new application, to some extent. For example, an ESB can serve as a migration gateway that can help in transitioning from an existing system to a new system (re-written or COTS).

5.3. Strategy 3: Do nothing (Defer)

In many practical situations, it is better to defer the reengineering task. This is especially true for applications that do not have high business value and thus short expected life. For example, if an application supports a business function that is being phased out within a year, then investment in re-architecting this application is unwise. If needed, such an application can be outsourced to an application hosting site that will maintain this application before it is phased out.

6. Solution development and detailed cost benefit analysis

This stage translates the architecture generated in the previous stage into plausible solutions with appropriate commercial-off-the-shelf (COTS) packages and detailed cost-benefit evaluations. The COTS selection requires support from a COTS database, e.g., knowledgestorm.com, so that the user can select the most appropriate solution based on cost constraints, the services needed and the technical interdependencies. Cost-benefit analysis is an important step in solution development. Fig. 4 shows a conceptual view for cost-benefit analysis of different reengineering options. Assuming that the current state of a system is S_0 and future state is S_f, then reengineering introduces an intermediate state S_i. In a status quo situation (Fig. 4a), no intermediate state is introduced and the system is operated and maintained for its expected life T1. However, changes and enhancements are typically needed even in status quo due to business drivers (there may be N such changes, each with an average cost of C1). In a reengineering scenario, shown in Fig. 4b, the system is operated and maintained between S_0 and S_i as is and then reengineered through integration or migration. Fig. 4b depicts a scenario where all changes are made to the system *after* reengineering because the changes after reengineering are assumed to be cheaper, i.e., we are assuming that C2, the average cost for a system change from S_i–S_f, is less than C1. This is a very reasonable assumption because one of the major drivers for reengineering to SOA is to develop a flexible architecture which would decrease the cost of system changes. This assumption can be easily removed without loss of generality. The following discussion first develops a simple cost-benefit model that does not consider system changes. System changes are then introduced to determine when and how the system should be reengineered.

This discussion is intended to answer the following questions in terms of costs and benefits:

- Should the current system be kept as is (status quo)?
- Should the current system be transitioned to S_i, an intermediate state, that may represent an integrated or completely new (replaced) system?
- What exactly should be S_i, should it be an integration, a migration or both?
- How can the costs and benefits for different reengineering options be estimated?

6.1. The basic cost-benefit model

Sneed (1995) suggested a simple cost-benefit model for reengineering of legacy applications that serves as a good starting point. The cost benefit analysis proposed by Sneed consists of the following simple model that is used to develop the benefits of reengineering versus status quo:

P1: Current annual maintenance cost
P2: Current annual operations cost
P3: Current annual business value
P4: Predicted annual maintenance cost after reengineering
P5: Predicted annual operation cost after reengineering
P6: Predicted annual business value after reengineering
P7: Estimated reengineering cost
P8: Reengineering risk factor
T1: Expected life of the system, expressed in year or fractions of year
T2: Estimated reengineering calendar time (assuming T1 $\gg$ T2)

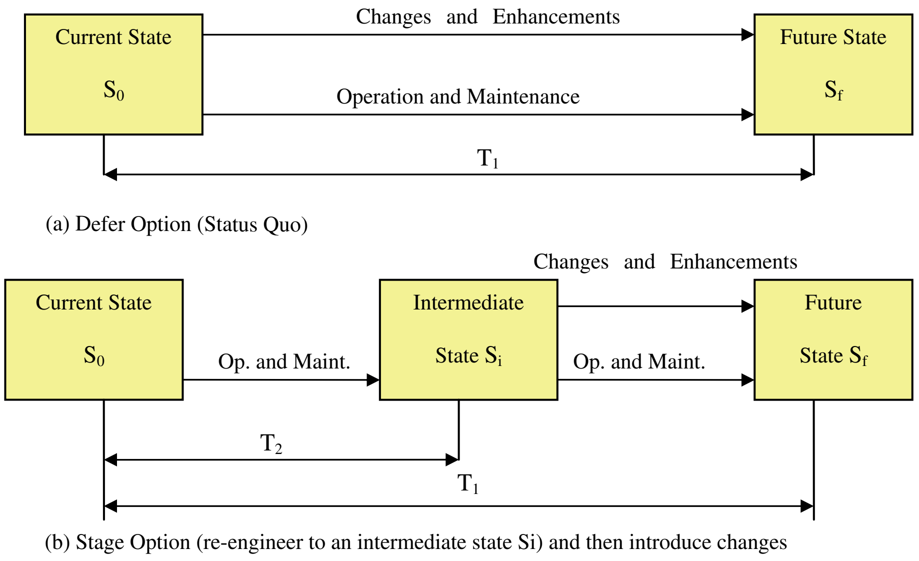

Fig. 4. A reengineering framework for cost benefit analysis.

The benefit of maintaining the status quo is given by:

$$\text{BenefitS} = [\text{P3} - (\text{P1} + \text{P2})]^{*}\text{T1} = (\text{P3} - \text{P1} - \text{P2})^{*}\text{T1} = \text{MS}^{*}\text{T1}$$

where MS is the profit margin of the status quo solution, per year.

