Wojciech Ganczarski, Robert Winter

# On the Interplay of Organizational Architecture and Software Architecture

*Enterprise architecture frameworks sometimes provide an additional architectural layer between business-oriented artefact types (e.g., business processes, organizational units) and technical artefact types (e.g., software components, data structures). This "integration" or "alignment" layer is intended to bridge the gap, which results from different life cycles, different ownerships, and other sources of IT/business misalignment. The development of specific models and artefact types on the integration layer is in its early stage. Existing methods for information systems design constitute a first starting point. However, most of them lack a clear differentiation between the integration layer and the software layer and therefore cannot be reused as-is. This paper contributes to the research of design methods, models, and artefact type specifications for the integration layer. The focus lies primarily on the alignment of organizational architecture and structural software architecture, two important components of enterprise architecture. A comparison of organizational and software architecture design methods yields that both types of structures are usually constructed according to different design criteria so that un-aligned architectures result. Traditional integration artefacts, such as "logical" applications, which specify coherent areas of ownership over software artefacts, are too closely linked to actual software system structures and therefore usually fail in aligning with the organizational architecture. It is argued in this paper that instead, integration artefacts should be much more decoupled from actual software structures.*

## 1 Introduction

The lack of alignment between information systems and their corresponding business environment is seen as one of the root causes for why today's companies are not able to realize value from their IT investments [HeVe93]. By focusing on the interplay of organizational architecture and structural software architecture, this paper concentrates on a subset of the multifaceted IT/business alignment problem. Organizational architecture is understood as the fundamental organization of activities and decision rights over corporate assets. Structural software architecture is understood as the fundamental organization of software systems (or business applications as a specific class of software systems).

By looking at a company's structural software architecture, it is often possible to tell the industry the company is operating in, or to generate first hypotheses about the company's business model. However, it is much less straightforward to conclude the company's organizational architecture. Why is it that organizational architecture and structural software architecture are usually so different? According to Orlikowski and Barley [OrBa01], structural differences between organizational and software structures can be attributed to epistemological differences of information systems (IS) research and organizational research. Whereas IS research focuses on the functioning of engineered artefacts, organizational research is more concerned with the interrelations of human behaviour [OrBa01]. Enterprise architecture frameworks, which acknowledge those structural differences explicitly foresee a specific architectural "integration" or "alignment" layer which is comprised of models that link business-related architectures with technical architectures. A simple ownership allocation model (OAM), which specifies the decisions rights of organizational units over individual software artefacts, is one example for such an integration model. Beyond ownership, information exchange constitutes a further relationship that can be found between the business and the IS environment. Other types of integration models can therefore be utilized to ensure alignment in terms of information exchange.

Whereas the information exchange relationship has already been thoroughly analyzed in the work of other authors (e.g., see [Wall96]), only few contributions have been dedicated to the ownership relationship.

Therefore, the goal of this paper is to analyze the effectiveness of existing integration models, especially OAMs based on "applications", and to propose modifications to those models in order to enhance the alignment between the business and the IS environment. But before those models can be analyzed, a detailed understanding of the reasons is necessary for which organizational architecture and software architecture structurally diverge.

The paper is organized into six sections. Following this introduction, Section 2 discusses related work. In Section 3, the (extended) Business Engineering framework is introduced which will be used as the frame of reference. Section 4 will provide a comparison of organizational design and state-of-the-art approaches for IS and software architecture design. In order to understand the effectiveness of OAMs based on "applications", the property rights theory is applied in Section 5. A modification is proposed on how more effective ownership allocations over software artefacts could be defined. The paper closes in Section 6 with an outlook on how those insights can be taken further to support in the development of effective integration models.

## 2 Related Work

The analysis of differences and commonalities between the design of organizational architecture and software architecture fits into the subject area of IT/business alignment. In 1993, Henderson and Venkatraman introduced the Strategic Alignment Model [HeVe93], which since then is referred to as the standard framework for classifying IT/business alignment issues. They distinguish four individual problem domains which require mutual alignment: business strategy, organizational infrastructure, IT strategy, and IT infrastructure. Within this framework, the organizational architecture constitutes one piece of the organizational infrastructure, whereas the software architecture belongs to the IT infrastructure. The question under analysis within this paper can therefore be assigned to the problem domain of aligning organizational infrastructure and IT infrastructure. While proposing roles and responsibilities, Henderson and Venkatraman do not provide a specific alignment method.

Further work which deals with the alignment problem can be found within the subject area of IT Governance, a term coined by Weill and Ross in 2004 [WeRo04]. Weill and Ross define IT Governance as an act of specifying the decision rights and the accountability framework to encourage desirable behaviour in using IT [WeRo04]. They distinguish between decisions concerning IT principles, IT architecture, IT infrastructure strategy, business application needs, and IT investments [WeRo04]. Through an analysis of IT Governance processes in 300 international companies, they found a correlation between "good" IT Governance and enterprise performance [WeRo04]. Although various best practice examples for effective IT Governance processes are provided in their work, Weill and Ross do not propose specific integration methods or models which could help companies in actually defining and monitoring IT decision rights.

With respect to the work of Weill and Ross, this paper will focus on the analysis of business application decision rights which empower organizational units to buy, build, or adapt individual business applications [WeRo04]. Ownership as it is used here refers to a mandate which is granted to organizational units by the top management. It empowers the grantees to make decisions about buying, building or adapting individual business applications and makes them accountable for the effective use of these resources.

According to Weill and Ross, those decision rights are the least mature and also least understood within today's companies. Once allocated badly, they may lead to redundant system capabilities, wasted resources, and excessive time to market.

