An Architecture Model for Supporting Application Integration Decisions

Robert Winter
Institute of Information Management, University of St. Gallen
Mueller-Friedberg-Str 8, 9000 St. Gallen, Switzerland
E-mail Robert.Winter@unisg.ch
Phone +41 71 224 2934 Fax +41 71 224 2189

Abstract

For a successful information systems architecture management, reference models and decision rules must be provided. In this paper, an architecture model and design rules are proposed that are intended to support information systems architects in integration decisions. The goal is to improve application architecture by balancing intra-application integration costs and inter-application integration costs. Based on a general discussion of architecture models and integration costs, related work is reviewed, and a model is proposed that combines two existing models. Using that model, existing application design is analyzed, and an ideal generic application design is presented. In order to minimize inter-application integration costs, three approaches to optimal application interfacing are discussed. The work is based on experience from several integration projects at large Swiss and German companies, mainly from the service sector.

Keywords

Information systems architecture, Application architecture, Application integration, Enterprise architecture, Integration management

Introduction

Information systems (IS) architectures and architecture models are a central topic of IS research (King 1995). Since the late 1970ies, vendors / consultants (e.g. IBM 1984; Zachman 1987) as well as researchers (e.g. Krcmar 1990; Österle et al. 1992) contribute to this discussion. More recently, IS architecture has been linked to business architecture (McDavid 1999; Youngs et al. 1999), resulting in comprehensive architectural frameworks often designated as 'enterprise architecture' (Malhotra 1996; Zachman 1999).

While a high level of aggregation and a broad range of perspectives and aspects is common to most IS architecture model proposals (e.g. Zachman 1987, Scheer 2000), it is disputed whether design rules must be included (e.g. Youngs et al. 1999). We follow Sinz' (1997) definition of IS architecture that comprises not only models that can be used to represent IS components and their relationships, but also methods that support IS architects in creating appropriate architectures.

According to the ISA concept (Krcmar 1990), IS architecture comprises a strategy component, process architecture, organization architecture, application architecture, data architecture and communications architecture. Applications represent tightly related software artifacts (modules, data structures and / or components) on an aggregate level. Such tight

relations among software artifacts may result from common data usage, joint support of a certain business process, or common ownership.

In order to manage (re-)design linkages between strategy, processes, applications and software, many enterprise architecture frameworks propose a multi-level approach: (Winter 2003)

(1) Strategy (re-)design is intended to optimize the positioning of business units in an industry, in certain marketplaces or in a value network, thereby – among others – specifying appropriate products and / or services.

(2) Business process (re-)design is intended to optimize the organization and managerial control of producing the products and / or services identified in step (1).

(3) Application (re-)design is intended to optimize the integration of applications that support the business processes identified in step (2).

(4) Finally, software (re-)design is intended to optimize the reuse of software artifacts (components, modules, data structures) that implement applications identified in step (3).

The application level is differentiated from the software level as well as from the process level because the respective optimization goals are different. This paper is located on level (3), i.e. deals with models and methods for improving application integration.

Two forms of application integration can be differentiated:

- Application design: Software artifacts that are tightly interconnected, have to be integrated to form an application.
- Application interfacing: Efficient interfaces between applications have to be developed.

With regard to total integration costs, application design and application interfacing constitute a tradeoff: If many small applications result from application design, the respective intra-application costs are low due to the small average application complexity, while many interfaces have to be created that induce high inter-application integration costs. If, on the other hand, only few large applications result from application design, interfacing costs are much lower while intra-application integration is much more expensive due to higher application complexity. The tradeoff is illustrated by figure 1.

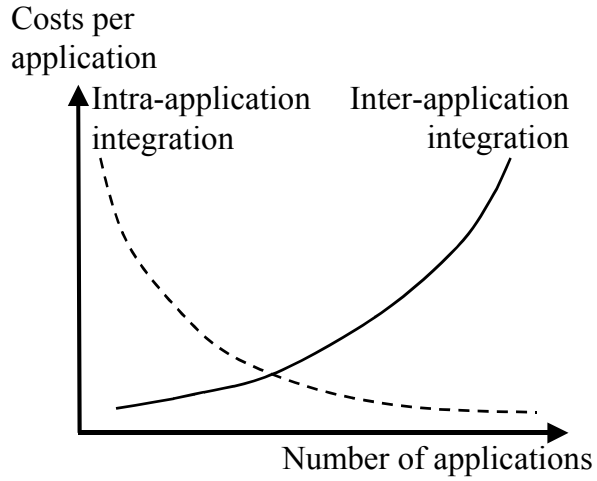


Figure 1. Integration cost tradeoff

An optimal integration level is achieved when total integration costs are minimized. This understanding of integration management is strongly linked to similar problem interpretations from organizational theory (e.g. Frese 1995).