The benefit of reengineering is similarly given by two components, which are related to time periods having, respectively the reengineered system running and the existing system running, during the reengineering implementation and transition phase:

$$\begin{aligned}\text{BenefitR} &= [(\text{P6} - (\text{P4}+\text{P5}))^{*}(\text{T1}-\text{T2}) - (\text{P7}^{*}\text{P8})] + [\text{P3} - (\text{P1}+\text{P2})]^{*}\text{T2}\\ &= (\text{P6}-\text{P4}-\text{P5})^{*}(\text{T1}-\text{T2}) - \text{P7}^{*}\text{P8} + (\text{P3}-\text{P1}-\text{P2})^{*}\text{T2}\\ \text{BenefitR} &= \text{MR}^{*}(\text{T1}-\text{T2}) - \text{TC} + \text{MS}^{*}\text{T2}\end{aligned}$$

where MR is the profit margin of the reengineered solution per year; TC, is total transition costs.

Note that this model does not include system changes. For system changes, we introduce the following additional parameters, without considering an estimated increased business value:

C1: Average cost per change on current system (status quo).
C2: Average cost per change after system reengineering.
N: Number of changes introduced in the system.

Then the following updated expressions for benefits hold:

$$\begin{aligned}&\text{BenefitS}' = \text{BenefitS} - N^{*}C1\\ &\text{BenefitR}' = \text{BenefitR} - N^{*}C2\\ &\text{BenifitRS} = \text{Benefit of reengineering versus status quo}\\ &\quad = \text{BenefitR}' - \text{BenefitS}' = \text{BenefitR} - \text{Benefit}S + N^{*}(C1 - C2)\\ &\quad = \text{BenefitR} - \text{BenefitS} + \text{MC}\ldots \qquad (1)\end{aligned}$$

where MC is the total marginal cost savings due to lower cost of changes after the reengineering process.

According to Eq. (1), BenefitRS increases by a value MC assuming that $C1 > C2$ and BenefitS is constant. This implies that if a system is to be changed more frequently, then the benefit of reengineering is higher. In addition, BenefitRS = 0 in Eq. (1) gives

$$\text{BenefitS} = \text{BenefitR} + \text{MC} \qquad (2)$$

Eq. (2) ties the benefit of reengineering and status quo (BenefitR and BenefitS) to number of changes. Thus even if BenefitR is lower than BenefitS, the cost savings due to changes makes status quo equivalent to Re engineering. This equation can be used also to decide when exactly the reengineering should be undertaken, i.e., what is the value of T2. Exhibit 2 (Part A) shows that T2 can be expressed as following:

$$\text{T2} = \text{T1} - (\text{TC—MC})/(\text{MR—MS}) \qquad (3)$$

Eq. (3) shows estimated re-engineering calendar time (T2) in terms of the expected life of the system (T1), transition cost (TC), marginal cost savings due to changes (MC), profit margin of the reengineered solution (MR), and profit margin of the status quo (MS). This equation displays several important characteristics of transition to SOA. First, marginal cost saving due to changes (MC) is an important motivation for introducing SOA because SOA can allow new services to be introduced and existing services to be extended inexpensively due to reusable components that can be invoked over an ESB. Second, the transition cost (TC) is mitigated by MC, thus the cost of transition to SOA can be recovered by reduced costs of changes. Third, a higher (TC–MC) decreases the re-engineering time (T2), i.e., it suggests a quicker transition to SOA (the solution implementation must be done in short time, T2 is low, otherwise it is not convenient). Finally, T2 is more sensitive to TC and MC changes when MR and MS are close (MR should be greater than MS). This equation can be easily modified for greater detail by substituting MS, MR and MC in terms of different types of costs (C1, C2, P1, P2, P4, P5, P7), system business values (P3, P6) and number of expected changes (N).

The model presented so far only handles two strategies (status quo or reengineering). This model can be effortlessly modified and extended to handle the three basic strategies (status quo, integration and migration) as shown in Exhibit 2 (Part B).

An important issue is to estimate the reengineering cost (P7). The cost and effort needed to reengineer systems depends largely on the reengineering strategy chosen. For example, integration costs differ from migration costs. Specifically, the cost of integration is based on the number and type of adapters needed, and the cost estimation for migration depends on the gradual migration versus replacement decisions. In addition, costs of transitions (C1, C2) depend on current state (S_0) of system (monolithic, data decomposable, highly decomposable). Specifically, the flexible architectures based on SOA and MDA are easier and cheaper to integrate than rigid monolithic architectures. The following discussion addresses these issues in terms of integration costs versus migration costs.

Exhibit 2: Quantitative cost/benefit analysis

Part A: Estimating Reengineering Time T2

Starting with the benefit of status quo and reengineering without changes and with changes given by:

$$\begin{aligned}&\text{BenefitS} = (\text{P3—P1—P2}) * \text{T1} = \text{MS} * \text{T1}\\ &\text{BenefitR} = (\text{P6—P4—P5}) * (\text{T1—T2}) - \text{P7} * \text{P8} + (\text{P3—P1—P2}) * \text{T2}\\ &= \text{MR} * (\text{T1—T2}) - \text{TC} + \text{MS} * \text{T2}\\ &\text{BenefitS}' = \text{Benefit with changes} = \text{BenefitS} - N * \text{C1}\\ &\text{BenefitR}' = \text{Benefit with changes} = \text{BenefitR} - N * \text{C2}\\ &\text{BenifitRS}' = \text{Benefit of reengineering versus status quo}\\ &\qquad = \text{BenefitR}' - \text{BenefitS}' = \text{BenefitR} - \text{BenefitS} + \text{MC} \qquad (6)\end{aligned}$$

Then, BenefitRS′ = 0 in Eq. (6) gives

$$\text{BenefitS} = \text{BenefitR} + \text{MC} \qquad (7)$$

Substituting the values for BenefitS and BenefitR in Eq. (7), we get:

$$\text{MS} * \text{T1} = \text{MR} * (\text{T1—T2}) - \text{TC} + \text{MS} * \text{T2} + \text{MC}$$

