Enterprise Information Systems Architecture—Analysis and Evaluation

Nan Niu, *Member, IEEE*, Li Da Xu, *Senior Member, IEEE*, and Zhuming Bi, *Senior Member, IEEE*

***Abstract*—Numerous software architecture proposals are available to industrial information engineers in developing their enterprise information systems. While those proposals and corresponding methodologies are helpful to engineers in determining appropriate architecture, the systematic methods for the evaluation of software architecture are scarce. To select appropriate software architecture from various alternatives appropriately, a scenario-based method has been proposed to assess how software architecture affects the fulfillment of business requirements. The empirical evaluation on the selection of a supply chain software tool has shown that the developed method offers remarkable insights of software development and can be incorporated into the industrial informatics practice of an organization with a moderate cost.**

***Index Terms*—Enterprise information systems, industrial informatics, software architecture, scenario-based method, system evaluation.**

I. Introduction

INFORMATION TECHNOLOGY (IT) plays a dominant role in today's industrial automation. For example, in designing a controlled drive system, an engineer working in the 1980s would deal primarily with mechanical and electronic components, whereas 90% of today's engineering time is devoted to the tasks on information systems [1]. Enterprise Information Systems (EISs) are the key IT assets for industrial enterprises to organize, plan, schedule, and control their business processes [2]. In particular, for supply chain management, EISs have become critical enablers for modern enterprises to streamline processes and achieve effectiveness, efficiency, competency, and competitiveness of the material flow. An essential component of an EIS is *software architecture*. Software architecture describes a set of system components as well as their topological relations in an EIS [3]. An advantage of architecture analysis lies in the early decisions about a software system's high level design [4]. Due to the importance of software architecture, the study in this field is emerging.

Researchers have recently proposed many software architecture descriptions to accelerate industrial applications [5]. These descriptions can be classified into domain-specific EISs [6], distributed real-time control [7], [8], embedded and dependable systems [9], [10], agent platforms [11], [12], and service-oriented architecture [13], [14]. In designing and implementing an EIS, software architecture must support the key business drivers; these drivers are also referred to as *quality attributes* or *non-functional requirements* (NFRs). Such a support is aligned with the enterprise missions and adds value to the system level [3]. As a proof-of-concept, existing architecture only supports single NFR, e.g., extensibility [8], fault tolerance [9] and so on.

When selecting software architecture for an EIS, industrial engineers need to consider multiple and often conflict NFRs. For example, a system's flexibility and real-time performance are conflicted with each other and must be balanced in software development [12]. Despite the increasing number of the proposals of software architecture options, fewer methods are available to evaluate software architecture against the requirements of a specific application. This has caused a hurdle in developing an EIS since the objectives of software architecture must be simultaneously considered to meet the requirements of today's IT-driven industrial automation. To select software architecture, a user-oriented method has been proposed to evaluate software architecture choices [15]; key NFRs are reviewed and the quality attribute scenarios are leveraged to assess the degree to which software architecture choices have influenced the fulfillment of the NFRs.

The reported work has been motivated to assist and support engineers in understanding the strengths and weaknesses of software architecture. This understanding will guide the selection of an EIS solution to meet business purposes. A new evaluation method is proposed and validated via a case study, and the result has shown that the evaluation on software architecture can offer concrete insights into EIS design and evolution [4]. The rest of the paper is organized as follows. Section II lays the background of our research based on the literature review, the classifications and identified limitations of existing methodologies. Section III details our scenario-based assessment approach for software architecture. Section IV presents an application of our approach and discusses the empirical study results for the verification. Section V summaries our works with the conclusions and the directions of our future work.

Manuscript received November 08, 2012; revised December 01, 2012; accepted December 21, 2012. Date of publication January 11, 2013; date of current version October 14, 2013. Paper no TII-12-0762.

N. Niu is with the Department of Computer Science, Mississippi State University, MS 39762 USA (e-mail: niu@cse.msstate.edu).

L. D. Xu is with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, China, and also with the Shanghai Jiao Tong University, Shanghai 200052, China, and also with the Department of Information Technology and Decision Sciences, Old Dominion University, Norfolk, VA 23529 USA (e-mail: lxu@odu.edu).

Z. Bi is with the Department of Engineering, Indiana University Purdue University Fort Wayne, Fort Wayne, IN 46805 USA (e-mail: biz@ipfw.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2013.2238948

II. Classifications of Existing Research Efforts

EISs have been emerged as vital tools with modern computing technologies to support the business processes at both of intra- and inter-organizational levels [2]. For examples, the applications of EISs in aerospace engineering have recently been explored [16], [17]. In this section, we review the key driving factors behind EISs and discuss how they have been supported by the research community.

A. Classification of NFRs

In software development, *functional requirements* describe what the system can do and *non-functional requirements* (NFRs) describe how well the system can be to fulfill required functions. NFRs, such as usability and reliability, represent the consideration of subjective performances at the system level. NFRs assist decision-making in selecting software tools in the hierarchical structure of an EIS [18].