The architecture model proposed in this paper is intended to support IS architects in making application design and application interfacing decisions that intend to reduce the total of intra-application integration costs and inter-application integration costs. To reflect the multi-level structure, process specifications and organizational structure are inputs for the model.

Based on a discussion of related work, a model for representing application architecture is derived in Section 2. Using that model, the analysis of existing application design and an ideal generic application design are presented in Section 3. In order to derive an ideal generic application architecture, optimal application interfacing is discussed in Section 4. Section 5 summarizes the results and suggests future extensions.

The contribution of this paper is as follows:

- Compared to existing models, the proposed architecture model includes additional aspects into the application design process. The proposed architecture model comprises explicit dimensions and scales so that integration metrics could be defined and used to formalize application architecture optimization in the future.
- In a first move towards formalized integration optimization, heuristics are proposed for application design as well as application interfacing.
- On an aggregate and conceptual level, it becomes apparent that different types of common interface layers provided by the software industry have similar effects on application architecture and therefore should be managed similarly.

Modeling application architecture

From the large number of published application architecture models, two have been selected because they provide a well-defined system of dimensions and scales to support the identification and representation of applications and interfaces, respectively. If the positioning of an application in an architecture model is more or less arbitrary due to missing explicit dimensions, missing arrangement rules, or fuzzy scales, both the proposal of reference architectures and the optimization of integration costs cannot be supported effectively. We discuss these two models and, since both models are based on important considerations, derive a combination model.

Related work

With its roots as early as in the 1960ies, IBM's (1984) Business Systems Planning method (BSP) has been widely used by companies to design application and model application architecture. Data usages of business processes are represented in data cluster / business process matrix. After being topologically sorted according to data flow sequence, high density areas of that matrix indicate a tight coupling of certain data and certain business processes (i.e. application candidates), while matrix areas of low usage density indicate efficient interfacing potentials between applications.

An alternative approach to application design (and application architecture modeling) has been proposed in IMG's (2000) Promet Systems and Technology Planning (STP) method. Again a two-dimensional matrix is used to identify areas of high density (= application

candidates) and areas of low density (= interface candidates). In contrast to BSP, STP's matrix dimensions are business units and functional clusters. The identification of application candidates is based on function ownership. Figure 2 illustrates the similarity of BSP's and STP's approach to application identification.

The CONTEXT architecture model (Penzel 2002; Spitzer 2002) provides three dimensions for application references to products, process steps (e.g. acquisition, ordering, execution, post calculation), and platforms (e.g. application, middleware, database, operating system, hardware, network). For every chosen value in one dimension, 'slices' (= two-dimensional models) can be derived that represent a business process (for a selected product), a platform (for a selected process step) or an infrastructure model (for a selected platform). Hence the CONTEXT architecture model allows to represent much richer context information for applications than two-dimensional models. However, rules for identifying 'building blocks' do not base on hard facts like data utilization or function ownership. Instead, functionalities should be clustered in a building block that are most similar with regard to relevant requirements, relevant know-how and interdependencies, but which also have the lowest interdependencies with other building blocks (Spitzer 2002). Building blocks should also represent industry best practices and should be economically efficient (Spitzer 2002).

