

Chapter 2

Architecture Management

Summary Architecture management plays a key role in the sustainable success of managed evolution. Architecture management defines the target architecture, a number of models, a set of technical standards and guiding principles for the evolution of the system. It defines and operates processes to evolve architecture standards, to communicate them to the organization and to ensure their implementation. Strong review activities on projects enforce adherence to the standards. Architecture metrics finally allow the progress of key indicators to be measured. A federated architecture organization is essential for powerful architecture management in a very large organization.

2.1 What is IT Architecture and Why Is It Important to Managed Evolution

The Oxford Advanced Learner's Dictionary ([[Oxford_05](#)]) defines architecture as follows:

1. The art and study of designing buildings: to study architecture
2. The design or style of a building or buildings: the architecture of the eighteenth century, modern architecture
3. (Computing) The design and structure of a computer system

Similarly, IEEE (Institute of Electrical and Electronics Engineers in [[IEEE_00](#)]) defines IT architecture as:

“The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution”

The definitions above show two facets of architecture. On the one hand we have “architecture” as an *activity*, art, or profession (Point 1 in the definition), on the

other hand we call the results of the architect's work "architecture" (Points 2 and 3 in the definition).

Understanding IT architecture as an *activity*: In the *architecture process*, architecture is developed, design questions are answered and architectural standards are implemented in the system. In a very large information system, the architect's work at the highest level is very similar to the work of a city planner. The architect defines guidelines and IT standards comparable to city zoning plans, setting constraints for building in certain areas. Reviews and approvals of designs are comparable to implementation permits, just as city planners issue building permits for construction projects. The architect plans and builds the necessary infrastructure to allow for a managed evolution of the city — via IT strategy and architecture driven investment programs.

Understanding architecture as design and *structure*: Architecture consists of its systems, representing the current state of the architecture and of documents defining structure, standards, concepts, principles and roadmaps that govern the future evolution of the system.

It is important to understand that *every* system has an architecture, be it an implicit or an explicit one. If the architecture is implicit we have no way to control, analyze, reason about, evolve and communicate it. It is the role of the architect to establish explicit architecture of the system and to communicate it well to all involved parties.

As described in Chap. 1, there are two ways to improve agility in the managed evolution strategy: One way is to invest into explicit *architecture programs*, the other is to steer every single project into a direction that improves the overall agility (Fig. 1.7). Both approaches only work if there is a clear and explicitly communicated *target architecture*. This is particularly important in a very large information system where thousands of developers make thousands of small design decisions every day. The sum of these micro-decisions determines the overall direction of the system's evolution. So, if more people understand and support the target architecture it will be implemented at lower cost. It is a waste of energy if, due to bad

"Architecture is something of a black art in the IT world. Architects learn on the job, bringing years of experience in design and technology to the business problems they tackle. It's not an easy task to impart architecture knowledge"
Ian Gorton, 2006

communication or bad understanding, design decisions point in opposite directions and annihilate each other. In addition to that, a control element is required to detect and react on deviations from the target architecture.

Furthermore, investment programs to build the necessary infrastructures or to remove duplicate functionality and data are needed.

Fully aware of the fact that architecture and design takes place on a number of different abstraction layers, such as from the detailed design of a data center network all the way up to an abstract enterprise-wide business object model, we avoid a general discussion on IT architecture, which can be found in a number

of excellent books, such as [Masak_05], [Masak_07], [Mens_08], [Gorton_06], [Lankhorst_05], [Abramowicz_07], [Erl_04], [Woods_03], [Garland_03], [Britton_01], [Cummins_09], [Bernstein_09]. We rather focus on elements of IT architecture as far as they are necessary to support managed evolution of very large information systems. In our experience, the key elements to manage the architecture of very large information systems are the following:

- A set of fundamental *principles* guiding the architecture,
- A *structure* that helps divide the very large information system into more tractable pieces,
- A management *process* to develop, communicate, implement and control architecture,
- A federated *organization* that implements the architecture process aligned to the target structure of the system.

In the following we will describe these elements in more detail and add examples from our own experience to illustrate the concepts in a non-trivial case.