Reorganizing the formula we get:

$$\begin{aligned}\text{T2}(\text{MR} - \text{MS}) &= (\text{MR} - \text{MS})\text{T1} - \text{TC} + \text{MC}\\ \text{T2} &= [(\text{MR} - \text{MS})\text{T1} - \text{TC} + \text{MC})]/(\text{MR} - \text{MS})\end{aligned}$$

Finally we get

$$\begin{aligned}\text{T2} &= \text{T1} - \text{TC}/(\text{MR} - \text{MS}) + \text{MC}/(\text{MR} - \text{MS})\\ \text{T2} &= \text{T1} - (\text{TC} - \text{MC})/(\text{MR} - \text{MS})\end{aligned}$$

Part B. High Level Cost-Benefit Analysis for Integration and Migration

The following parameters can be used to quantify cost/benefit analysis.

R1: Current annual maintenance cost
R2: Current annual operations cost
R3: Current annual business value
R4: Predicted annual maintenance cost after integration
R5: Predicted annual operation cost after integration
R6: Predicted annual business value after integration
R7: Estimated integration cost
R8: Integration risk factor
R9: Predicted annual maintenance cost after migration
R10: Predicted annual operations cost after migration
R11: Predicted annual business value after migration
R12: Estimated migration costs
R13: Migration risk factor
T1: Expected life of the system (time between S_0 and S_f)
T2: Estimated integration calendar time (time between S_0 and S_i)
T3: Estimated migration calendar time (time between S_0 and S_i)

The benefit of maintaining the status quo is given by:

$$\text{BenefitS} = [R3 - (R1 + R2)] * T1$$

The benefit of integration is similarly given by:

$$\text{BenefitI} = [R6 - (R4 + R5)] * (T1 - T2) - (R7 * R8) + [R3 - (R1 + R2)] * T2$$

The benefit of migrating the system is given by:

$$\text{BenefitM} = [R11 - (R9 + R10)] * (T1 - T3) - (R12 * R13) + [R3 - (R1 + R2)] * T3$$

T2 and T3 can be calculated as function of the various parameters in a similar manner as exposed in Exhibit 2, Part a).

6.2. Integration costs

The cost and effort needed to integrate systems depends largely on the number and nature of integration adapters. For example, if an application interacts with three inventory applications of different type (database, well defined API, monolithic legacy) that reside on different platforms (MS Windows, Linux and MVS-mainframe), then three different type of adapters (remote SQL, SOAP, and screen-scraper) can be used. In general, based on the knowledge about the number and type of adapters needed in an integration project, rough estimates of effort and cost can be obtained by using techniques similar to function point analysis (Symons, 1991). The following discussion elaborates on this point and proposes an algorithm to estimate cost of integration projects.

Adapters can be simple (e.g., a currency converter) or complex (a protocol converter with semantic considerations). In addition, an adapter could be defined at a B2B integration or enterprise application integration (EAI) levels to integrate application plans of two enterprises or all applications of a single enterprise, respectively. Let us first deal with adapter complexity as a measure of distance/cost between two different applications. For instance, if two applications belong to the same package by the same vendor, the distance will be 0, if they both offer Web Services the distance will be small, if one of them is still working on a legacy system the distance will be great. However, "technology compliance" is not the only measure to estimate distance and complexity of an adapter. "Semantic compliance" is an important aspect of system integration and is a driver for many enterprise ontology efforts (Fox and Gruninger, 1997). We are thus proposing a two dimensional measure to estimate the complexity of an adapter: technology compliance and semantic compliancy (see Fig. 5). Naturally an adapter between two applications that does not use the same technology as well as semantics will be more complex because the distance between the two applications is higher. This measure may allow us to estimate the cost of integration by suggesting adapters for different values of this attribute (see Fig. 6).

For a preliminary estimation of integration costs, consider the following example. Four applications have to be selectively integrated; in the example four point-to-point integrations paths have been defined; each integration path has a specific distance d_i. If application 3 and application 4 do not need to be integrated, then (e.g., app 2 just needs to be integrated with 3 apps from 3 different suppliers), $d4$ would not be needed.

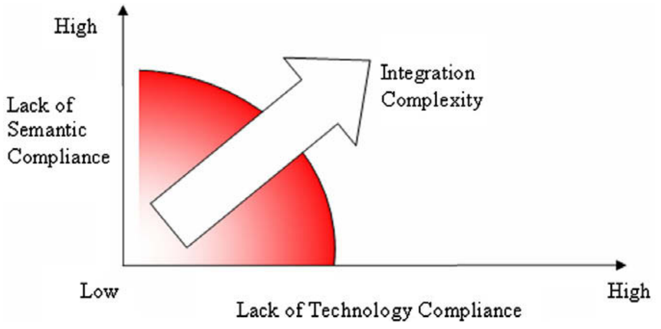

Fig. 5. Two dimensional view of integration complexity.

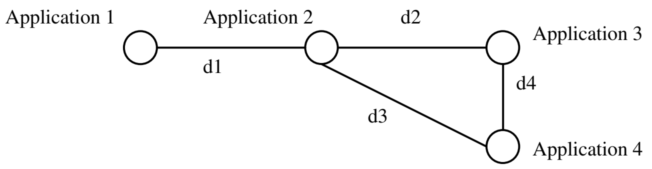

Fig. 6. Integration paths and distances in a point-to-point integration environment.