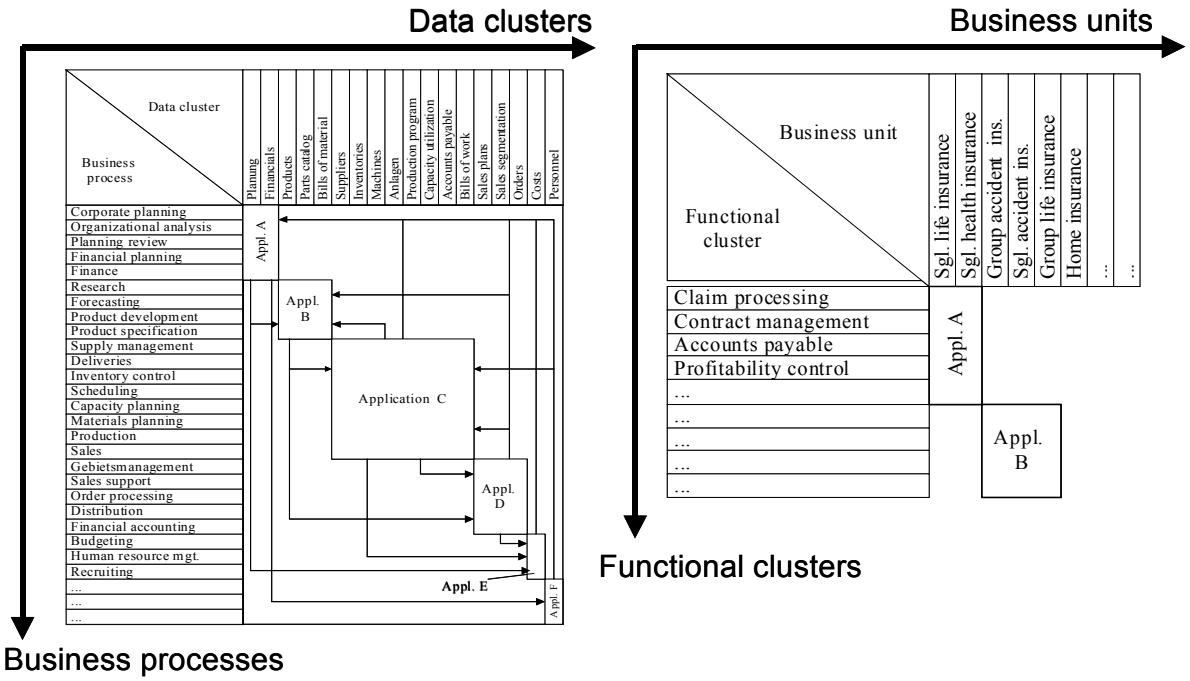


Figure 2. Application architecture model of BSP (left) and STP (right)

Derivation of a combination model

While BSP's identification of application candidates is based on data utilization, STP uses function ownership. CONTEXT's identification of 'building blocks' is more complex and includes 'soft' evaluations and assessments. Rules for application design should be as straightforward as possible. Therefore, the application building rules from BSP and STP are combined in the following. In general, the approach proposed in this paper would also allow the CONTEXT rules to be applied.

Data utilization as well as function ownership are important for integration optimization: While basing application design on data utilization ensures that integration complexity is considered, basing application design on ownership ensures a sustainable organizational embedding. As a consequence, the two two-dimensional architecture models of BSP and STP should be combined which resulted in a four-dimensional model. Two of the four dimensions should however be merged in order to avoid the illustration and communication problems of a four-dimensional model.

The following merge options exist:

- Data clusters and functional clusters: Since different structuring principles govern data architecture and functional architecture, these dimensions are usually incompatible and should not be merged.
- Data clusters and business units: Data ownership could provide a consistent foundation for merging data clusters and business units. However if data ownership is not consistently specified and / or enforced in an organization, a merger would create problems.
- Business processes and functional clusters: It is one of the key features of business processes that they are defined across functional clusters. Therefore it cannot expected that a merged dimension can specified consistently.
- Business processes and business units: In a modern organization, most business units should be specified according to business processes and should also be owners of the respective business processes. If the scales are generally compatible, the merged dimension reflects the product (or service) structure as well as the organizational structure creating these products (or services).

Hence the business process dimension and the business unit dimensions are merged to transform the four-dimensional model into a three-dimensional model that comprises the following dimensions:

1. **Organization / product line**: Business units or product lines or market segments around which applications may be integrated. The integration is across several functionalities and several information subjects.
2. **Functionality**: Reused functionalities around which applications may be integrated. The integration is across all organizational units and information subjects reusing these functionalities.
3. **Information subject**: Information subjects or core data around which applications may be integrated. The integration is across all organizational units and functionalities reusing these information subjects or core data, respectively.

The resulting three-dimensional model is illustrated by figure 3. For a retail bank, typical values for the organization / product line dimension would be mortgage processing, loans processing or deposits processing. For the functionality dimension, typical values would be manage contracts, process payments, check authorization, create statements, or use a specific communications channel (e.g. web portal, call center). For the information subject dimension, typical values would be customer knowledge, product data, tariffs, risk, or revenues / costs.