2.2 Architecture Principles

Fundamental to the architecture of a very large information system is a set of *architecture principles* to be applied across the whole system. This set of principles should be small and address a couple of key questions, such as the position on technology diversity. It is important that the principles can be described briefly and communicated well across the IT organization. As every developer is making his or her own *micro-architecture* decisions, it is fundamental that these principles are broadly understood, accepted and consistently followed.

Managed evolution as the central architectural principle at the heart of this book is one of the prime architecture principles for very large information systems.

Almost a *meta-principle* to be clarified is that IT architecture is *not* an exact science. There are many conflicting goals and interests in designing a system. Short time-to-market or low-cost development may lead

“‘Architecture’, in a broad sense, is the synergy of art and science in designing complex structures, such that functionality and complexity are controlled”
Marc Lankhorst, 2005

to high operational cost. Design for highest performance may reduce flexibility in parameterization and vice versa. Inserting additional layers of abstraction may increase runtime overhead, but reduce development, maintenance and future enhancement cost. Higher security requirements must be carefully implemented in order not to lead to reduced user comfort and potentially to lower availability and higher operating cost. It is important to be aware of and *accept* those trade-offs. One-dimensional optimization typically leads to sub-optimum solutions.

Compliance oriented governance does not necessarily lead to the desired results in architecture. In order to properly handle the trade-offs, both experience and good quantitative models are needed.

If the system supports a dynamic business, then flexibility and agility with regard to business organization and expansion is a must. Applications tend to live longer than the structure of the organization, the legal environment, or the global footprint. Therefore, flexibility principles with respect to certain dimensions are needed. Here are a few requirements that the authors have encountered across a number of different industries:

- *Multiple entities*: Several legally independent organizations should be able to co-exist in the same system in order to leverage scale effects. Each entity needs a certain degree of independence in order to be able to position itself in the market. It is crucial to set the right degree of independence and to implement flexibility accordingly. Typical topics in this include independent pricing of products and services, appropriate separation of data among legal entities and flexible support for different business organizations and processes.
- *Multiple countries*: The system should be able to support international growth by flexibly accommodating new countries, new markets and new business. Key topics in this area include the support of different languages, different currencies, different legal and regulatory environments and operation across multiple time zones, particularly if your system is based on a daily online/batch cycle.
- *Multiple channels*: Sales channels have become more heterogeneous, especially if the organization sells a purely virtual product that lends itself well for electronic distribution. So, flexibility with respect to different sales channels, direct or indirect, physical or electronic is fundamental to a successful long-term architecture.

Depending on your business, other principles regarding flexibility of the value chain, scalability and the like are needed.

Some principles should address and clarify the *technology risk* appetite. They define the structure of the *technology portfolio*. The principles should answer questions such as: How must the technology portfolio be aligned to technology market mainstream in order to profit from the technology ecosystem? How aggressively shall new and risky technologies be brought in? What is the position on desirable, or not so desirable, diversity in the technology portfolio? Where should exceptions to the principles be allowed? Chap. 4. (Infrastructure) explains the underlying rationale and processes for technology management in more detail.

One fundamental architectural choice to be made is the question of whether components of the system should be bought and integrated or whether the system will be based mainly on proprietary development. This choice is dependent on the size of the organization, on the market of relevant software, on the capabilities of the organization building the system and many other criteria. In the experience of the authors, any very large information system has both types: Proprietary development and buy and integrate components.

The following Side-Story 2.1 explains how a generic requirement is reflected in architectural principles to be applied across the application landscape.

Side-Story 2.1: Multi-Country Capability Architecture

Credit Suisse has IT operations in many locations, some of them large and some of them rather small. However, the customers expect the same high quality service, independently from the location of their bank branch. It is, however, not feasible to install and operate the rich functionality of a large location in all the small locations worldwide.

Therefore, in order to provide richer functionality to the smaller locations and to increase processing volume, Credit Suisse intends to use its Swiss system for global business. This brings a number of challenges, such as handling multiple legal entities, dealing with the specifics of different countries, adapting to different laws and regulations, providing products for local markets and supporting local languages. Traditionally the existing applications do not support international business.