## 3 Modelling Enterprise Architectures

### 3.1 The business engineering framework

Business Engineering (BE) [ÖsWi03] constitutes a comprehensive and integrated approach to the design and evolution of business architecture, organizational architecture, and IT architecture. Due to its focus on consistent "business-to-IT" design, BE is adopted as the terminological and methodological foundation for this paper. The initial BE framework as described by [Wint03a] consists of three major architectural layers: strategy, organization, and IS. In particular the IS layer has been extended and detailed by recent work of, e.g., Schelp and Schwinn [ScSc05]. This paper is based on the five-layer extension of the BE framework which comprises the following architectural layers:

The **strategy layer** represents the business architecture which positions the company (or business unit, or company network) according to market needs and competencies. It provides specifications of products / services and organizational goals from a strategic perspective [BrWi05]. Design criteria for the business architecture can be derived from different approaches to strategic management [WiFi07].

On the **organization layer**, the static and dynamic organization of service development, service creation, and service distribution are represented [WiFi07]. The organization layer can be subdivided into the process architecture which provides specifications for business processes, and the organizational architecture, which specifies individual organizational units, their relationships and key performance indicators [BrWi05]. Major design principles for this layer are effectiveness and efficiency [WiFi07].

On the **integration layer**, IS components are organized in the relevant enterprise context [WiFi07]. The integration layer represents the fundamental dependencies and links between IS components (or software artefacts, such as software components and data structures) on the one hand and business requirements (specified by processes, organizational units, responsibilities, performance indicators, and informational flows) on the other hand [BrWi05]. Integration architecture design focuses on agility, cost efficiency, integration, and speed [WiFi07], but may also be aimed at transparency, consistency, and/or simplification.

The **software layer** can be subdivided into the structural and the behavioural software architecture. The structural software architecture represents the fundamental organization of software artefacts, such as software components and data structures [WiFi07], which altogether form software systems. Software components can be installed, configured, and executed whereas data structures can be stored, retrieved and manipulated. Software services constitute specific software components which are designed according to W3C Web Services design standards. Beyond that, the behavioural software architecture represents dynamic aspects, such as data flows or interactions of software components, services or systems. The general design goals for this architectural layer are high reuse of software artefacts [BrWi05] as well as minimization of data redundancies [ScSc05].

The **infrastructure layer** represents the organization of computing and telecommunication hardware [WiFi07]. Design criteria on this layer are, e.g., efficiency of resource usage, robustness, and scalability.

The explicit separation of the integration and the software layer constitutes one of the major characteristics of the extended BE framework. It is motivated by the notion of the business environment (represented on the strategy and on the organizational layer) and the IS environment (represented on the software and on the infrastructure layer) as two separate, self-contained systems connected via a *provider-consumer-relationship*. Within this relationship, the IS environment provides *services* to the business environment.

As good practice, general systems theory proposes to shield the internal design of the provider system from the consumer system so that changes of the provider's internal design are not propagated to the consumer [AiWi08] and the overall system's manageability is enhanced. This shielding, also referred to as *loose coupling*, can be achieved through the introduction of an additional abstraction layer between both systems. This abstraction or *integration layer* (technically spoken) offers a stable interface to the consumer, translates consumer requests into the internal design of the provider, and finally transforms the provider's response back into the format of the stable interface. Such an integration layer is not only capable of buffering changes of IS environment and prevent adoptions on the business environment, but also vice versa [ScWi07b]. Due to significant differences in innovation cycles of organizational artefacts (lifetimes of up to 2 years) and software artefacts (lifetimes of more than 10 years) [BrWi05], buffering changes from each of the layers becomes a crucial capability for IT/business alignment.

This loose coupling can be compared with the loose coupling introduced by the Java Virtual Machine (JVM), which allows to port software systems seamlessly between multiple hardware platforms. Here, changes of the hardware platform are fully buffered through the JVM and not propagated to the software system. Key to the JVM is that it offers the consumer (or software system) the Java programming language as a stable, hardware-independent interface towards the services of any specific hardware platform.

Similarly as the JVM, the integration layer requires a stable interface which is comprised of artefacts that encapsulate the internals of the software layer and that can be referenced by the organizational layer. Artefacts, such as logical applications, which will be detailed in the following paragraphs, are already in use today. It is however argued in this paper that those artefacts and corresponding interfaces are not sufficiently shielding the organizational layer from the internals of the software layer and therefore need to be adapted.

### 3.2 Information systems, business applications and logical applications

The precise understanding of what constitutes an "information system" compared to an "application", and a "software system" varies considerably.

According to Ferstl and Sinz, an *information system* is defined as a set of communicating corporate objects, which work on or deal with information resources [FeSi95]. Those corporate objects can be either of manual or automated type. Human beings represent manual corporate objects whereas automated

corporate objects are implemented within *business applications*. The term "application" is sometimes also used in a slight different context. For example, the BE framework regards *applications* as logical constructs which link implemented IS functionalities and information subjects to joint business ownership and respective activities ([BrWi05] or [ScSc05]). Those *applications* are specified as components of application architecture [WiFi07], which itself constitutes a component of the integration layer. But how do these understandings of business applications and applications match with each other?

*Business applications* are specific software systems which implement corporate objects. They consist of software artefacts and can be ascribed to the software layer. Their fundamental organization is specified within the software architecture, which is part of the IS architecture. In comparison to business applications, other types of software systems may for example provide solely infrastructure services without implementing specific corporate objects.