The following algorithm uses d to estimate cost of an integration effort:

- Given N integration paths and N distances d_i between two applications of one integration path.
- $L(d)$ = lines of code (approximate) needed to implement d. This can be based on techniques similar to function point analysis (Symons, 1991).
- $C(d)$ = cost of integration for a distance $d = G.\ (L(d)/P)$ where P = productivity in terms lines of code per day (e.g., 20) and G is the wages per day (e.g., \$500 in the US) for a software developer.
- Estimated cost for integration adapters=

$$I1 = \sum_{i=1}^{N} C(d_i) = \sum_{i=1}^{N} \frac{G \cdot L(d_i)}{P} \tag{4}$$

- Add the fixed cost of integration adapter and yearly fee to get the total estimated cost of integration I:

  $I = I1 + I2 + I3(T1 - T2)$

  where $I2$ and $I3$ are the purchase cost and yearly fee of integration gateway, respectively.

This algorithm can be easily adapted to handle the usage of integration platforms instead of specific adapters; in that case, generally $L(d)$ decreases while fixed costs and yearly fee increase.

6.3. Migration (gradual and replacement) costs

The replacement costs due to the cold turkey approach depend on vendor specified cost ($Q1$) plus yearly fee ($Q2$). Thus the total replacement cost is given by:

$Q1 + Q2(T1–T3)$ where $Q1$ and $Q2$ are the purchase cost and yearly fee of the new package, respectively.

The gradual conversion cost must also include the cost of conversion plus the migration gateway cost to allow a gradual migration. The following algorithm estimates the cost of a gradual migration effort:

- Given N modules to be migrated.
- L_x = lines of code (approximate) needed to implement a migration gateway for module x. This can be based on techniques similar to function point analysis (Symons, 1991).

- C_x = cost of developing module $x = G.\ (L_x/P)$ where P = productivity in terms lines of code per day (e.g., 20) and G is the wages per day (e.g., \$500 in the US) for a software developer.
- Estimated cost for migration cost

$$M1 = G\sum_{x=1}^{N}(Lx/P) \tag{5}$$

- Add the fixed cost of migration gateway and yearly fee to get the total estimated cost of gradual migration M:

  $M = M1 + M2 + M3(T3)$

  where $M2$ and $M3$ are the purchase cost and yearly fee of the migration gateway, respectively.

7. Case study – wondermeat company

The decision model presented so far has been incorporated into an intelligent decision support system for SOA planning. This system, known as PISA (Planning, Integration, Security and Administration) environment, at present provides automated support for the users to develop and evaluate integrated architectures for SOA for a wide range of business scenarios based on business patterns (Adams et al., 2001; IBM e-Business Framework; Kalakota and Robinson, 2000). Detailed discussion of PISA is beyond the scope of this paper (additional details can be found at the PISA Website (NGE Solutions PISA website)). PISA has been used in several practical situations. The following case study exemplifies a situation that involved detailed tradeoffs between integration and migration.

WonderMeat is a small company working in the food market. The company business model covers a large part of the process chain, from slaughtering to processing and distribution of single portions of food, meat sauces, or large meat cuts. The core processes of meat transformation of WonderMeat are already well organized; it is also a common knowledge that the meat packing industry is very labor-intensive since it is not, and cannot be, highly automated. WonderMeat wants to increase the performance of the support processes, particularly the ones related to order entry and fulfilment. They are facing two types of problems:

- Lack of internal integration; WonderMeat information systems are mainly based on two applications, developed by different vendors; these applications are initially completely decoupled:
  - APP1: covers general ledger, accounts and finance, sales orders entry, generation of delivery documents, sales invoices, sales analytics, and more;
  - APP2: specifically designed for the meat market, covers sales orders fulfilment, traceability and generally the transform chain;
- Lack of integration with key customers (5 customers, 4% of the total, are responsible for 80% of the orders and orders value; WonderMeat has a very concentrated Pareto curve); these customers are delivering large and frequent orders.

This scenario covers internal integration as well as B2B integration and illustrates how the approach presented above is used in this case.

7.1. Strategic and architectural analysis

Strategic analysis has been used to answer the question "Should the system to be re-engineered or not?" This analysis is particularly important for internal integration (generally, a company cannot impose a system re-engineering to business partners!), but is useful also for B2B integration since it can clarify if internal application are good candidate for B2B integration. Strategic analysis has been applied to both WonderMeat applications; Table 5 shows the corresponding strategic values (these values were determined based on the guidelines suggested in Exhibit 1A).

For both applications an integration in place strategy is initially suggested. WonderMeat added a few more intangible attribute values with significance values, to corroborate the suggestion. Tables 6 and 7 show the expanded analysis for APP1 and APP2, respectively. The values in these tables were determined based on the guidelines suggested in Exhibit 1. It can be seen that in both cases the suggestion for an integration strategy has been validated. In addition, the additional factors show that for APP1 partial-integration (point-to-point) is slightly better and for APP2 full integration (ESB) is somewhat better. Based on this analysis, the company is adopting the strategy full integration by using commercially available ESBs because it has higher long range benefits.

7.2. Solution development and detailed cost benefit analysis

The evaluation of costs and benefits has been based by WonderMeat on the Basic Model. In fact, WonderMeat has a quite stable business model and is not planning substantial business transitions in the following years. The values of parameters for cost and benefit estimation are shown in Table 8, which summarizes costs and benefits for both applications and includes operation benefit evaluation of the B2B integration too.