Instead of operating a large number of individual IT installations in many countries, IT services will be offered out of a small number of hubs. A team of specialists defined the required multi-country capability of the IT system, including applications, presentation, reporting, etc. These requirements impact existing applications. Before transforming thousands of applications, the architecture team derived clear and comprehensive principles from the generic requirement. A representative choice of the application landscape architecture principles covering the multi-country capability is listed in Table 2.1.

The principles in Table 2.1 were approved by the architecture steering committee and communicated to the development community. In addition, specific implementation guidelines were published for each of the multi-country application landscape architecture principles, such as the BOUID format specification <BOUID = numeric4>. The complete set of multi-country application landscape architecture principles allows application developers to work on many applications in parallel and to implement the multi-country capability in a short time and with full interoperability.

2.3 Architecture Layers and Models

Models are essential elements of very large information system architecture. They allow the architect to manage complexity and to document the system consistently

"Models must not be regarded as rather inconsequential artifacts of software development but instead as the core products"

Anneke Kleppe, 2009

and precisely. Models help simplify reality by abstraction. The first purpose is the *structuring* of the large number of parts in the

Table 2.1 (Side-Story 2-1) Application landscape architecture principles for multi-country capability

Principle name	Principle text	Remarks
Multi-Entity Capability (MEC)	<p>The entity in MEC (the “E” in MEC) is the <i>Business Operation Unit</i> (BOU). Each BOU is uniquely identified by its Business Operation Unit Identifier (BOUID)</p> <ul style="list-style-type: none">• Each Business Operation Unit belongs to exactly one legal entity• Each Business Operation Unit is associated with exactly one country• A BOUID is assigned only once, i.e. BOUIDs are never reused• A legal entity can span several BOUs and countries <p>All customer- and employee-related data must refer to the BOU of its legal owner. All database structures storing such data must have an attribute <BOUID> which identifies the context</p>	<p>The BOU is the central concept because it allows unambiguous linking of metadata, data and functionality to a unit operating in exactly one organizational (legal) entity and in one jurisdiction (Country).</p> <p>Reuse of BOUIDs is not allowed because otherwise the historical and archive data would become ambiguous</p>
Data Affiliation		Points to the laws and regulations protecting the data
Data Separation	<p>The customer- and employee-identifying data must be separated from the content they refer to.</p> <p>All applications must work properly without client-identifying data.</p>	<p>This is necessary for cross-border anonymization. Instead of the client-identifying data, secure, reversible cryptographic techniques are used, generating a code which allows the processing of the data without exposing the client</p>
Data Authorization	<p>Data access protection must be enforced according to the national laws and regulations and according to the contractual obligations of the different BOUs</p>	
MEC Control	<p>All components have to be designed, implemented and deployed in such a way that new business units can be easily supported, ideally by updating configuration tables</p>	Assures the flexibility for adding new business units
Interfaces	<p>All interfaces over which business operation unit specific data is passed must include the BOUID field</p>	Applies in fact to most interfaces

system (see Sect. 1.2) and the second purpose is shaping the connections. The focus in this book is on models that are important on the enterprise architecture level ([Lankhorst_05]). For the implementation of functionality and data structures, additional models (notably the logical and the physical models) are required, which will not be discussed in this book. Refer to [Schmidt_99], [Simson_05], [Nicola_02] and many other texts for more information on logical and physical modeling.

When working with different models, like domain models, business object models and others, the models must be kept *consistent* at all times. Models evolve gradually along the managed evolution path, thus becoming complex over time. Keeping models consistent among each other require special methods. Very helpful instruments are rich metamodels ([Marco_04], [Frankel_03], [Knöpfel_05], [Inmon_08], [Wegener_07]) underlying each model and automatic model checkers ([Baier_07]) being made available to all model developers.

The most important models for structuring are the *layer models* (Fig. 2.1), the *domain models* (Side-Story 2.3 and Fig. 2.9) and the *business functional map* (Side-Story 2.2). The set of models required for architecture management will be described in this chapter, starting with the *layer model*.