Compared to that, *applications* that define ownership areas are logical constructs and belong to the integration layer. To avoid misunderstandings in the following, we will explicitly designate these constructs *logical applications*. Logical applications are not composed of software artefacts and therefore can be neither installed, configured nor executed. Instead, they are *linked* to software artefacts on the one hand and business artefacts on the other hand. Due to their relational nature, logical applications are capable of representing organizational ownership of software artefacts.

For many state-of-the-art IS design methods, the "software" and the "logical" view of an application is tightly coupled. Either the structure of a business application automatically determines the structure of a logical application or vice versa (1:1 relationship). Sometimes, logical applications refer to a set of business applications, while the structure of all business applications jointly again determines the logical application (1:n relationship). A n:m relationship, which would allow to structurally decouple software architecture from integration architecture is not foreseen within those approaches.

# 4 Comparing Organizational and Software Architecture Design

## 4.1 Organizational design

Organizational design is concerned with the structuring of goal-oriented social systems [HiFU94]. The need for structuring a social system stems from the limited information processing and problem solving capacities of individual human beings, a phenomenon also referred to as the "organization problem" [Fres05]. Two basic types of structures for the organization of a social system are usually distinguished: a static organizational architecture and a dynamic process architecture [Schr96].

Companies are specific social systems whose performance considerably depends on the quality of their organizational design [Mack86]. Hill et al. point out that the evaluation of any organizational design therefore needs to be conducted on the basis of how well the design supports overall performance [HiFU94]. Frese proposes to evaluate an organizational structure against two major criteria: a) its *coordination efficiency,* and b) its *motivational efficiency* [Fres05]. Coordination efficiency is primarily concerned with the efficiency of a corporate system to utilize its resources and potentials, whereas motivational efficiency focuses on how well the system is able to motivate individual human beings to maximize their performance. Hill et al. also differentiate two basic evaluation criteria that are very similar to those of Frese, which they call *productivity* and *socio-emotional satisfaction* [HiFU94].

But how are organizational architectures actually constructed? According to Hill et al. [HiFU94], tasks, competencies, and responsibilities constitute the fundamental structural elements of static organizational design [HiFU94]. Those fundamental elements are grouped within organizational units, departments, and divisions, which itself are linked within hierarchies. Multiple forms of structuring exist. For example, Hill et al. differentiate four approaches: a) *functional* structuring, b) *object-oriented* (or *divisional*) structuring, c) *regional* structuring, and d) *phase-oriented* structuring [HiFU94]. Functional structuring results in functionally oriented organizational units. Object-oriented structuring is primarily concerned with the objects that tasks are performed upon. Those objects can be, e.g., product lines or customer segments [HiFU94]. There are different opinions on what is exactly meant by object-oriented structuring. Whereas Hill et al. [HiFU94] as well as Frese [Fres05] only consider product and market segment based structuring as object-oriented, for Schreyögg [Schr96] regional structuring constitutes a further sub-type of an object-oriented structuring. In contrast to that, Hill et al. see regional structuring as a separate dimension. Finally, phase-oriented structuring distinguishes between tasks, competencies, and responsibilities that are either related to the planning, execution, or monitoring phase of an enterprise's value generation process.

In addition to the formation of organizational units, a coordination mechanism which defines the basis for their collaboration needs to be put in place. Two

separate coordination paradigms can be distinguished. On the one hand, *hierarchical coordination* organizes units as a hierarchy which empowers individual units to govern and coordinate its subordinated units. On the other hand, *market coordination* establishes internal markets within a company which coordinate relationships between organizational units primarily on the basis of internal market prices [Fres05].

Even though market coordination concepts, such as profit centres or agreements on transfer prices are widely used, hierarchical coordination can be still regarded as the primary coordination mechanism employed in today's companies. The deployment of market mechanisms as coordination mechanisms within a company has its clear limitations which originate in the nature of a firm as described by Coase [Coas37]. Thus, companies are required to realize internal scale effects in order to stay competitive. This limits the number of potential partners as well as the negotiation power within internal market relationships [Fres05].

According to Frese, hierarchically coordinated functional, divisional, and regional forms of structuring suffice to describe most of the organizational structures of today's companies. Frese also provides a first comparison of those three forms of structuring with respect to the design criteria of coordination and motivational efficiency. Functionally structured organizations require high coordination between individual organizational units, because most of the business processes span across multiple organizational units [Fres05]. However, functionally grouped organizations provide a high efficiency with respect to resource utilization and utilization of market opportunities, because resources that are required for those activities are bundled within the same unit [Fres05]. Divisionally structured organizations enable organizational units to act with high autonomy [Fres05] and, as an immediate consequence, maximize individual responsibility and therefore motivational efficiency. Schreyögg adds that a divisional (or object-oriented) organizational design enhances manageability and is more result-oriented than a functional design [Schr96]. On the other hand, resource efficiency is lower in a divisional structure, because resources, which are required to execute similar or identical business processes are not pooled (like in a functional setup), but rather need to be replicated for each division [Fres05]. Finally, the main advantage of a regionally structured organization is a high focus on market efficiency. However, similar to the divisional setup, a regional organization requires a replication of resources which reduces resource efficiency [Fres05].

Table 1 summarizes the described design criteria and forms of structuring for organizational design.