Using this model first sketched in (Winter 2001), an application can be represented as a cube referring to the appropriate values of the organization / product line, functionality, and information subject dimensions. If all applications are modeled using this model, deficiencies

of existing application design can be visualized and, using an appropriate metric, even quantified. Furthermore, it can be shown that certain design decisions avoid overlaps, fill gaps or enable more efficient application interfacing.

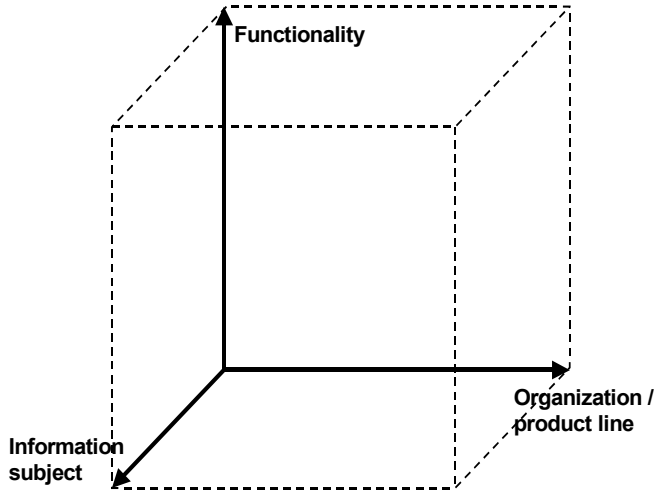


Figure 3. Three-dimensional model for representing application architecture

Using the architecture model for application design

The architecture model proposed in the preceding Section is now used to illustrate deficiencies of many existing, historically grown application architectures. Based on that analysis, an ideal, 'architected' application design is presented. By such a design, intra-application integration costs as well as costs for controlling redundancies could be reduced significantly. On the other hand, however, extensive interfacing needs between different types of applications are created.

Analysis of existing application design

Applications are usually not developed in an 'architected' way, i.e. strictly following a set of specific design rules that avoid overlaps, avoid gaps and are oriented at only one integration direction (organization / product line, functionality, or information subject).

Historically, integration along organizational structures or product lines dominated for a long period. Examples for such applications are a mortgage processing system or a securities trading system. With the growing significance of databases and management information systems, later applications were developed that are integrated along an information subject (e.g. revenue controlling system, risk management system). More recently, the growing number of distribution and access channels and the desire to offer services consistently over multiple channels led to the development of applications that are integrated around a specific channel or that reuse specific functionalities (e.g. e-commerce system, call center system, access security system).

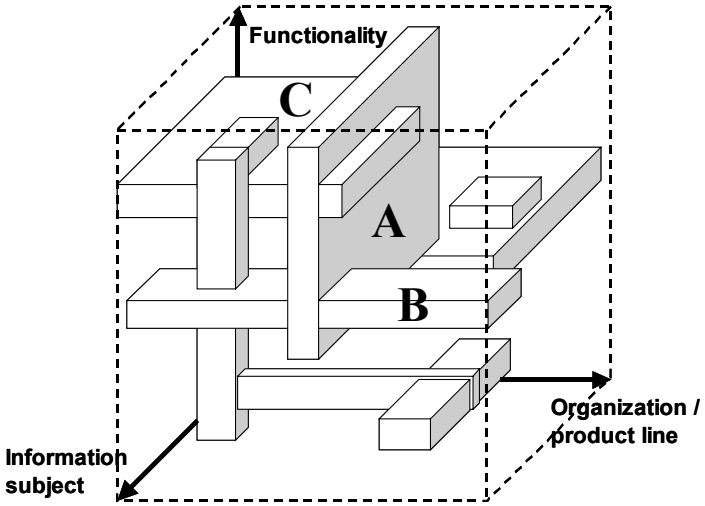


Figure 4. Three-dimensional model for representing application architecture

Figure 4 illustrates a situation that is typically found in most medium and large companies: Since the integration focus changed over time and no consistent architecture management was enforced, many applications overlap (i.e. functionalities or data usages are implemented in more than one application) and many gaps are present (i.e. certain functionalities are not available for every channel or reused by every process). In figure 4, 'A' denotes a typical organization / product line centered application, 'B' denotes a typical information subject centered application, and 'C' denotes a typical channel or functionality centered application. Please note that the graphical cube representation of A, B and C does not imply that these (or other) applications are implemented as monolithic pieces of software. Usually, all applications comprise a number of modules, data structures and / or components that are closely related and therefore integrated by appropriate implementation mechanisms (common database, data transfers, message pool, etc.).