It is very common to manage enterprise architecture as three separate layers. A *business architecture layer* describes the entities, processes and functions on

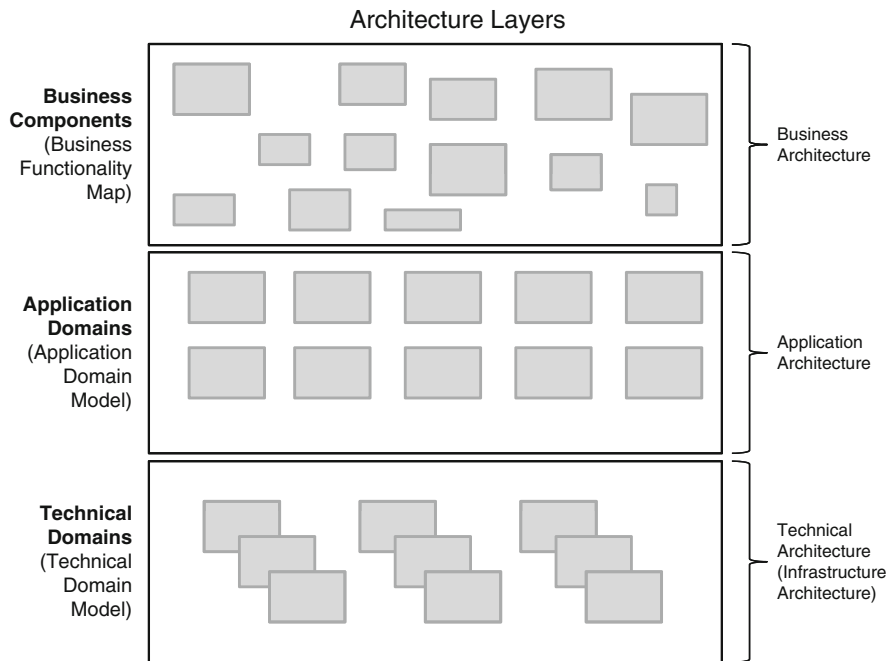


Fig. 2.1 Architecture layers

a business level. An *application architecture layer* describes how business requirements are implemented in applications. And a third layer, the *technical architecture layer* deals with the underlying infrastructure (see Fig. 2.1).

Business architecture is structured according to a *business functional map*. Technical and application architecture are both structured by their *domain models* which are presented later in this book.

2.4 Business Architecture

Systems implement products, processes and services of real businesses. Therefore

"An enterprise system is an extraordinary complex application of information technology designed to support organizational processes in a highly integrated fashion"
Kent Sandoe, 2001

it is not surprising that the system architecture should be derived from overarching *business architecture*. In the literature a number of different

definitions for business architecture can be found. As for all architectures, business architecture is at the same time an activity and a well-structured documentation of a current or future state. The OMG's¹ business architecture working group defines this aspect of business architecture as "A blueprint of the enterprise that provides a common understanding of the organization and is used to align strategic objectives and tactical demands". Depending on the *framework*, business architecture covers aspects like strategy, products, processes, information, capabilities, knowledge and organization. Good background information on business architecture can be found in ([Zachmann_08], [OMG_08], [Sandoe_01], [Gharajedaghi_06], [Moser_07], [Knittel_06], [Ross_06]).

For the purpose of system architecture the important aspects of business architecture are the following:

- *Information/Semantics*: Systems serve to store and process business information. To implement such systems, precise structure and *semantics* for the business information must be defined ([McComb_04]). Typical tools for this are information models, glossaries and ontologies ([Sowa_00]). Although it would be useful to capture semantics formally for system implementation, current practices are informal text-based definitions. The direct counterpart to this in application architecture is

"The world is changing fast and, for better or worse, *semantics* is at the heart of it"
Dave McComb, 2002

¹Object Management Group.

the *business object model* ([Nicola_02]), formalizing the information model in a more implementable way.