## 4.2 Information systems design

Compared to organizational design, the discipline of IS design has not yet agreed on a specific set of design criteria and a well-defined system of structuring types. This could be ascribed to the different maturity levels of both disciplines. Therefore, design criteria and forms of structuring need to be extracted out of representative state-of-the-art approaches for IS design. The following three approaches have been selected for the analysis:

- IBM's Business Systems Planning (BSP) method [Zach82]
- IMG's Promet Systems and Technology Planning (STP) method [IMG99]
- Winter's application architecture model [Wint03b]

**Business Systems Planning (BSP):** The BSP method supports in the construction of a high-level IS architecture. This architecture is comprised of individual "applications" that are determined by their functional

| Architectural layer | Partial architecture in focus | Design criteria | Forms of structuring |
|---|---|---|---|
| Organization layer | Organizational architecture | - Coordination efficiency<br>• Market effectiveness<br>• Resource efficiency<br>• Process efficiency<br>• Delegation efficiency<br>- Motivational efficiency and effectiveness<br>• High responsibility<br>• High manageability<br>• High market pressure | - By functions<br>- By products<br>- By market segments<br>- By regions<br>- By phases |

*Table 1: Summary for the organizational design analysis*

scope (see [Wint03b] and [IBM84]). Those *BSP applications* simultaneously define structures for ownership areas, which are assigned to organizational units (i.e., logical applications), but also set the foundation for the structure of individual business applications. The BSP method is based on a matrix, which represents dependencies between business processes and data clusters. Areas with a high density of dependencies define the functional scope of individual BSP application candidates (see [Wint03b] and [IBM84]). Structuring BSP applications along data clusters intends to ensure data consistency and to avoid data redundancies [Zach82], because, when structured along data clusters, (business) applications define and store data structures uniquely within their own boundaries. On the other hand, structuring BSP applications along business processes shall reduce integration complexity, because there are few data handovers across multiple (business) applications. This is due to the fact that the majority of information flows within a company occurs between individual activities belonging to the same business processes.

**Promet Systems and Technology Planning (STP):** Similar to BSP, the Promet STP approach constructs a high-level IS architecture by defining the functional scope of individual "applications". Again, STP applications simultaneously define the structure for business applications as well as for ownership areas (i.e., logical application). In comparison to BSP, the STP method uses a matrix comprising business units on the one axis and functionality clusters on the other axis. This matrix is primarily intended to document functionality ownership within a given organizational architecture. STP application candidates describe high-density areas of functional ownership (see [Wint03b] and [IMG99]). (Business) applications structured along functionality clusters ensure a high reuse of functionality. On the other hand, (logical) applications, which are structured along business units, ensure clarity of the ownership structure and a sustainable organizational embedding [Wint03b].

**Winter's application architecture model:** Winter found that the BSP and Promet STP methods are complementary to each other and proposed to merge them within a three-dimensional architecture model [Wint03b]. The architecture model is defined along three dimensions: Functionality clusters, information subjects, and business processes [Wint03b]. Structuring (business) applications along functionality clusters shall enhance reuse of functionality. Information subjects correspond to data clusters used in the BSP approach and structuring along those shall enhance data consistency. The third dimension is assumed to represent business processes jointly with organizational units. Winter assumes that the dimension of business processes and that of organizational units could be merged [Wint03b].

Most of the presented approaches for IS design presume a tight link between the structures of logical applications, which define ownership areas on the one hand and business applications that represent actual software systems on the other hand. The objective of the presented design methods is to define an IS architecture that fulfils two goals simultaneously: 1) to satisfy technical needs of the software architecture, and 2) to ensure an efficient integration into the organizational context. BSP, STP, and Winter's proposal are therefore highly influenced by design criteria from pure software architecture design.

This also implies that design criteria used by those approaches cannot be automatically assigned to either the integration or the software architecture layer. We argue that except for the clarity of ownership, all remaining design criteria and structuring dimensions should be assigned to the software architecture layer, since they all have immediate impact on the efficiency of the software architecture as such. Integration complexity, functional reuse and data consistency can be measured within a software architecture even without any knowledge about the organizational environment. On the other hand, clarity of ownership clearly requires a link to the organizational architecture.

Assigning the data consistency and functional reuse criteria to the software layer is consistent with the BE framework. On the integration layer, the BE framework regards integration complexity as a design criterion whereas clarity of the ownership structure is not found explicitly.

### 4.3 Software architecture design

Similar as for IS design, design criteria and forms of structuring for software architecture design are extracted through the analysis of a state-of-the-art design method. The well-known Architecture Trade-off Analysis (ATAM) developed by Kazman et al. has been selected for this purpose [KaKZ00].

**Architecture Trade-off Analysis (ATAM):** ATAM was developed by Kazman et al. in 2000 to formally evaluate individual architectural decisions for software systems. The evaluation of architectural decisions is based on the following predefined quality attributes: *performance, modifiability, availability,* and *security* [KaKZ00]. For each individual architecture evaluation project, a utility tree is constructed to rate the specific importance of those parameters. The utility tree is then used to evaluate multiple optional architectural styles and to make a recommendation. The ATAM method complements the previously presented methods since it focuses on aspects that are solely relevant for the design of software systems, and is not restricted to business applications, as well,

| Architectural layer | Partial architecture in focus | Design criteria | Structuring dimensions |
|---|---|---|---|
| Integration layer | Logical application architecture | - Clarity of ownership | - By organizational units |
| Software layer | Structural software architecture | - Integration complexity<br>- Functional reuse<br>- Data consistency<br>- Performance<br>- Modifiability<br>- Availability<br>- Security | - By business processes<br>- By functions<br>- By information subjects<br>- By time critical procedures<br>- By change dependencies<br>- By high-availability areas<br>- By data security areas |

*Table 2: Summary for the information systems and software architecture design analysis*

as less concerned with the integration of software systems into the organizational environment.

Table 2 summarizes the design criteria and forms of structuring, which have been extracted from the analyses of IS design and software architecture design methods.