Maximum reengineering calendar time can be estimated using the formulas shown in Exhibit 2.

$$\begin{aligned}&\mathrm{MS} = \mathrm{P3} - \mathrm{P1} - \mathrm{P2} = 2200\\&\mathrm{MR} = \mathrm{P6} - \mathrm{P4} - \mathrm{P5} = 2540\\&\mathrm{TC} = 400\\&\mathrm{MC} = 0(\text{basic model})\\&\mathrm{T2} = \mathrm{T1} - (\mathrm{TC})/(\mathrm{MR} - \mathrm{MS})\end{aligned}$$

For WonderMeat the calculated value is:

$$\mathrm{T2} = 10 - 400/(2540 - 2200) = 8,82 \text{ years}$$

The calculated value is much greater then the estimated value, advising a concrete and quick benefit of integration. This was expected, since all the values of parameters are estimated to have much better values after the integration.

Solution overview

The number of applications to be integrated is relatively small:

- 2 internal applications, to be integrated together;
- 5 external applications to be integrated with the internal APP1.

A point to point architecture could be used, but WonderMeat preferred to focus on an EAI framework that supports most of the features of an ESB. Some very interesting open source solutions are available today; WonderMeat decided to use the Mule framework (Mule Framework), considering that:

- the cost of the framework is nearly zero;
- the cost of maintenance and support slightly increases current maintenance cost;
- the cost of the integration is mainly concentrated on consulting services;

Table 5
Strategic analysis for wondermeat applications

Application	Business value	Flexibility requirements	Technical value
APP1	High	Low	High
APP2	High	Low	Medium

Table 6
WonderMeat reengineering strategy based on intangibles for APP1

Evaluation factor values and suggested strategy	Significance	Do nothing	Migration		Integration	
			Gradual migration	Complete migration	Partial	Full
Business value of application: High	12	0	12	12	12	12
Flexibility and growth requirements: Low	4	4	0	0	4	4
Technical status of existing application: High	4	4	0	0	4	4
Number of applications accessed by clients for needed data: Low	1	0	0	0	1	0
Integration needs with other applications: Low	1	0	0	0	1	0
Total	22	8	12	12	22	20

Table 7
WonderMeat reengineering strategy based on intangibles for APP2

Evaluation factor values and suggested strategy	Significance	Do nothing	Migration		Integration	
			Gradual migration	Complete migration	Partial	Full
Business value of application: High	12	0	12	12	12	12
Flexibility and growth requirements: Low	4	4	0	0	4	4
Technical status of existing application: Medium	4	0	4	4	4	4
Number of applications accessed by clients for needed data: High	1	0	0	0	0	1
Integration needs with other applications: High	1	0	0	0	0	1
Total	22	4	16	16	20	22

Table 8
Cost estimates

Parameter	Value	Unit of measure
P1: Current annual maintenance cost	50	Thousand USD per year
P2: Current annual operations cost	250	Thousand USD per year
P3: Current annual business value	2500	Thousand USD per year
P4: Predicted annual maintenance cost after reengineering	60	Thousand USD per year
P5: Predicted annual operation cost after reengineering	150	Thousand USD per year
P6: Predicted annual business value after reengineering	2750	Thousand USD per year
P7: Estimated reengineering cost	200	Thousand USD
P8: Reengineering risk factor	2	–
T1: Expected life of the system	10	Years
T2: Estimated reengineering calendar time	1,5	–

- the definition of integration solutions inside the framework requires only the writing of configuration files, and practically no code in the simplest case, taking advantage of standard and predefined integration components;
- the framework can be used to increase internal information exchange between the two applications, to a high extent; the main elements that will be automatically exchanged are:
  – from APP1 to APP2, definition of items, and customers, new order headers and lines;
  – from APP2 to APP1, fulfilled or partially fulfilled order headers and lines;
- the framework will also be used for B2B integration, with minimal requirements for modifications on information systems of the partners;
- the use of the framework helps to reduce distance between applications to be integrated (integrations are easier and cheaper).

Fig. 7 shows an overview of the Mule framework architecture and conceptually illustrates the availability of prepackaged adapters that can be used in several integration scenarios (see Mule Framework for additional configurations and details about the Mule system). In fact, WonderMeat used only available Mule components, and wrote a configuration and orchestration file to:

- automate internal integration, and particularly data exchange between the two applications, defining Mule file connectors;
- automate B2B data exchange defining Mule ftp connectors and XSL transformers.

In the first B2B integration release, key customers are required to make an XML version (customer dependent) of their orders and deposit it on a secure FTP server. Here's a small snapshot of Mule configuration file, where a standard XSL transformer has been defined.

```
⟨transformers⟩
    ⟨transformer name="Transform_01"
        className="org.mule.
        transformers.xml.XsltTransformer"⟩
        ⟨properties⟩
            ⟨property name="xslFile" value="xsl/order.xslt"/⟩
        ⟨/properties⟩
    ⟨/transformer⟩
⟨/transformers⟩
```

WonderMeat has already planned future enhancements that will be focused on a better B2B integration, in particular related to item definition updates and order fulfilment confirmation; now new items, a very infrequent case, are manually loaded in the systems of the 5 key customers. They decided to use only one iteration because their solution space was driven by the Mule platform (it is an open source platform that was already operational in the company).