- *Functionality*: The business functionality is provided by the applications. A structured *business functionality map* can be used to analyze the application landscape for functional overlaps, gaps and misalignments. Often the business functionality is linked to the information architecture, describing what information is used by which business functionality. Side-Story 2.2 gives an example for such a map. This is similar to the domain model and the application portfolio, although experience shows that the structure of the application domain model might look quite different from a business functionality map, as the structuring criteria are different. You may want to keep functionality grouped around the same data or being offered in one package together in the same application domain, for example.
- *Processes*: Business uses well-defined processes to provide their services to the clients and to manage the internal processing. With the focus on business *process engineering* and flexible orchestration of services into workflows, understanding the business processes is important. From the business processes we can identify the process steps common to multiple processes. These can then be provided as services for flexible orchestration. Generally, good business process definitions can be found nowadays for well structured operational areas of the business, where they are being used successfully to systematically improve efficiency by continuously capturing process data, analyzing it and improving the process based on the data.
- *Organization and roles*: Organizations are structured and have clearly defined roles which execute their assigned tasks. For many decisions in IT architecture it is important to understand the organization and the roles of people in the organizations. Examples for this include access control systems or the design of portals, where all functionality for a particular role is presented in an integrated way.

Most business architecture frameworks capture the information described above in one or the other form. In the experience of the authors there typically is a *precision gap* between business architecture and IT architecture. While business architecture definition tends to be loose at times, IT architecture must be more precise, so that it can be implemented in systems. Often large enterprises don't have a central owner for all aspects of business architecture across all areas. Therefore the coverage may be patchy and the business architecture is defined implicitly by the behaviour of the people and systems behind a business process. Increased regulatory scrutiny, such as the Sarbanes-Oxley act ([Marchetti_05]) and improved formal optimization methods, such as Lean Six Sigma ([George_04], [George_05]), have lead to more formal documentation of business architecture in recent years.

Most large organizations do not have an explicit function that is responsible for the overall evolution of business architecture. So, in practice, business architecture

to the extent necessary for the underlying IT architecture is often maintained by the IT architecture organization.

The main structural model in *business architecture* is the *business functional map*. As an example, the business functional map – in the form of a business component map – of Credit Suisse is described in Side-Story 2.2.

Side-Story 2.2: Credit Suisse Business Component Map (BCM)

Recently the strong growth of trading activities revealed functional and performance deficits in the Credit Suisse trading and operations application landscape in the Swiss hub. Some of the applications were quite old (more than 20 years) and the extensions required by the very active business could only be implemented with difficulty.

In 2006 Credit Suisse IT management decided to define a new, modern Trading and Operations Target Application Landscape (TOTAL). This alignment was to be based on a *business component map* ([IBM_05a]). Business component map development follows a modeling methodology introduced in 2006 by IBM: *Business Component Modeling* (BCM). In this context, *business components* are the modular building blocks that make up the specialized enterprise. Each component has a business purpose which is the reason for its existence within the organization and conducts a mutually exclusive set of activities (IT-supported, manual, or mixed) to achieve its business purpose. Each business component is based on its own governance model and provides/receives *business services*.

A team consisting of Credit Suisse experts and IBM consultants was formed and worked for 4 months in close cooperation. IBM brought three generic business component maps (for generic retail, private and investment banking) to the table as a starting point. At the end of the definition phase, the Credit Suisse *Business Component Map* (BCM) as shown in Fig. 2.2 was delivered.

Business component modeling – resulting in the business component map as its main deliverable – has proven its value as a methodology to define a functional alignment between business and IT, including a view into the future. After this activity, the planned target architecture for the trading and operations landscape was defined from a functional point of view.

The BCM (Fig. 2.2) has successfully been used to identify functional redundancy, such as overlaps, multiple implementations and to plan a road-map for the implementation of the target architecture. Several major development programs have subsequently been launched at Credit Suisse to transform the current trading and operations application landscape into the desired target architecture.

The BCM methodology has proven to be so powerful that it is now used globally in Credit Suisse in various areas and for several uses.