The implementation of an EIS is usually focused on a few of major NFRs. However, the ill-consideration of driving NFRs in selecting suitable software architecture could be the most fatal mistake, which is very hard to be fixed at the late stage of the system implementation [18]. To uncover the key NFRs, a set of commercial software tools and the relevant publications on EISs [2], [7]–[13], [19] are selected and compared in Table I. The source references of each identified NFR have been provided. The 1st column lists what we believe the most descriptive quality attributes an EIS should possess. It is encouraging to note that, even with a limited number of the reviewed publications, a diversified set of recurring NFRs have been explored by the researchers, and various terminologies are employed to describe similar meanings. Therefore, concepts semantically close to the key NFRs are given in the 2nd column. Note that the exact meanings of a terminology are not necessary the same (e.g., "flexibility" in [7] and [12] have different emphases). Each NFR's goal is expanded by the topics in the 3rd column. Table I has helped to reveal the contemporary informatics concerns. Interested readers might look into the detailed discussion on EIS requirements [2], [11]. A new perspective provided in Table I is the distinction between business- and software-driven NFRs. The business drivers at one end of the spectrum enable an enterprise to organize and promote its businesses; such capabilities are viewed as a superior advantage in contrast to the continuous evolution of traditional information systems [2]. The NFRs at the other end of the range reflect the guiding principles that drive the software architecture design.

TABLE I
Key NFRs for EISs

NFR	Related Concepts	Topics	
Integration [1,11, 13, 21]	Interoperability [1] Coordination [1] Synchronization [11]	Linking and coordinating business processes over systems	Business-Driven
Extensibility [13]	Scalability [6,11]	Adding new functionalities with ease	
Customer-oriented [13]	Customization [22] Intelligence [1, 21] Flexibility [13]	Aligning a company's businesses with customers' needs	
Performance [14]	Efficiency [1 14] Real-time [8, 11] Schedulability [8] Memory usage [21]	Optimizing system performance under multiple constraints	
Agility [10]	Adaptability [23, 24] Flexibility [20] Autonomy [10, 25]	Responding change and uncertainties rapidly	
Reliability [14]	Robustness [4] Accountacy [9] Fault handling [14]	Operating a system to resist system or product failure	Software-Driven
Security [13]	Safety [9], [11] Info. protection [14]	Being free from danger or threat	
Reusability [1, 11, 25, 27]	Reconstructability [1]	Being reusable for new creations	
Testability [13]	Reviewability [9]	Verifying and validating a software artifact	
Usability [24]	Simplicity [1]	Being useful and usable	
Modularity [21]	Recomposability [14] Reconfigurability [13]	Making systems decomposable and reusable	

B. Types of Software Architecture

Software architecture describes system components as well as their external properties, and the internal relations of components [3]. This research field emerged in the 1990s when the major work was to establish the fundamentals of software architecture including *description languages*, *formal logic*, *architectural styles*, *design patterns*, and the like. In the context of industrial informatics, software architecture represents the business structures and processes of an EIS, and it is a vital tool to support the assessment of design operations at an early stage [2].

A significant contribution to the development of software architecture is the patterns codification which can be used as the blueprint of components, constraints, and their relations. Patterns define the general solutions that can be reused to accelerate the software development process. Some methods to apply the operational patterns for the EIS design are as follows.

- *General purpose software packages* encapsulate data structures and algorithms to implement a generic but customizable solution of business problems based on the best practices. Market-leading providers include SAP AG, Oracle Corporation, and Baan Co. In these packages, many operational patterns are exploited: database-centered data sharing, pipeline-based data processing, event driven message invocation, to name a few. The packaged software tools have been adopted by a variety of enterprises to optimize their business processes [2].
- *Domain-specific software architecture* is tailored to EISs in a specified domain. Such architecture includes some special components which differ from common components of generic software architecture. For example, an industry-oriented ERP is capable of accommodating the requirements especially for a certain industry domain; some insignificant software elements and tools included in generic software package can be removed to reduce the complexity [6]. Major enabling technologies for DSSA domain-specific software architecture include Enterprise Java Beans (EJB), Microsoft's Component Object Model (COM+), and business component factory [20].
- *Distributed computing* involves several interacting elements coordinated to achieve a system-level goal. Distributed programming typically falls into one of the following architectural options: *client-server*, *n-tier architecture*, and *peer-to-peer*. For the application of distributed computing, Ferrolho and Crisóstomo [7] presented distributed architecture to develop a flexible manufacturing

cell using the Ethernet network. In addition, programming languages with parallel and concurrency supports (e.g., C++) and middleware technologies (e.g., CORBA) are among the key enablers for distributed computing.

- *Agent and multi-agent systems (MAS)* have received much attention recently and they have been deployed widely. *An agent* is an autonomous entity situated in the environment; whereas a MAS is composed of a group of agents; the agents within a MAS can cooperate or compete each other to achieve the goals at the system level [21]. MASs have been successfully applied in manufacturing [11], behavior scheduling [12], workflow management [22], and business rules integration [23]. For example, Metzger and Polaków have investigated the applications of MSA in the process control [19].
- *Service-oriented architecture (SOA)* can be viewed as a recent advance in integrating heterogeneous platforms including legacy software tools [2]. A SOA allows an EIS to extend its capabilities by applying reusable software modules so that the development cost can be reduced without reinventing a wheel [24]. Equipped with methods like Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Discovery, Description, and Integration (UDDI), the SOA has been introduced as a critical enabling technology to EISs [13], [14].

In summary, industrial information engineers have begun to leverage the central ideas of software architecture—abstraction and separation of functions—to tackle the complexity of EIS development. Some combined approaches are also proposed; for example, Zhang and Jiang incorporated software agents into the SOA for the coordination and interactions in complex systems [22]. As the number of EIS solutions keeps increasing, it becomes important to systematically evaluate software architecture in designing and implementing an EIS.