### 4.4 Comparison

A first comparison of Tables 1 and 2 shows that structuring along functions constitutes a common form of structuring, intended to enhance reuse and avoid redundancies. In addition, data consistency can be understood as a specialization of resource efficiency criteria, because the goal is to avoid multiple storage locations and therefore synchronization efforts for the same type of data. Within software architecture, this goal is achieved by structuring along information subjects. However, no corresponding form of structuring can be found within organizational design. On the other hand, a structuring along phases, such as planning, execution, and monitoring, is less relevant for software architecture, because software is primarily used as an executing resource and less for (e.g., strategic) planning and monitoring activities. Moreover, motivational efficiency represents a design criterion which is specifically relevant for maximizing human resource performance, but has no relevance for technical artefacts, such as software systems.

Given the assumption that different design criteria and forms of structuring also lead to different structures, these results can be taken as a starting point to explain why organizational architecture and structural software architecture are so different. The analysis shows that both types of structures are usually optimized with respect to design criteria that are specific to each structure and overlap only partially.

## 5 Aligning Organizational and Software Architecture

The previous sections provide a first explanation for the existence of structural differences between organizational and software architecture. Based on these findings, the effectiveness of logical applications – a specific integration and alignment mechanism between those two architectural layers – is now analyzed. Later, a modification to this alignment mechanism will be proposed in order to enhance its effectiveness.

### 5.1 OAM based on logical applications

This paragraph is intended to illustrate how organizational architecture and structural software architecture are aligned traditionally using logical applications. A conceptual model is introduced in order to represent the relevant characteristics of these architectures. Similar to the BE core meta model [ÖWH+07], the conceptual model will use UML class diagrams as its modelling language (see, e.g., [Oest05]).

According to the BE framework, an organizational architecture is specified by a set of organizational units and their relationships [BrWi05]. Organizational design describes an organizational unit in terms of the competencies (or decision rights) that are assigned to it. Among the multitude of competencies which can be assigned to organizational units, a subset empowers individual units to be responsible for the definition of business functions or information subjects. These types of competencies shall be referred to as *definition ownerships*. On the other hand, structural software architecture is specified by a set of business applications and their relationships. As was described above, those business applications group individual software artefacts (software components or services and data structures).

Organizational architecture and structural software architecture can then be connected in the following way: Business functions are implemented within specific software components, and the content of information subjects is stored within data structures. In addition, competencies on the development of business applications are grouped within logical applications, which then are assigned to organizational units. These types of competencies will be designated as application ownerships in the following.

Figure 1 illustrates this (simplified) conceptual model of logical applications as traditional integration artefacts between organizational architecture and structural software architecture.

## 5.2 Alignment issues for OAM based on logical applications

The intention of this paragraph is to show that an ownership allocation solely based on logical applications does not appropriately consider *structural differences* between the organizational and the software architecture. In addition, the property rights theory is utilized to illustrate that a lack of addressing those structural differences results in economically undesirable behaviour.

### *5.2.1 Defining structural differences*

What is actually meant by a structural difference between organizational architecture and software architecture? Before structural differences can be defined, it is first necessary to understand and define structural equivalence: An organizational architecture and a structural software architecture are regarded as structurally equivalent, *if each of the organizational units has ownership over those software artefacts (or fragments) which implement and store exactly those business functions and information subjects that are under its definition ownership*.

Based on the above definition, a structural difference is simply defined *as any deviation from structural equivalence*. Structural differences occur if business functions or information subjects, which are under the definition ownership of a unit A, are implemented or stored within a business application that is owned by unit B. This means that a single business application is comprised of software components and data structures that implement business functions and information subjects of multiple (hierarchically independent) organizational units.

Based on the findings of the third Section of this paper it can be argued that OAMs based on logical applications will inevitably lead to structural differences, because differing design criteria of software and organizational design will result in business application structures, which comprise software components and data structures belonging to at least two (hierarchically independent) organizational units. In the following, a first analysis indicates economic implications which are caused by those structural differences.

### *5.2.2 Foundations of the property rights theory*

The property rights theory (PRT) constitutes an economic theory, which is concerned with the creation and allocation of property rights onto individual economic actors [Mart00] and is based on the theory of asymmetrical information. PRT analyses the implications of alternative allocations of property rights [Fees97], which are defined as the sum of all legally permitted usage rights for a specific resource [Mart00].

According to PRT, the distribution of property rights determines the efficiency of the overall resource usage [Mart00]. Highest efficiency of resource usage can only be achieved if each of the economic actors fully owns the property rights of the resource he or she is using. Only then economic actors can be fully compensated or punished based on the effectiveness of their resource usage. As a consequence, they will deliver their maximal performance [Mart00]. PRT assumes that individuals exploit their possibilities for opportunistic behaviour in the case of shared property