8. Comparison with related work and expected contribution

A great deal of the literature on SOA is being published but systematic approaches to reengineer applications for SOA are virtually

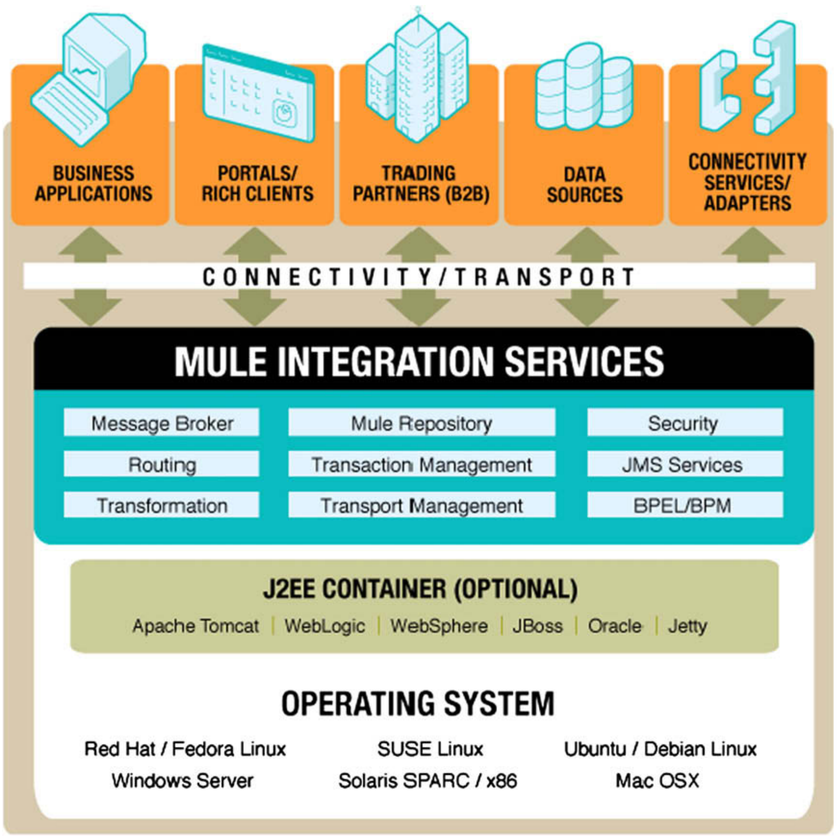

Fig. 7. Mule architecture overview (see Mule Framework for details).

non-existing. Most of the current literature in this area is from consulting firms that want to market their services (see, for example, (SOA Systems website)). However, a few high level reengineering approaches for SOA have been reported with no analysis of migration versus integration issues. This is probably due to the relative newness of SOA. In addition, general application reengineering literature has been published that deals with enterprise integration and migration under headings such as enterprise application integration (EAI), ERP systems, and legacy application reengineering. For a thorough analysis, we divided the literature review into two parts: examination of the general application reengineering literature for general guidance and then the SOA reengineering literature for detailed comparison.

The general application reengineering literature includes textbooks (Brodie and Stonebraker, 1995; Lithicum, 2000; Umar, 1996) and research papers (Boehm and Abts, 1999; Bussler, 2002; Chalmeta et al., 2001; Lee et al., 2003; Losavio et al., 2005; Murphy and Simon, 2002; Sharif, 2004; Sneed, 1995; Sutherland and Heul, 2002; Van der Enden et al., 2001). We reviewed the aforementioned literature to locate an approach and/or a decision support tool that addresses the tradeoffs between integration and migration. The research literature so far consists of different definitions and conceptual models of EAI (Lee et al., 2003; Sharif, 2004), B2B integration (Bussler, 2002), case studies (Van der Enden et al., 2001), development and comparison of EAI frameworks and reference architectures (Chalmeta et al., 2001; Losavio et al., 2005), and developments in enterprise ontologies (Fox and Gruninger, 1997; Maedche et al., 2003) for common vocabularies. This literature does not directly help the managers to support integration versus migration decisions. Instead, many ideas are dispersed across different papers that can be stitched together to formulate a methodology. However, Sneed (1995) does present a legacy application re-engineering planning approach that identifies the basic ideas we have used in this paper. The main limitation of Sneed's paper is that it does not go into the technical tradeoffs between the migration and integration strategies and the cost-benefit model is somewhat simplistic. As discussed in Section 5, we have considerably extended the Sneed model.

The specific literature related to reengineering for SOA consists of several sources (Chen, 2005; Chung and Davalos, 2007; Kontogiannis et al.; Zhang et al., 2005). The work by Zhang et al. (2005) is the closest to our research because it describes a four step reengineering process that transitions legacy applications to SOA. However, this is a high level (only 5 page long) paper that describes an overall approach for reengineering without technical details, cost-benefit analysis, and tradeoffs between migration versus integration decisions. We also disagree with Zhang et al. (2005) assertion of a two step process where the first migration is from monolithic systems to Web-based systems followed by a migration from Web-based systems to service- oriented systems. In our practical experience, we have not observed this practice – the reality is an iterative process that is closer to the model presented in Fig. 1. Chung and Davalos (2007) uses feature analysis to develop an approach to support service-oriented reengineering with a focus on migration by using service identification and packaging processes. Kontogiannis et al. presents a research perspective for maintenance and reengineering of SOA by outlining the challenges in 3 domains (business, engineering, and maintenance) without suggesting specific approaches. Chung and Davalos (2007) briefly describes an application reengineering methodology that can help reengineering of tightly coupled legacy information systems into loosely coupled service-oriented information systems. Several additional papers have been published on partial reengineering efforts (these do not meet the 3 requirements for SOA we proposed in Section 2). For example, some efforts show how to reengineer typical legacy systems to a Web interface (Bisbal, 1999). In addition, the INSIDE project (Lavery, 2004) investigated the problems and possible solutions for incrementally evolving the existing systems into Web-based services. Similarly, Serrano (2002) defines a reengineering environment that assists with the migration of legacy systems to distributed object environments and Tilley (2004) provides a perspective of the business value and technical challenges of adopting Web services. Zou et al. (2001) presents a framework to address the issues on migrating legacy systems into a Web enabled environment involving CORBA wrapper and SOAP/CORBA IDL translator and Litoiu (2004) describes a performance engineering approach for legacy systems to migrate to Web services based on latency and scalability. Arsanjani (2002) proposes to migrate enterprise architecture to a service-oriented architecture by externalising useful business services, which are implemented in large-grained enterprise components.