III. Scenario-Based Software Architecture Analysis

The user-oriented method is proposed to facilitate the decision-makings related to EIS software architecture. The evaluation in the proposed method, according to Section II-A, shall be performed based on the intended business and quality attribute goals. This is because the way software architecture supports the driving NFRs determines how an EIS will behave. The performance of an EIS in turn will shape the business strategies and technical capabilities of enterprises.

The benefits of fulfilling the EIS NFRs are unarguable [10]; however, there are many unsolved practical issues when NFRs are considered in the implementation of EISs. For example, NFRs are usually subjective and hard to be quantified. This calls for qualitative methods to reason how well the EIS software can meet the NFRs [18]. Moreover, users express their missions with different terminologies [26], even though there are several standards related to NFRs, e.g., the International Organization for Standardization and International Electro-technical Commission (ISO/IEC) 25030. Terminological interferences relevant to the EISs are unavoidable, as illustrated in the first and second columns of Table I. Another challenge is that an EIS has to balance a set of the conflict objectives to determine its

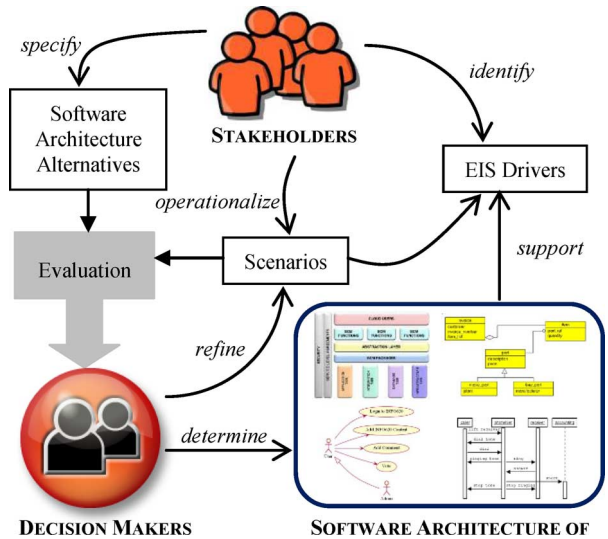


Fig. 1. Framework for scenario-based EIS software architecture analysis.

software architecture. Some examples of the conflict objectives are *flexibility* vs. *productivity*, *scalability* vs. *reliability*; moreover, all of the objectives contribute to the cost factor. It is necessary for industrial information engineers to consider all of the objectives simultaneously at the system level.

To meet practical challenges in evaluating software architecture based on the given NFRs, we propose a scenario-based method as shown in Fig. 1, where boxes and arrows represent entities and activities respectively. The core component—"evaluation"—is highlighted in a shaded callout. As depicted in Fig. 1, the scenarios play two important roles in the evaluation: firstly, they allow the abstract NFRs to be concretely defined, operationally measured, and meaningfully communicated among the stakeholders; secondly, they link architecture choices to the satisfaction of the EIS drivers, which helps the management to make an informed decision about the system that is best suited to their needs. The rest of this section describes key activities of the proposed method in details.

A. Identifying NFRs

Without losing the generality, an EIS for an academic department is used as an example in this section; it is called a research administration system (RAS) in the rest of paper. The purpose of the RAS is to manage the research expenses for a group of researchers. The participants in the RAS can be easily extended to multi-institutions even multi-countries. In the proposed RAS, participators are able to register and take part in project events, upload or download project reports, and claim costs for the tasks. The project activities are planned and the progress is updated periodically.

RAS is a typical EIS, which is capable of maintaining data and providing the interfaces to users to access, transfer, process, and report data related to the research projects. Meanwhile, the fulfillment of several NFRs exhibited by the system will determine the success of RAS. Despite that many NFRs have been listed and discussed in Table I; when a specific application of EIS is considered, a sub-set of NFRs can be selected based on the priorities of the goal in the application. In other words, a

NFR might be crucial to one application but insignificant to another application. While all of the NFRs should be considered, an industrial information engineer should select a few of major NFRs in applying the proposed method to simplify the evaluation. The following NFRs are considered as major NFRs in the example EIS.

- **Performance**. RAS deals with transactions varying in duration and complexity. Without good performance, RAS could lead to unsatisfactory services or even corporate loss. As indicated in Table I, the performance can be refined along the time and space dimensions.
  - — **Response Time**. Speeding information processing will shorten the duration of transactions, thereby improving the performance of RAS.
  - — **Memory Usage**. The performance can be affected if the RAS uses the internal computing memory extensively. Making efficient use of external computer storage, on the other hand, enhances system performance.
- **Integrity**. RAS must ensure accuracy and consistency of the data to maintain the business standards. Note that business standards are the necessary authorities and routines in the business practice. They are used to ensure that the business organizations regulate the business processes.
- **Persistence**. RAS is required to store and retrieve the state as data in non-volatile storage. Persistence refers to the characteristic of state that outlives the process that created it. Such a characteristic must be accounted for as an architectural decision.

While other NFRs, such as security factor, may be necessary to be considered for an RAS, the above list of NFRs is sufficient for an illustrative purpose. However, the stakeholders need to elicit the key NFR drivers for their particular EIS instead of adopting existing NFR standards without any modification [27]. To identify a driving NFR, one might justify whether or not the change of this NFR will have an impact on the whole software architecture [3].