Ideal application design

In order to avoid overlaps or gaps and thereby reduce integration costs, applications should be 'architected', i.e. production / service execution, information processing and channel functionalities (or other reused functionalities) have to be implemented by different applications. In the proposed model, such 'architected' applications would be completely integrated around only one model dimension, i.e. they focus on a certain value in only one dimension and extend across all values of the other dimensions:

- Organization / product line centered applications should integrate all functionalities and all information subjects for a certain organizational unit or product line.
- Information centered applications should integrate access to a certain information subject across all organizational units / product lines and across all functionalities.
- Functionality centered applications should make a certain functionality available across all organizational units / product lines and across all information subjects for consistent reuse.

The ideal application architecture that results from strictly 'architected' application design is illustrated by figure 5.

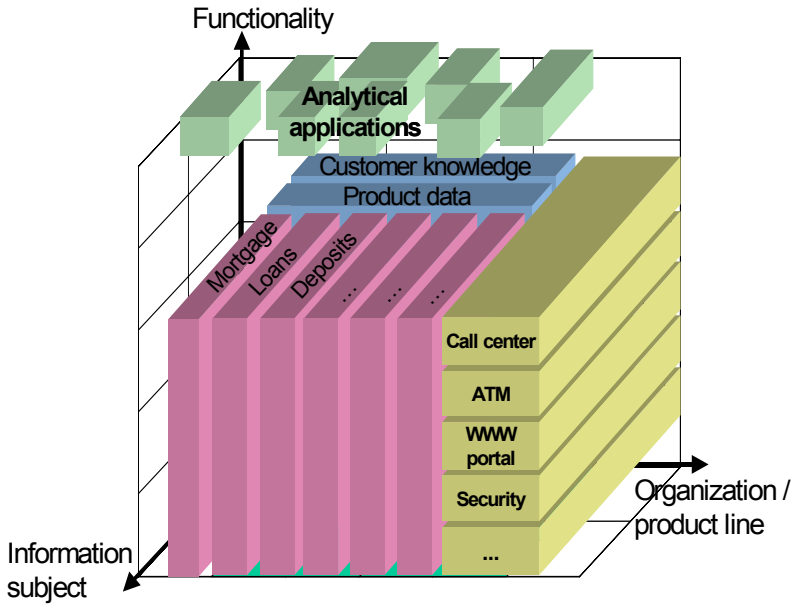


Figure 5. Ideal application architecture

Although being oriented on information subjects and hence having similarities with information centered applications, analytical applications are represented by a separate segment in the model. This is due to the fact that analytical applications support management processes rather than operational processes. As a consequence, different functionalities are implemented which results in different values regarding the functionality dimension.

Organization / product line centered applications are often designated as 'vertical' applications. Functionality centered applications (and particularly channel-oriented applications) are often designated as 'horizontal' applications. In contrast to analytical applications that support management processes, vertical, horizontal and information-centered applications are often generalized to 'operational' applications because they support operational business processes (Winter 2001).

In comparison to non-architected, historically grown application architecture, ideal application design reduces intra-application integration costs as well as costs for controlling redundancies. Moreover, distribution / access channels can be more flexibly assigned to products / services and common functionalities can be reused more efficiently if product processing, channel functionalities and reusable functionalities are assigned to separate applications.

The transformation of non-architected applications into an ideal or near ideal application architecture is an expensive and time-consuming process. The separation of information processing and product / service execution has been on the IS architect's agenda since the 1980ies and in many companies is still not completed. But the expected integration cost savings and flexibility gains of architected applications are so huge that large service companies have started large application re-architecture projects. A recent example is UBS's re-architecture project that is scheduled to be completed around 2007 after having spent several millions of Swiss Francs (Knecht 2003).

Unfortunately, the separation of product / service execution, information processing, and channel functionalities / reused functionalities creates massive interfacing requirements which lead to high inter-application integration costs. The following Section investigates how

inter-application integration costs can be reduced without invalidating ideal application design.