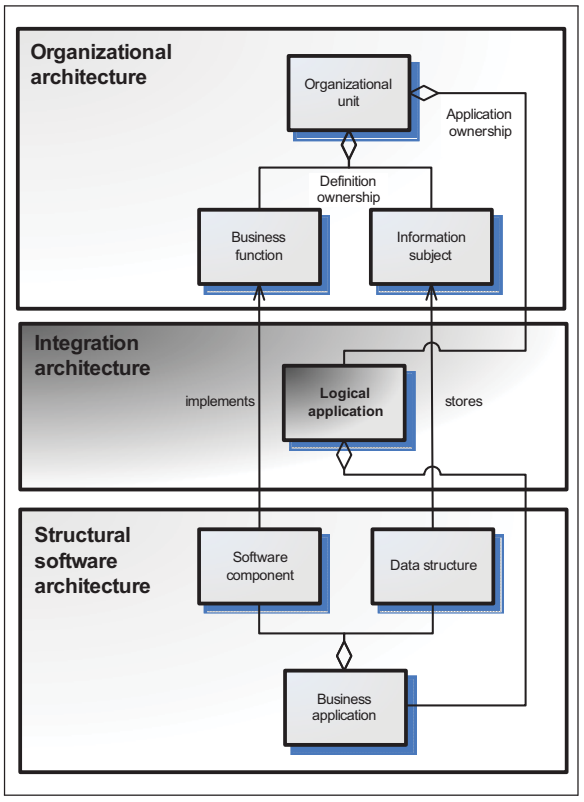


*Figure 1: OAM based on logical applications*

rights, when individual behaviour cannot be observed, a specific type of information asymmetry which is also called *hidden action* [Krae99]. This exploitation allows for an individual optimization, but reduces the effectiveness of the total resource usage.

Two separate approaches are generally distinguished on how resource usage effectiveness can be enhanced. One solution is the clear delineation of property rights onto the resource users; the other solution is the elimination of information asymmetries.

The delineation of property rights requires to setup and control contracts between the involved parties, which itself causes separation costs [Mart00]. On the other hand, information asymmetries, such as hidden actions, can be addressed with appropriate incentive systems [Vari99], where parties using a resource are also compensated or punished on the basis of the total resource usage effectiveness.

### *5.2.3 Structural differences in the light of the property rights theory*

The property rights theory is applied here to explain economic implications of structural differences between organizational architecture and structural software architecture. The following case is constructed: A business application implements business functions and stores information subjects which are under the definition ownership of two hierarchically independent organizational units A and B. Those two units share the ownership or property rights over the entire business application, which leads to a structural difference as defined in 5.2.1. Each of the units has the competence to initiate change projects; moreover development and maintenance costs for the business application are shared between both units.

According to the property rights theory, the lesser both organizational units are able to observe the other parties' behaviour, the higher the incentives for each of them will be to behave opportunistically at the expense of the other. The ability to behave opportunistically also depends on how good development and maintenance costs of the business application can be assigned to the causing organizational unit. It is assumed that each unit has its own cost budget and that assigned development and maintenance costs for the business application are deducted from that cost budget. Based on this, three scenarios can be distinguished:

1) Development and maintenance costs are divided upon both units using a fixed factor, regardless of the actual resource usage of both units
2) Development costs are assigned to the causing unit and only maintenance costs are divided upon both units using a fixed factor
3) Development and maintenance costs are assigned fully to the causing unit

In case 3) none of the organizational units is able to act opportunistically. However, a fair allocation of maintenance costs, sometimes even development costs in the case of one joint business application is usually difficult to achieve. In case 2), both organizational units will behave opportunistically if this behaviour leads to a reduction of their own total costs at the expense of the shared maintenance costs. Practically, this could be done by saving development costs, e.g., by advising "quick-and-dirty" solutions which incur higher long-term maintenance costs. In case 1), organizational units can improve their individual cost situation if they manage to reduce the rest of their cost budget at the expense of development or maintenance costs, which are then shared with the other unit. This could for example be done by saving costs on the elaboration of functional specifications (which is done by the organizational unit's own staff) at the expense of additional development costs.

As this application of PRT shows, structural differences between organizational architecture and software architecture can have undesirable economic implications. In the case of shared business

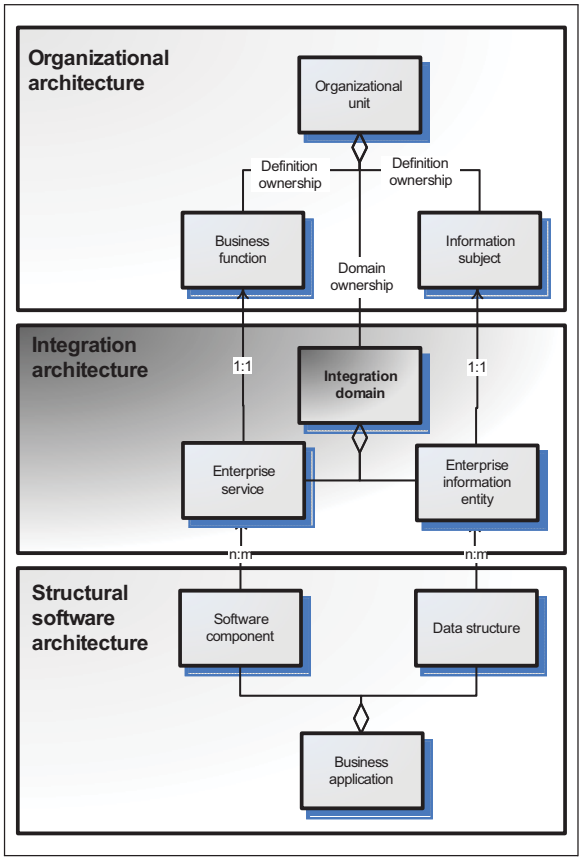


*Figure 2: OAM based on integration domains*

application ownership, individual organizational units are incentivized to behave opportunistically and to reduce their own costs at the expense of the total business application development and maintenance costs. According to PRT, this problem can only be omitted if either all information symmetries are eliminated or if the property rights for the business application resource are separated. The next paragraph introduces a new OAM that aims to provide logical artefacts, which allow to separate property rights for business applications. Those logical artefacts are structured according to business structures instead of software structures (such as in the case of logical applications).

### 5.3 A modified ownership allocation model

If the delineation of ownership on the basis of logical applications is not suitable, how then should an OAM be designed that would encourage more efficient economic behaviour of individual organizational units?