B. Eliciting Scenarios

In this section, a *scenario* refers to the quality attribute scenario. A scenario is an abstracted description of system to be designed; both the user and designer's perspectives have to be considered. Scenarios play an important role in requirements elicitation and analysis.

Scenarios are frequently used in the system development process, e.g., use case scenarios are a part of the rational unified process and the primary sources of the definition of requirements in the agile software development.

When dealing with subjective concepts like NFRs, scenarios can be used to evaluate if a set of subjective attributes can be satisfied by software architecture. For this reason, the quality attributes scenarios are created to evaluate the interactions of system from the perspectives of the subjective attributes [15]. In contrast to the terminology scenarios used by others, the scenario here must relate to subjective attributes.

In other words, NFRs have to be defined clearly in a scenario. For example the statement "a system is flexible' is invalid since it is vague and meaningless. All systems are flexible to accommodate a certain type of changes [28]. On the other hand, the following statement will be valid for a scenario:

> *"A user expects to insert an editable field for searching and add an active in the graphical user interface; the icons in the toolbar must be scaled, and the changes should be completed within 3 hours; these changes address the issues 4 and 12 raised in the bug report so that usability will be improved."*

The scenarios make NFRs measurable and also help resolve terminological ambiguities by capturing the stakeholders' precise concerns. For an RAS, we devise the following scenarios.

- **Sce**1: To meet the system requirements, RAS developers take into account the organizational workload. Some transactions (e.g., Submit_Expense_Report) are invoked more frequently than others (e.g., Foreign_Exchange). Caching frequently occurring and business-critical transactions will improve the response time and persistence, though the memory usage is likely to experience extra overhead.
- **Sce**2: A temporal constraint of RAS is that a participator is not allowed to attend a meeting unless he or she registers it. RAS administrators want efficient enforcement of integrity constraints so that unsatisfied business rules can be detected, monitored, and eventually corrected. However, searching and checking every constraint at every point in time can negatively affect system performance.
- **Sce**3: When planning a research-related business trip, the users would like RAS to interact with a variety of external services like flight, hotel, and car rental so that the main memory usage can be reduced. Not only shall RAS ensure local integrity constraints, but the complex cross-organizational constraints such as "the travel expense must be less than the maximized amount specified for each research project" must also be enforced.

There are other factors be considered. For example, a scenario should associate the tasks with the roles of participators [15]. In this way, system usages can be evaluated from multiple perspectives: software developers in **Sce**1, system administrators in **Sce**2, and end users in **Sce**3. The other factor is the use of templates; although a few of templates have been introduced [3], the proposed scenario used general descriptions instead of formatted structure. In Table II, the relationship of the scenarios and the NFRs is specified on a qualitative rating scale. The check mark and cross mark have shown the positive and negative contributions of the scenario to the NFR, respectively, and the field with 'N/A' has shown that the NFR is not explicitly considered in the scenario [18].

C. Evaluating of System Architecture

Tradeoff on the conflict objectives must be made in an engineering situation that involves competing contingencies. Tradeoff on software architecture is about how to make the decisions with a full comprehension of both the upside and downside of a particular choice. We examine two software architecture alternatives for RAS.

The first is database-centered architecture presented in Fig. 2, where the design considerations are arranged in a grid. The horizontal axis depicts the types of *conceptual* or *ontological* fea-

TABLE II
CONTRIBUTION RELATION BETWEEN SCENARIOS AND NFRS

	Performance		Integrity	Persistence
	Response Time	Memory Usage		
Sce_1	✓	✗	N/A	✓
Sce_2	✗	N/A	✓	
Sce_3	N/A	✓	✗	✗

Legend:
'✓': means the scenario contributes positively to the NFR,
'✗': means the scenario contributes negatively to the NFR, and
'N/A': means the scenario contributes has nothing to do with the NFR

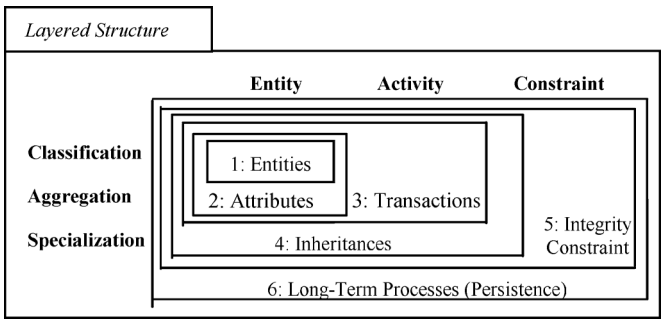


Fig. 2. Database-centered layered architecture.

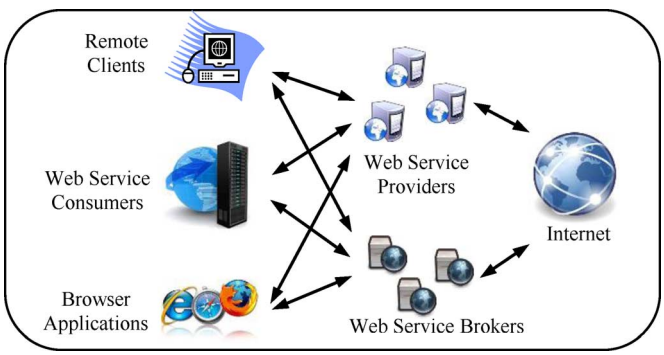


Fig. 3. Web services architecture.

tures involved in an RAS: *Entity*, *Activity*, and *Constraint*. The vertical axis describes the features of the *organizations in a* semantic model: *Classification*, *Aggregation*, and *Specialization*. The other narrative contents in the plot have shown how the objectives of RAS are translated and related to the system components. The architecture, as shown in Fig. 2, is a layered structure where the information flow occurs only in two adjacent layers. This modular property of information hiding, together with the centralized data model, makes the database-centered architecture a suitable support for the caching organizational workload requirement in **Sce**1 and the checking local integrity constraints requirement in **Sce**2.