In short, we have benefited from the general application reengineering literature, in particular the reengineering methodology presented by Sneed (1995) and the classification of migration issues presented by Brodie and Stonebraker (1995). We have also used the specific literature related to reengineering for SOA (Chen, 2005; Chung and Davalos, 2007; Kontogiannis et al.; Zhang et al., 2005) to determine what seems to be missing. Based on this analysis, we feel that this paper makes significant practical contribution to the literature at several levels. First, it clearly identifies the main integration and migration issues for SOA. Second, it captures the essence of strategic and cost benefit decisions in SOA reengineering projects and pushes the purely technical and vendor product decisions to lower level choices in later stages. This is important because application reengineering is not a strictly technical issue; it entails business plus technical issues, thus our model is useful both for practitioners and for managers. This paper also attempts to synthesize the strategic, architectural, and technical reengineering ideas that are dispersed over different papers into a decision model to help the managers in deciding what approach to use when and how. In addition, the proposed decision model has been instantiated and incorporated into an intelligent decision support system, known as planning, integration, security and administration (PISA) environment, that provides automated support for the users to develop and evaluate integrated architectures for SOA (see (NGE Solutions PISA website) for additional details about PISA). Future research will extend this model as we use PISA in different SOA consulting assignments and gain additional insights. This instantiation and use in real-life situations significantly increases the practical value of this work. Finally, the model presented in this paper can be generalized for many non-SOA situations that require tradeoffs between integration versus migration. In fact, we first used an earlier version of this model in the telecom sector to migrate applications from non-CORBA to CORBA environments in the late 1990s. The main differences were that the target (“new” in 1997) environment was a CORBA broker instead of an SOA ESB. Thus the three compliance requirements specified in Section 2 were defined in terms of CORBA (e.g., distributed objects, CORBA bus, etc). However, the overall decision model shown in Fig. 1 stayed the same. Thus for any other, perhaps new, target environment, the compliance requirements for the target environment need to be defined and the decision model needs to be customized slightly for the target environment.

9. Concluding comments

To conclude, comprehensive reengineering approaches that address integration versus migration tradeoffs for SOA are virtually non-existing. As noted above, the general and older literature on application reengineering is only partially applicable to SOA while more recent research on reengineering for SOA is either too high level or does not examine integration versus migration tradeoffs in detail. The discussion in most of the literature is oriented towards software engineers and less for IT managers who are concerned about costs and strategic tradeoffs. This paper has presented a management decision model for migration versus integration tradeoffs specifically for SOA and combined strategic factors with cost benefit analysis for the key decisions. We have incorporated this decision model into an intelligent decision support system for SOA planning and have explained the proposed model through a case study. Work of this nature has not been reported in the literature. We strongly feel that formulation of management decision models for integration or migration decision making for SOA is an important area of work that requires more attention. This research is an important step in that direction.

References

Adams, J. et al., 2001. Patterns for e-Business: A Strategy for Reuse. IBM Press.

Arsanjani, A., 2002. Developing and integrating enterprise components and services. Communications of the ACM 45 (10), 31–34.

Barry, D., 2003. Web Services and Service-Oriented Architectures: The Savvy Manager’s Guide. Morgan Kaufman.

Bieberstein, N. et al., 2005. Service-Oriented Architecture (SOA) Compass: Business Value Planning and Enterprise Roadmap. IBM Press.

Bisbal, J. et al., 1999. Legacy Information Systems: Issues and Directions, IEEE Software.

Boehm, B., Abts, C., 1999. COTS integration: plug and pray. IEEE Computer 32 (1), 135–138.

Brodie, M., Stonebraker, M., 1995. Migrating Legacy Systems. Morgan Kaufman.

Bussler, C., 2002. The of B2B engines in B2B integration architectures. SIGMOD Record 31 (1), 57–72.

Chalmeta, R., Campos, C., Grangel, R., 2001. References architectures for enterprise integration. Journal of Systems and Software 57 (3), 175–191.

Chen, F. et al., 2005. Feature analysis for service-oriented reengineering. Software Engineering Conference, 2005. APSEC’05. 12th Asia-Pacific, 15–17, 8 p.

Chung, Davalos, 2007. ervice-Oriented Software Reengineering: SoSR. 40th Annual Hawaii International Conference on System Sciences (HICSS’07), p. 172c.

Dedene, G., DeVreese, J., 1995. Realities of off-shore reengineering. IEEE Software, 35–45.

Erl, T., 2005. Service-oriented achitecture: concepts. Technology and Design. Prentice Hall PTR.

Fox, M., Gruninger, M., 1997. On Ontologies and Enterprise Modeling. International Conference on Enterprise Integration Modeling.

IBM e-Business Framework website http://www-106.ibm.com/developerworks/patterns/.

Kalakota, R., Robinson, M., 2000. e-Business 2.0 – Roadmap for Success. Addison Wesley.

Kontogiannis, K. Lewis, G. and Smith, D., “The Landscape of Service-Oriented Systems: A Research Perspective for Maintenance and Reengineering”, Site: http://www.cs.vu.nl/csmr2007/workshops/1-%20PositionPaper-SOAM-v2-4.pdf.