Based on the insights of the former paragraphs, one requirement for an alternative OAM can be stated as follows: logical artefacts should allow to assign ownership over software artefacts (or fragments) that fits well with or at least approximates the definition ownership of organizational units implied by the organizational architecture. In other words, such logical artefacts should be capable of reducing the magnitude of structural differences as defined in 5.2.1.

The most straightforward way to fulfil this requirement is to decouple logical artefacts from existing structures and artefacts of the software layer and structure them according to the design criteria of the organizational layer instead. In order to achieve this, the meta model is modified as follows: First, new logical artefacts named *enterprise services* are introduced on the integration layer. Enterprise services represent implementations of specific business functions within software components. They are therefore linked to business functions using a 1:1 relationship. Enterprise services itself are linked via a n:m relationship with software components. A n:m relationship is required here, because differing design criteria of software and organizational design may require one enterprise service to be implemented within two or more software components. Conversely, one software components may also be used to implement multiple enterprise services.

Moreover, *enterprise information entities* are introduced as further logical artefacts to represent information subjects which are stored within data structures. Again, enterprise information entities are linked to information subjects via a 1:1 relationship, whereas differing design criteria require a n:m relationship towards technical data structures.

Logical applications are replaced by artefacts named *integration domains*. Those integration domains constitute containers of enterprise services and information entities and are assigned to individual organizational units in order to define their ownership over the content of linked software artefacts. Instead of being granted an application ownership, organizational units are now granted *domain ownership*. Figure 2 illustrates the adapted meta model of the new ownership allocation model based on integration domains.

Because enterprise services and information entities are structured according to design criteria of the organizational layer (or can even be seen as structural copies of business functions and information subjects), integration domains offer ownership areas that avoid structural differences. In addition, integration domains are also independent of the underlying software architecture and therefore should be robust towards any technical changes. For example, the consolidation of several business applications should neither effect the structure of logical artefacts nor integration domains. Instead, changing the links of logical artefacts to the newly formed consolidated business applications should suffice to make it consistent with the new software architecture.

Beyond a robustness with respect to changes in the structural software architecture, integration domains should also remain stable in cases when the organizational architecture changes. It should therefore be avoided to define integration domains as structural copies of existing organizational units. Instead, integration domains should be defined on a more granular level so that in case of a restructuring, existing integration domains can be reallocated to the newly formed organizational units.

The challenge for engineering on the integration layer lies now within making those introduced logical artefacts actually tangible. Only if logical artefacts can be delineated from each other, organizational units will accept them as substitutes for logical applications and then can be made fully responsible for their development and maintenance costs.

One possibility to make enterprise service tangible is linking them to a *service interface* offered on the software layer (regardless whether this interface is exposed by a monolithic business application or an elementary software service). But linking is also possible when no service-oriented software architecture is in place, e.g., by linking enterprise services to functions accessible through the GUI of a business application.

*5.3.1 Example*

An example shall be provided that demonstrates the application of the new OAM in the financials domain. The work of a *financial accounting* and a *management accounting* unit (both belonging to the finance department) is supported by a single highly-integrated ERP system. Ownership over the ERP system is delineated between both units using integration domains. On the one hand, the financial accounting unit is responsible for the two integration domains *core bookkeeping* and *legal reporting*. On the other hand, the management accounting unit is responsible for the integration domains *planning* and *management reporting*. Each of the integration domains can be broken down to individual enterprise services. E.g., the legal reporting domain can be further broken down into the two enterprise services *financial consolidation* and *financial statements reporting*.

Integration domains and enterprise services are independent of the actual ERP system's software architecture. Therefore, even if the underlying software structures change (e.g., a new version of the ERP system is introduced or the provider is switched), enterprise services and integration domain structures can remain stable and therefore provide the organizational layer a robust and stable reference to the software layer. However, it is much more difficult to define integration domains that are also stable with respect to organizational changes, since not all possible reorganizations can be anticipated upfront. In the given example, the definition of the integration domains would for example support a reorganization that splits the financial accounting unit into a *book-keeping* and a *legal reporting* unit without having to redefine the integration domains. However, in case of any other possible split, the integration domains would have to be redefined.

## 6 Conclusion

The results of this paper can be summarized as follows: First, it has been shown that the construction of organizational architecture on the one hand and software architecture on the other hand adheres to different design criteria so that inconsistent structures are created. Alignment mechanisms, such as ownership allocation models (OAMs) are required to align these different architectures. It has been argued that a traditional OAM for software artefacts on the basis of logical applications creates opportunities for organizational units to act opportunistically which leads to suboptimal solutions on a company-wide level.

Ownership over IS artefacts therefore needs to be defined not on the basis of business applications, but on the basis of logical artefacts that are structured according to organizational design criteria, thereby more accurately approximating actual ownerships over business functions and information subjects implied by the organizational architecture.

This paper is focused on a rather static alignment of organizational and software architecture. Further analysis might include dynamic aspects. For example, the flexibility of traditional versus more granular OAMs could be compared in cases where organizational architectures change, but software architecture remains unchanged (at least on a short-term basis). The assumption so far is that an OAM based on logical artefacts will be able to adjust more easily to changed organizational architectures compared to a mechanism based on logical applications. This hypothesis however needs to be verified by further research.

## References

[AiWi08] Aier, Stephan; Winter, Robert: Virtual Decoupling for IT/Business Alignment – Theoretical Foundations, Architecture Design and Implementation, Research Report, Institute of Information Management, University St. Gallen, St. Gallen, 2008, pp. 1–22.