The second option of software architecture for RAS, as illustrated in Fig. 3, is a distributed service-oriented infrastructure that exploits the Web services. In a nutshell, the interactions among the machines are supported by a web service, which is WSDL-based machine-processable interface. As a result, the Web services architecture is particularly suited for supporting the remote communication use case described in **Sce**3. At the same time, the caching need stated in **Sce**1 can be supported by client-side applications.

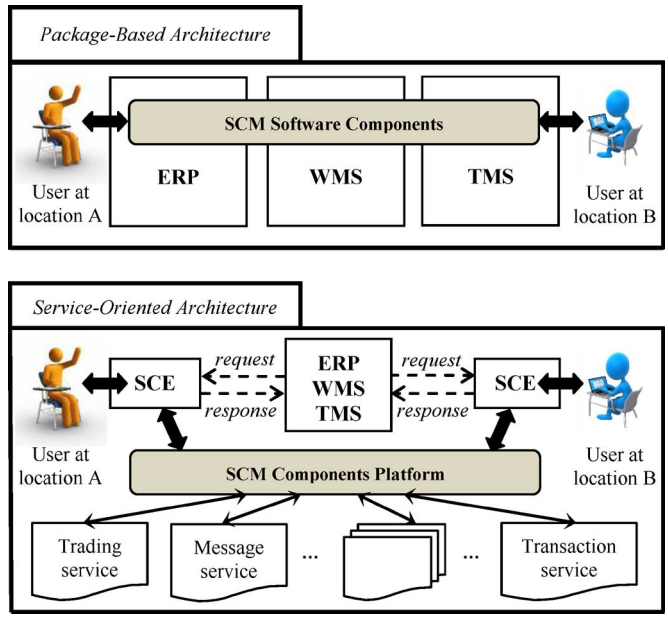


Fig. 4. BXD's SCM software architecture alternatives.

Having teased out what scenarios are explicitly supported by each design alternative, software architecture options can be evaluated. The main idea is to use the scenarios to connect software architecture with the driving NFRs and to propagate the contribution relations through qualitative design reasoning. For reasoning, the evaluation is focused on finding a solution that is sufficiently good rather than fully optimal in all aspects. The underlying rationale of reasoning in such a matter is that the satisfaction of NFRs is not a straight true or false answer [18]. Take Table II as an example, no matter which scenario or combination of scenarios are supported, there is always one or more NFRs that are negatively affected. In another word, the optimal architecture that meets all of the RAS stakeholders' needs is simply nonexistent.

Effective design analysis under the circumstances like the design of an RAS, in our opinion, depends on the ability to aggregate and present the relevant information in an insightful way to help decision makers to find the right balance among the NFRs. We use an intuitive and integrated graphical representation to serve this purpose.

D. Documentation of Architectural Decisions

The integrated view of Fig. 4 offers not only a novel synergy of quality attribute information [17] and label propagation [18], but also a valuable input to the architectural decision-making process. For decision makers, recommending a top-down method is recommended to prioritize the NFRs firstly and then determine an architecture solution that meets the priorities the best. In Fig. 4, if **Integrity** is a key factor that determines the success of RAS, then **Sce**2 needs to be supported and thus the database-centric architecture can be selected. Similarly, RAS can be developed by using Web services if **Memory Usage** is a primary concern.

It is worth to mention that the process of the proposed evaluation is interactive and human can be involved at any stages. For example, although the label propagation can be partially or even

fully automated [18], designers can offer invaluable guidance in the process; the interactions can be made to override established labels. However, no override should be performed unless the rationales of change are properly documented. In this sense, the composed contribution graph shown in Fig. 4 serves as one of the views for documenting software architecture. Other commonly employed views include components and connectors, use cases, and structural and behavioral diagrams [3]. The bottom-right component of Fig. 1 has shown the advantages of multi-views for the representation of software architecture.

It cannot be overemphasized that an architecture solution is determined only if the design tradeoffs are thoroughly appreciated. In cases that informed timing decision cannot be reached, an incremental evaluation is necessary. The "refine" arrow of Fig. 1 demonstrates such an iterative process. In the case of RAS, it can be inferred from Table II that Memory Usage and Integrity seems to be mutually conflict NFRs—any scenario that helps to achieve one requirement will fail to achieve the other. If this is the case, the priority must not be given to both NFRs; otherwise, more scenarios could be elicited to establish a compromise between them.

IV. Empirical Evaluation

We applied the proposed method to an EIS to illustrate its application. How to make tradeoffs in determining software architecture has been discussed to application, and how to make tradeoffs in determining software architecture in this section.

A. Background

The empirical evaluation is for the selection of system architecture for a supply chain management (SCM) software used in the BXD Corporation (a fictitious name); it is a leading electronics provider in the southeast of the United States. The main activities of the BXD's supply chain are involved in several stages including customer orders, replenishments, manufacturing, and procurements. Over the years, the responsible department for each stage focused on its own operation efficiency, e.g., BXD retail department improved its inventory ownership, delivery modes, order quantity, size of workforce, and service sequence. However, the lack of integration and coordination became a serious bottleneck in the transactions between two stags. Since the late 1990s, BXD has realized the importance of integrated planning, scheduling, and controlling the supply chain via a well-defined infrastructure.