Using the architecture model for application interfacing

If applications are created according to the proposed design principle, all intensely coupled modules, data structures and / or components are aggregated into respective operational or analytical applications, while interfaces are used to implement loose couplings. As a consequence, there should only a relatively small number of interfaces be necessary between different applications of the same type. However, for several reasons a much higher number of interfaces must be implemented between different types of applications:

- Analytical applications source their data from operational applications. In the worst case, a data sourcing interface is needed between every analytical application and every operational application.
- Horizontal (channel oriented) applications provide distribution and access functionalities for vertical (product oriented) applications. Information-centered applications provide information access functionalities for vertical and horizontal applications. In the worst case, an interface is needed between every operational application and every other operational application.
- Operational as well as analytical applications exchange or share data with applications outside the regarded company to support inter-company business processes (e.g. common planning in a supply chain, sharing of customer data in a multi-industry loyalty program). In the theoretically worst case, an interface is needed between every application in the regarded company and every inter-company application.

Although worst case scenarios are rare, large companies have developed and are maintaining thousands of interfaces to implement such couplings.

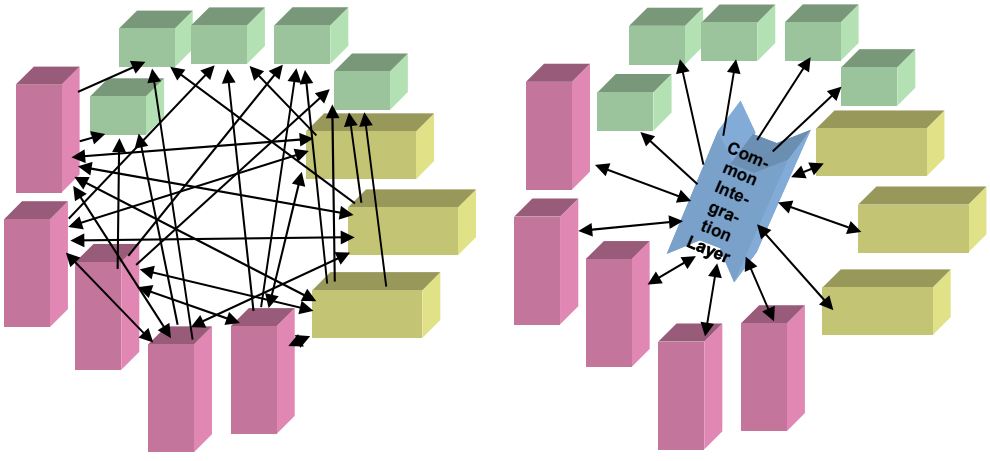


Figure 6. Individual interfaces (left) vs. hub and spoke concept (right)

Reduction of interfacing requirements by common integration layers

The most straightforward approach for reducing the number of interfaces between applications is to introduce a common integration layer. Instead of having to develop and maintain *n*(n-1)* interfaces (theoretically worst case) between *n* applications, only *n* adapters to the common integration layer are needed. This 'hub and spoke' principle is illustrated by figure 6. During the last two decades, the software industry has developed three types of common integration layers:

- **Data warehouse** (DWH) systems provide a single, consistent information base for analytical applications by integrating (i.e. collecting, transforming and cleansing) data from multiple sources. Instead of having to develop and maintain a read-only interface between every analytical application and all its relevant source applications, the DWH system does only need one extraction interface to every relevant operational application and one channeling interface to every supported analytical application or data mart (Devlin 1997).
- **Enterprise application integration** (EAI) systems provide a platform that can be used to publish and subscribe to data updates or other messages between applications. Beside messaging oriented solutions, functional or process oriented integration technologies may be used in EAI concepts as well. Their common goal is to overcome the need to develop and maintain an uni- or bi-directional interface between every pair of applications that exchanges data or other messages. The EAI system does only need one interface to every relevant operational application, which reduces complexity and costs (Rowohl and Schelp 2002).
- The **business collaboration infrastructure** (BCI) provides a set of services and standards to exchange data updates and other messages between applications in different companies. Instead of having to develop and maintain an uni- or bi-directional interface to / from every application that exchanges data or other messages with an external application, common BCI services and standards can be used (Cäsar et al. 2002).