Land, R., Crnkovic, I., 2006. Software systems in-house integration: architecture process practices and strategy selection. Journal of Information and Software Technology. Elsevier. vol. 49(5), p. 419–444.

Lavery, J. et al., 2004. Modeling the evolution of legacy systems to web-based systems. Journal of Software Maintenance and Evolution: Research and Practice 16, 5–30.

Lee, J., Siau, K., Hong, S., 2003. Enterprise Integration with ERP and EAI. CACM 46 (2), 54–60.

Lithicum, L., 2000. Enterprise Application Integration. Addison Wesley.

Litoiu, M., 2004. Migrating to web services: a performance engineering approach. Journal of Software Maintenance and Evolution: Research and Practice 16, 51–70.

Losavio, F., Ortega, D., Pérez, M., 2005. Comparison of EAI frameworks. Journal of Object Technology 4 (4), 93–114.

Maedche, A., et al, 2003. Ontologies for Enterprise Knowledge Management. IEEE Intelligent Systems.

Morgenthal, JP. 2000. B2B Integration: Marketplaces vs Point-to-Point, eAI Journal.

Murphy, K., Simon, S., 2002. Intangible benefits valuation in ERP projects. Information Systems Journal 12, 301–321.

Ovum report, “Enterprise Application Integrators”, Ovum Group, 1999.

Ruh, W., Maginnis, F., Brown, W., 2002. Enterprise Application Integration. John Wiley.

Serrano, M. et al., 2002. Reengineering legacy systems for distributed environments. Journal of Systems and Software 64, 37–55.

Sharif, A. et al., 2004. Integrating the IS with the enterprise: key EAI research challenges. Journal of Enterprise Information Management 17, 164–170.

Sneed, H., 1995. Planning the reengineering of legacy systems. IEEE Software, 24–34.

Stojanovic, Z., Dahanayake, A., 2004. Service-Oriented Software System Engineering – Challenges and Practices. IDEA Group Publishing.

Sutherland, J., Heul, W., 2002. Enterprise application integration and adaptive complex systems. CACM 45 (10), 59–64.

Symons, C.R., 1991. Software Sizing and Estimating: Mk II FPA (Function Point Analysis). John Wiley.

Tilley, S. et al., 2004. On the business value and technical challenges of adopting web services. Journal of Software Maintenance and Evolution: Research and Practice 16, 31–50.

Umar, A., 1996. Application (Re) Engineering. Prentice Hall.

Umar, A., e-Business and Distributed Systems Handbook: Integration Module. <www.amjadumar.com>, August 2004.

Umar, A., e-Business and Distributed Systems Handbook: Architecture Module. <www.amjadumar.com>, August 2004.

Van der Enden, S. et al., 2001. A Case Study in Application Integration. OOPSLA Workshop on Business Objects and Components, Minneapolis.

Zhang, Z., Liu, R., Yang, H., Service identification and packaging in service-oriented reengineering. In: Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Graduate School, 2005. <http://www.cse.dmu.ac.uk/STRL/research/publications/pubs/2005/2005-8.pdf>.

Zou, Y., Kontogiannis, K., Towards a web-centric legacy system migration framework. In: Proceedings of the third International Workshop on Net-Centri Computing (NCC): Migrating to the Web, International Conference on Software Engineering (ICSE'01), Toronto, Canada, 2001.

Weblink1: SOA Portal at: <http://www.service-architecture.com/>.

Weblink2: IEEE Computer Society Technical Committee on Services Computing – <www.servicescomputing.org>.

Weblink3: IBM site on SOA <www.ibm.com/soa>.

Weblink4: Sun site on SOA <www.sun.com/soa>.

Weblink5: Mule framework <http://mule.mulesource.org/display/MULE/Home>.

Weblink6: The open group website <www.opengroup.org>.

Weblink7: SOA Systems website <www.soasystems.com>.

Weblink8: NGE solutions PISA website <www.ngesolutions.com/pisa>.

Amjad Umar received an M.S. in Computer and Communication Engineering and a Ph.D. in Information Systems Engineering, both from the University of Michigan. He has more than 20 years of senior management, industrial consultation and academic experience in different aspects of distributed information systems, has authored 6 books, and published over 50 papers. He is a Fulbright Senior Specialist with the US Council of International Exchange of Scholars and a Senior Advisor to the United Nations initiative on 'ICT for Developing Countries'. Amjad was previously Director of eBusiness and Distributed Systems Research at Bellcore (now known as Telcordia Technologies) for 8 years and a Professor of Information and Communication Systems at the Fordham Schools of Business for 5 years. He is currently an Adjunct Professor of Electrical and Systems Engineering at the University of Pennsylvania and CEO of NGE Solutions - a company that specializes in business and technical aspects of next generation enterprises (NGEs). He is currently leading research in IT planning, integration, security and administration (PISA) of NGEs. He is a member of ACM and IEEE Computer Society. He can be reached at umar@amjadumar.com.

Adalberto Zordan received an M.S. in Computer Science from the University of Padua, Italy. He is an independent consultant in enterprise systems, with more than ten years experience, focusing on organization reengineering and information systems. He was previously a consultant for four years for Bellcore (now known as Telcordia Technologies) where he worked on enterprise architecture and integration projects. He is currently a Contract Professor at the University of Padua where he teaches an advanced course on software engineering, with concentration on business modeling, enterprise software integration and migration, and service oriented software engineering. He is collaborating with NGE Solutions on the PISA project and is especially focusing on the design and development of the PISA automated framework for business modeling and software selection and evaluation. He can be reached at adalberto@adal.it.