Accordingly, software architecture has enabled BXD to increase the synergy of cross-functional business integration. BXD currently adopts a package-based architectural style for its SCM software, as shown in the top of Fig. 4. The SCM software components run through ERP, warehouse management systems (WMS), and transportation management system (TMS) via the electronic data interchange (EDI) interfaces. The ERP package has contributed to the SCM by fulfilling two main functions: 1) a transaction processing engine that allows the integrated management of data over the enterprise, and 2) a workflow management that controls numerous process flows, such as the order-to-cash and purchasing processes. The WMS basically controls the storage and the movement of the BXD products and raw materials, including shipping, receiving, and holdings. The WMS also helps maintain the key customer ordering pattern and the status of the bin utilization, especially when BXD's material flow is not uniform. The TMS, a third major component in BXD's package-based architecture, organizes the freight consolidation operations and coordinates company shipments. The key TMS functions are to plan terrestrial rounds, manage air and maritime supports, simulate transport scheme and costs, and track shipment batch records.

The BXD deployed the packaged-based architecture in the early 2000s as one of the "best practices" at that time. Having being evolved for nearly a decade to address the company's changing requirements, the SCM software became a legacy system that exhibited some deficiencies in satisfying business and quality attribute drivers, especially in terms of integrating distributed, heterogeneous, and cross-organizational functions.

For this reason, the BXD has explored the possibility of changing its SCM to a service-oriented architecture (SOA) sketched at the bottom of Fig. 4. Compared to the traditional package based architecture, the SOA has incorporated a set of supply chain engines (SCE) [14] that allow efficient handling of distributed users. Another distinction is that the previously tangled services are separated in the SOA due to the modularized architecture.

B. Verification and Results

We have collaborated with the BXD in evaluating the two architecture alternatives shown in Fig. 4. BXD provided us with the required documents such as the specifications of software systems, software descriptions, and definitions of the business missions. Studying these materials helped us understand the context of BXD's SCM.

Our main analysis was then performed with a group of six domain experts. In the analysis, we presented a list of NFRs from Table I to tailor in the needs of the company. BXD experts judged the relevance of the NFRs to their business and also provided additional NFRs that they thought necessary to be considered. As a result, 15 NFRs were viewed as important ones to the success of BXD. This included both business drivers (e.g., integration and performance) and software drivers (e.g., modifiability and testability). Nearly six scenarios were selected for each role of the BXD stakeholders.

Each scenario was applied in up to 4 NFRs, and it was described less than five sentences. The architecture alternatives were analyzed in a top-down way, as recommended in Section III-D, based on the criterion of how well they supported the prioritized NFRs. The assessment was iterated for several rounds to resolve ambiguities and consolidate expert opinions.

The conducted workshop generated many insights that BXD regarded valuable. Of particular interests were the insufficiently addressed NFRs. Our preliminary study has suggested that BXD's SCM software would exhibit the quality attributes in a sub-optimal way. The problems were further classified into three categories: 1) *Omission* refers to the risks caused by unfinished activities, e.g., risks from missing an operational definition for a flexible end-to-end procurement or logistic business process; 2) *Commission* refers to the risks caused by a suspicious decision related to software architecture, e.g., the risks from a wrong selection of operating platform which is

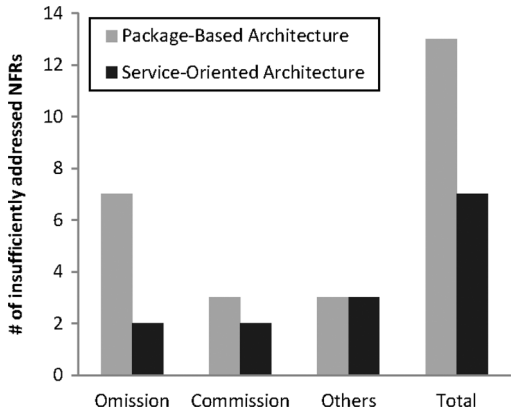


Fig. 5. Classification of insufficiently addressed NFRs in case study.

a failure to support a heterogeneous computing environment; and 3) *Others* refer to the risks that are neither omissions nor commissions, e.g., the system maintenance cost rate increases as the system meets the needs of diversified user communities for the purposes of integration, testing, and training. In this way, the above mentioned flexibility, performance, and cost NFRs are deemed as inadequately addressed and are classified as omission, commission, and others respectively.

Fig. 5 has shown the distribution of the inadequately addressed NFRs uncovered in the workshop. Note that the tradeoff analysis is applied to the package-based architecture in a retrospective manner as it is in operation for a period of time already. In contrast, the tradeoffs of the SOA are assessed in a proactive way. Among a total of 15 NFRs, 13 NFRs experience certain levels of deficiencies in the package-based architecture, whereas this number reduces to 7 in SOA. In other words, our method has effectively helped BXD stakeholders seek the answers for the right questions and avoid potential mistakes in determining software architecture. A synthesis of 18 architecture evaluations observes that the omissions are twice as many as the commissions [29]. While this is roughly the case for package-based architecture (7 omissions and 3 commissions), it is surprising to realize that the SOA analysis results in an equal number of omissions and commissions—2 occurrences of each risk type. This may suggest that our scenario-based method has overcome the limitation of architecture analysis by revealing a significant number of omission errors; however, such a speculation warrants further investigation.