Although the general integration principle is identical for these three types of common integration layers, their implementation is completely different. DWH systems integrate only by data. EAI systems integrate mostly by message brokering; data oriented EAI (e.g. by operational data stores) is a complementary implementation technique used for integration with DWH systems and for integration needs that require high data quality and that tolerate delayed data availability (Imhoff 2000). The BCI does neither provide significant data nor message brokering. Instead, standards and services are made available that allow for a dynamic and easy configuration of interfaces (Cäsar at al. 2002).

Ideal application interfacing

If 'architected' application design is strictly applied as described in section 3.2 and if common interface layers are utilized for application interfacing, the resulting application architecture meets the optimization criteria stated in the introduction: Intra-application integration costs are optimized because all tightly coupled functionalities / data are integrated and only loosely coupled functionalities / data are interfaced. Inter-application integration costs are minimized because the number of interfaces is minimized by using shared, integrated data, publish-and-subscribe mechanisms, common services and common standards

whenever possible. A generic application architecture that can serve as a reference architecture is illustrated in figure 7. Messages are denoted by envelope symbols, shared data by disk symbols, and services as well as applications by cubes.

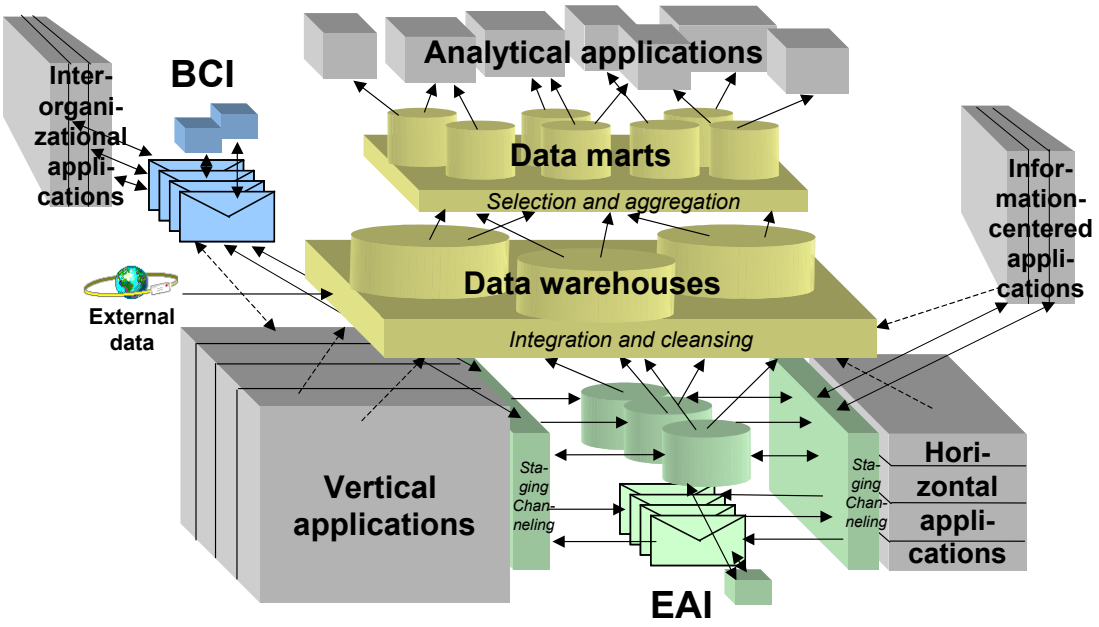


Figure 7. Generic reference application architecture

Conclusions

Architecture management can only be successful if reference architecture models and appropriate decision rules are provided. Application architecture models can support IS management by providing application development directions and by supporting integration decisions. The goal of this approach is to optimize application architecture by balancing intra-application integration costs and inter-application integration costs. If application design ('micro integration') observes the proposed design principle, non-redundant applications are created that reflect tight interconnections between data usage, functionality reuse and ownership. If application interfacing ('macro integration') is deploying common integration layers whenever possible, efficient interfacing and minimal inter-application integration costs (for the respective application structure) are realized.

Although being based on experience and insights gathered from a large number of actual integration projects in large service companies, the proposed reference models and design rules need to be validated quantitatively. In order to create a comprehensive 'integration method', components such as project classification, business case templates, reference models for organizational structures / responsibilities and reference processes for architecture management and meta data management still have to be developed and validated.