[BrWi05] Braun, Christian; Winter, Robert: A Comprehensive Enterprise Architecture Metamodel and Its Implementation Using a Metamodeling Platform. In: Enterprise Modelling and Information Systems Architectures, Gesellschaft für Informatik, Klagenfurt 2005, pp. 64–79.

[Coas37] Coase, Ronald H.: The Nature of the Firm. In: Economica, 4 (1937) 16, pp. 386–405.

[Fees97] Feess, Eberhard: Mikroökonomie – Eine spieltheoretische Einführung, Metropolis-Verlag, Marburg 1997.

[FeSi95] Ferstl, Otto K.; Sinz, Elmar J.: Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. In: Wirtschaftsinformatik, 37 (1995) 3, pp. 209–220.

[Fres05] Frese, Erich: Grundlagen der Organisation: Konzepte – Prinzipien – Strukturen, 9. Ed., Gabler, Wiesbaden 2005.

[HeVe93] Henderson, John C.; Venkatraman, N.: Strategic alignment: Leveraging information technology for transforming organizations. In: IBM Systems Journal, 32 (1993) 1, pp. 4-16.

[HiFU94] Hill, Wilhelm; Fehlbaum, Raymond; Ulrich, Peter: Organisationslehre 1: Ziele, Instrumente und Bedingungen der Organisation sozialer Systeme, 5. Ed., Haupt, Bern et al. 1994.

[IBM84] IBM: Business Systems Planning - Information Systems Planning Guide, IBM-Form GE20-0527-4, 4. Ed., IBM-Corporation, Atlanta 1984.

[IMG99] IMG: PROMET STP: Methodenhandbuch für die System System- und Technologieplanung, Release 1.0, IMG AG, St. Gallen 1999.

[KaKZ00] Kazman, Rick; Klein, Mark; Clements, Paul: ATAM: Method for Architecture Evaluation, Software Engineering Institute Carnegie Mellon University 2000.

[Krae99] Kräkel, Matthias: Organisation und Management, Neue ökonomische Grundrisse, Mohr Siebeck, Tübingen 1999.

[Mack86] Mackenzie, Kenneth: Organizational Design: the Organisational Audit and Analysis Technology, Ablex Publishing Corporation, New Jersey 1986.

[Mart00] Martiensen, Jörn: Institutionenökonomie – Die Analyse der Bedeutung von Regeln und Organisationen für die Effizienz ökonomischer Tauschbeziehungen, Verlag Franz Vahlen, München 2000.

[Oest05] Oestereich, Bernd: Analyse und Design mit UML 2, 7. Ed., Oldenbourg Verlag, München, Wien 2005.

[OrBa01] Orlikowski, Wanda J.; Barley, S. R.: Technology and Institutions: What Can Research on Information Technology and Research on Organizations Learn From Each Other? In: MIS Quarterly, 25 (2001) 2, pp. 145–165.

[ÖsWi03] Österle, Hubert; Winter, Robert: Business Engineering – Auf dem Weg zum Unternehmen des Informationszeitalters, 2. Ed., Springer, Berlin et al. 2003.

[ÖWH+07] Österle, Hubert; Winter, Robert; Höning, Frank; Kurpjuweit, Stephan; Osl, Philipp: Business Engineering – Core-Business-Metamodell. In: WISU – Das Wirtschaftsstudium, 36 (2007) 2, pp. 191–194.

[ScSc05] Schelp, Joachim; Schwinn, Alexander: Extending the Business Engineering Framework for Application Integration Purposes. In: The 20th ACM Symposium on Applied Computing (SAC2005), ACM Press, Santa Fe, New Mexico 2005, pp. 1333–1337.

[ScWi07b] Schelp, Joachim; Winter, Robert: Towards a Methodology for Service Construction. In: HICSS-40, IEEE Computer Society 2007, pp. 61–67.

[Schr96] Schreyögg, Georg: Organisation: Grundlagen moderner Organisationsgestaltung, Gabler Verlag, Wiesbaden 1996.

[Vari99] Varian, Hal: Grundzüge der Mikroökonomik, Oldenbourg Verlag, München 1999.

[Wall96] Wall, Friederike: Organisation und betriebliche Informationssysteme – Elemente einer Konstruktionslehre, Gabler, Wiesbaden 1996.

[WeRo04] Weill, Peter; Ross, Jeanne W.: IT Governance – How Top Performers Manage IT, Harvard Business School Press, Boston 2004.

[Wint03b] Winter, Robert: An Architecture Model for Supporting Application Integration Decisions. In: ECIS, Naples 2003.

[Wint03a] Winter, Robert: Modelle, Techniken und Werkzeuge im Business Engineering. In: Österle, H.; Winter, R. (Hrsg.): Business Engineering – Auf dem Weg zum Unternehmen des Informationszeitalters, 2. Ed. Springer, Berlin et al. 2003, pp. 87–118.

[WiFi07] Winter, Robert; Fischer, Ronny: Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. In: Journal of Enterprise Architecture, 3 (2007) 2, pp. 7–18.

[Zach82] Zachman, John: Business Systems Planning and Business Information Control Study: A comparison. In: IBM Systems Journal, 21 (1982) 1, pp. 31–53.

**Wojciech Ganczarski**

Institute of Information Management
University of St. Gallen
Mueller-Friedberg-Strasse 8
9000 St. Gallen
Switzerland
wojciech.ganczarski@unisg.ch

**Prof. Dr. Robert Winter**

Institute of Information Management
University of St. Gallen
Mueller-Friedberg-Strasse 8
9000 St. Gallen
Switzerland
robert.winter@unisg.ch