To shed light on handling EIS architecture tradeoffs in practice, important observations from the empirical evaluation are summarized as follows. We found the effort involved in carrying out our method was moderate and the process was straightforward so that it could be integrated into the implementation of other IT projects. From our experience, the creation of the scenarios resembles to software testing: it is hard to tell how many tests are sufficient; however, it is practical to determine a step from which the improvement of the credibility of testing results can be negligible. Another important factor is to reduce the required resources; with the helps from the experts, we devised about six scenarios for each stakeholder role in only 30 minutes, which turned out to be adequate for our analysis.

We realized that including a wide variety of stakeholder roles in conducting architecture tradeoff analysis is extremely important in sense that a comprehensive set of important and conflict goals could be balanced. The business-driven NFRs listed in Table I tend to be mentioned and emphasized more than the software driven ones during the BXD workshop. This imbalance toward business drivers is understandable among industrial informatics practitioners; but a more balanced view would allow the tradeoffs to be reasoned more thoroughly. In this sense, Table I or other codified NFR catalogs like [18] can be of great practical value. In fact, devising scenarios to make NFRs measurable and analyzing software architecture tradeoffs were not confined to the determination of software architecture. Our partner BXD believes that the proposed method can be extended and applied to the phase of system operations to predict potential problems in product life cycle at a minimized cost.

V. Summary and Future Work

A large group of industrial information engineers have focused on developing software tools for EISs, since factory automations are now driven by information technologies. Despite the emerging trend that more informatics is carried out at the early stage of product development, the evolution of EISs has not caught up with this trend. Specifically, the research on software architecture for EISs remains inadequate to deal with today's IT-driven industrial automation. Thus, there is a critical need to evaluate software architecture and select most appropriate one to fulfill the business requirements, in particular, NFRs. In this paper, we have identified the challenges confronting the EIS software architecture development, proposed a scenario-based method for the tradeoff analysis of software architecture, and conducted an empirical study to verify the method. The study on a manufacturing company's supply chain software has demonstrated the applicability and usefulness of the new method. On one hand, only moderate effort is needed to analyze and evaluate software architecture. On the other hand, the discovered architecture tradeoffs provide valuable and practical insights into EIS design and evolution.

There are several dimensions in which our work can be expanded in future. Firstly, the level of the details of the reported empirical study should be increased to enhance the strengths of the developed method. Secondly, identifying and codifying industrial-oriented NFR catalogs and analysis patterns [6] should be in order. Thirdly, in terms of evaluating the EIS architecture alternatives, it is worth comparing a proposed scenario-based method with other holistic approaches [30]. Finally, we would like to combine our approach with the recent thread on formal methods in informatics [31] to address cutting-edge research issues in EISs.

References

[1] O. Kaynak, "The exhilarating journey from industrial electronics to industrial informatics," *IEEE Trans. Ind. Informat.*, vol. 1, p. 73, May 2005.

[2] L. D. Xu, "Enterprise systems: State-of-the-art and future trends," *IEEE Trans. Ind. Informat.*, vol. 7, pp. 630–640, Nov. 2011.

[3] L. Blass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Norwell, MA, USA: Addison-Wesley, 2003.