By collecting architecture management experience from actual business practice and using these insights to refine and conceptualize respective reference processes, it seems realistic that even application architecture benchmarking would become possible in the near future. Then, finally, application design and application architecture would evolve from their current

state of mostly informal illustration and qualitative guidance into a powerful tool for IS management.

References

Cäsar, M.; Alt, R.; Grau, J. (2002), Collaboration in the consumer product goods industry - Analysis of marketplaces, *Proceedings of 10th European Conference on Information Systems* (ECIS), Gdansk.

Devlin, B. (1997), *Data Warehouse - from Architecture to Implementation*, Addison-Wesley, Reading etc.

Frese, E. (1995), *Grundlagen der Organisation*, Konzept – Prinzipien – Strukturen, 6. Aufl., Wiesbaden.

IBM Corp. (1984), *Business Systems Planning - Information Systems Planning Guide*, 4th ed., IBM-Form GE20-0527-4, Atlanta.

IMG AG (2000), *Promet® STP, Methodenhandbuch für die System- und Technologieplanung*, Version 1.1, St. Gallen.

Imhoff, C. (2000), The Corporate Information Factory, *DM Review*, December 1999, accessed 2000-03-29, <http://www.dmreview.com/editorial/dmreview>.

King, W.R. (1995), Creating a Strategic Capabilities Architecture, *Information Systems Management*, 12 (1), pp. 67-69.

Knecht, R. (2003), *Enterprise Architecture*, lecture notes, Executive MBA in Business Engineering, University of St. Gallen, accessed 2002-02-01, <http://seward.iwi.unisg.ch/bec/becportal.nsf>.

Krcmar, H. (1990), Bedeutung und Ziele von Informationssystem-Architekturen, *Wirtschaftsinformatik*, 32 (5), pp. 395-402.

Malhotra, Y. (1996): Enterprise Architecture: An Overview [WWW document]. @BRINT Research Institute, accessed 2002-09-13, <http://www.brint.com/papers/enterarch.htm>.

McDavid, D. W. (1999), A standard for business architecture description, *IBM Systems Journal*, 38 (1), pp. 12-31.

Österle, H.; Brenner, W.; Hilbers, K. (1992), *Unternehmensführung und Informationssystem*, Teubner, Stuttgart.

Penzel, H.-G. (2002), Die Universalbank neuen Typs - mit einer Informationstechnologie neuen Typs, *Proc. of the Third Conference on Innovations in the Banking Industry* (CIBI 2002), Institut für Bankinformatik, Regensburg.

Rowohl, F., Schelp, J. (2002), Enterprise Application Integration, Khosrowpour, M. (Ed.): *Issues & Trends of Information Technology Management in Contemporary Organizations*, Idea Group Publishing, Hershey etc., pp. 775-779.

Scheer, A.-W. (2000), *ARIS - Business Process Modeling*, 3rd ed., Springer, Berlin etc.

Sinz, E.J. (1997), Architektur von Informationssystemen, Rechenberg, P.; Pomberger, G. (Eds.): *Informatikhandbuch*, Hauser, München, pp. 875-887.

Spitzer, P. (2002), Gesamtarchitektur aus Sicht der HVB Group, *Proc. of the Third Conference on Innovations in the Banking Industry* (CIBI 2002), Institut für Bankinformatik, Regensburg.

Winter, R. (2001), The Current and Future Role of Data Warehousing in Corporate Application Architecture, Sprague, R. (Ed.): *Proceedings of the 34th Hawaii International Conference on System Sciences* (CDROM), IEEE 2001, 8 pages.

Winter, R. (2003), Modelle, Techniken und Werkzeuge im Business Engineering, to appear in: Österle, H.; Winter, R. (Eds.): *Business Engineering - Auf dem Weg zum Unternehmen des Informationszeitalters*, 2nd ed., Springer, Berlin etc.

Youngs, R.; Redmond-Pyle, D.; Spaas, P.; Kahan, E. (1999), A standard for architecture description, *IBM Systems Journal*, 38 (1), pp. 32-50.

Zachman, J.A. (1987), A framework for information systems architecture, *IBM Systems Journal*, 26 (3), reprinted in IBM Systems Journal 38 (2&3), pp. 454-470.

Zachman, J.A. (1999), Enterprise Architecture: The Past and the Future [WWW document], *DM Review*, December 1999, accessed 2002-09-13, <http://dmreview.com>.