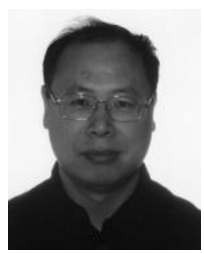
[4] G. A. Fodor, "Industrial informatics: predicting with abstractions," *IEEE Trans. Ind. Informat.*, vol. 1, p. 3, Feb. 2005.
[5] Z. D. Zhou, R. Valerdi, and S.-M. Zhou, "Guest editorial special section on enterprise systems," *IEEE Trans. Ind. Informat.*, vol. 8, no. 3, pp. 630–630, 2012.
[6] S. Wu, L. Xu, and W. He, "Industry-oriented enterprise resource planning," *Enterprise Inf. Syst.*, vol. 3, pp. 409–424, May 2002.
[7] A. Ferrolho and M. Crisóstomo, "Intelligent control and integration software for flexible manufacturing cells," *IEEE Trans. Ind. Informat.*, vol. 3, pp. 3–11, Feb. 2007.
[8] Q. Zhu, Y. Yang, M. Di Natale, E. Scholte, and A. Sangiovanni-Vincentelli, "Optimizing the software architecture for extensibility in hard real-time distributed systems," *IEEE Trans. Ind. Informat.*, vol. 3, pp. 621–636, Nov. 2010.
[9] F. Salewski and S. Kowalewski, "Hardware/software design considerations for automotive embedded systems," *IEEE Trans. Ind. Informat.*, vol. 5, pp. 156–163, Aug. 2008.
[10] D. Cancila, R. Passerone, T. Vardanega, and M. Panunzio, "Toward correctness in the specification and handling of non-functional attributes of high-integrity real-time embedded systems," *IEEE Trans. Ind. Informat.*, vol. 6, pp. 181–194, May 2010.
[11] M. Ulieru and M. Cobzaru, "Building holonic supply chain management systems: An e-logistics application for the telephone manufacturing industry," *IEEE Trans. Ind. Informat.*, vol. 1, pp. 18–30, Feb. 2005.
[12] S. Theiss, V. Vasyutynskyy, and K. Kabitzsch, "Software agents in industry: A customized framework in theory and praxis," *IEEE Trans. Ind. Informat.*, vol. 5, pp. 147–156, May 2009.
[13] S. Runde and A. Fay, "Software support for building automation requirements engineering—An application of semantic web technologies in automation," *IEEE Trans. Ind. Informat.*, vol. 7, pp. 723–730, Nov. 2011.
[14] G. Cândido, A. W. Colombo, J. Barata, and F. Jammes, "Service-oriented infrastructure to support the deployment of evolvable production systems," *IEEE Trans. Ind. Informat.*, vol. 7, pp. 759–767, Nov. 2011.
[15] I. Ozkaya, L. Bass, R. L. Nord, and R. S. Sangwan, "Making practical use of quality attribute information," *IEEE Software*, vol. 25, no. 2, pp. 25–33, Mar./Apr. 2008.
[16] Y. H. Yin, J. Y. Xie, L. D. Xiu, and H. Chen, "Imaginal thinking-based human-machine design methodology for the configuration of reconfigurable machine tools," *IEEE Trans. Ind. Informat.*, vol. 8, no. 3, pp. 659–6668, 2012.
[17] L. D. Xu, C. Wang, Z. Bi, and J. Yu, "AutoAssem: An automated assembly planning system for complex products," *IEEE Trans. Ind. Informat.*, vol. 8, no. 3, pp. 630–630, 2012.
[18] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Norwell, MA, USA: Kluwer, 2000.
[19] M. Metzger and G. Polaków, "A survey on applications of agent technology in industrial process control," *IEEE Trans. Ind. Informat.*, vol. 7, pp. 570–581, Nov. 2011.
[20] P. Herzuma and O. Sims, *Business Component Factory: A Comprehensive Overview of Component-Based Development for Enterprise*. New York, NY, USA: Wiley, 2000.
[21] M. Wooldridge, *An Introduction to MultiAgent Systems*. Chichester, U.K.: Wiley, 2002.
[22] M. B. Blake, "Agent-based workflow configuration and management of on-line services," in *Proc. Int. Conf. Electronic Commerce Research (ICECR-4)*, Dallas, TX, USA, 2001, pp. 567–588.
[23] S. Ali, B. Soh, and T. Torabi, "A novel approach toward integration of rules into business processes using an agent-oriented framework," *IEEE Trans. Ind. Informat.*, vol. 2, pp. 145–154, Aug. 2006.
[24] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices*. Upper Saddle River, NJ, USA: Prentice-Hall PTR, 2004.
[25] T. I. Zhang and H. Jiang, "A framework of incorporating software agents into SOA," in *Proc. Artificial Intelligence Soft Computing (ASC 2005)*, Benidorm, Spain, 2005.
[26] N. Niu and S. Easterbrook, "So, you think you know others' goals? A repertory grid study," *IEEE Software*, vol. 24, no. 2, pp. 53–61, Mar./Apr. 2007.
[27] N. Niu, M. Jin, and J.-R. C. Cheng, "A case study of exploiting enterprise resource planning requirements," *Enterprise Inf. Syst.*, vol. 5, pp. 183–206, May 2011.
[28] N. Niu and S. Easterbrook, "Concept analysis for product line requirements," in *Proc. Int. Conf. Aspect-Oriented Software Development. (AOSD-8)*, Charlottesville, VA, 2009, pp. 137–148.
[29] L. Bass, R. L. Nord, W. Wood, and D. Zubrow, "Risk Themes Discovered Through Architecture Evaluations," CMU/SEI, Pittsburgh, PA, USA, Tech. Rep. TR-2006-012, Sep. 2006.
[30] M. C. Chou, H. Ye, X.-M. Yuan, Y. N. Cheng, L. Chua, Y. Guam, S. E. Lee, and Y. C. Tay, "Analysis of a software-focused products and service supply chain," *IEEE Trans. Ind. Informat.*, vol. 2, pp. 295–302, Nov. 2006.
[31] J. Campos, "Guest editorial special section on formal methods in manufacturing," *IEEE Trans. Ind. Informat.*, vol. 6, pp. 125–126, May 2010.

Nan Niu (M'09) received the B.Eng. degree from Beijing Institute of Technology, China, and the M.Sc. degree from the University of Alberta, AB, Canada, and the Ph.D. degree from the University of Toronto, ON, Canada, all in computer science.

He is an Assistant Professor in Computer Science and Engineering at Mississippi State University, MS, USA. His research interests include software engineering, requirements engineering, program comprehension, and industrial informatics.

Li Da Xu (M'86–SM'11) received the M.S. degree in information science and engineering from the University of Science and Technology of China, in 1981, and the Ph.D. degree in systems science and engineering from Portland State University, Portland, OR, USA, in 1986.

He serves as the Founding Chair of IFIP TC8 WG8.9 and the Founding Chair of the IEEE SMC Society Technical Committee on Enterprise Information Systems.



Zhuming Bi (M'11–SM'12) received Ph.D. degrees from Harbin Institute of Technology, China, and the University of Saskatchewan, Canada, in 1994 and 2002, respectively.

He is an Assistant Professor of Mechanical Engineering at Indiana University Purdue University Fort Wayne (IPFW), IN, USA. His current interests include mechatronics, automatic robotic processing, reconfigurable manufacturing and assembling systems.