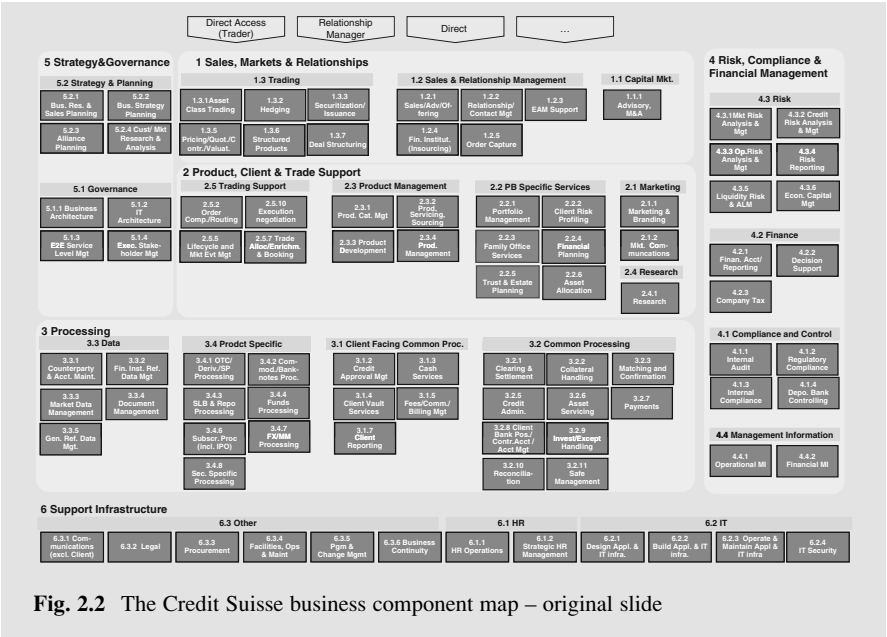


Fig. 2.2 The Credit Suisse business component map – original slide

2.5 Application Architecture

All business applications in the system form the *application landscape*. For the purpose of this book, an application comprises a set of *functionality*, the corresponding *data* and includes *interfaces* to other applications in order to achieve a specific business purpose. Applications consist of one or several software components designed to support business processes and to fulfill specific needs of users. A component consists of programs, data structures, configuration information and associated documentation.

The application landscape is partitioned into *application domains*. Each application belongs to one or more domains. Application domains are *subsets* of the application landscape characterized by a high cohesion from the business point of view. Central to an application domain are its business entities and business functions. Depending on the size of the application landscape one may introduce multiple levels of structure with domains and *subdomains*.

In the experience of the authors, up to two dozen domains can be handled on the top level. Within a domain up to twenty subdomains and 50–100 applications per subdomain can be handled. Experience shows that in a typical application landscape the domain model is not evenly populated, but the two level structure is sufficient to manage

"From the perspective of the underlying science and engineering knowledge base, software is the least well understood and the most problematic element of large scale systems"
Linda Northrop, 2006

landscapes of many thousands of applications, as is typical for very large information systems.

The choice of application domains is an important strategic choice. It goes with the assumption that applications in the same domain can be more tightly coupled than applications in different domains. Typical architectural rules that go with domain borders include that databases may be shared inside a domain, but not across. Sharing data across domains requires that additional quality criteria be imposed on interfaces that are offered for use outside the providing domain. In very large information systems these rules may again be defined on multiple levels.

The choice of the domain model also expresses certain assumptions about the structure of a business. Should the domains be defined regionally, along the process, or aligned to products? Domain model decisions are long-term decisions and have to be taken carefully. The model must be kept stable as a fundamental guiding principle for the managed evolution of the application landscape and the infrastructure. The domain model will over time influence the reality in the application landscape by more closely coupling applications in the same domain.

The domain model decomposes the *functionality* and persistent *data* of the application landscape into smaller, manageable containers. The subdomains contain non-redundant functions and data and together they cover the complete functionality and data universe of the application landscape. This is an ideal view, without taking into consideration the actual functionality of the existing applications. The modeling approach can be seen as first preparing a list of all functions (on a suitable level of granularity, such as “maintain customer address and customer contract information”) and data (again on a suitable level of granularity, such as “customer master data”). Second, all entries in this complete, consistent list of functions and data are then uniquely assigned to the best-suited subdomain.

Assigning all entries in the list to the most appropriate subdomain requires some rules:

1. Strong *cohesion* of the functions and data within a subdomain: Cohesion is a measure of how related functionality and data are gathered together and how well the parts inside a subdomain work as a whole. Cohesion is the glue holding a subdomain together. Weakly cohesive subdomains are a sign of bad decomposition. Each subdomain must have a clearly defined scope and not be a grab bag of unrelated functionality and data ([Spinellis_09]).
2. Low *coupling*: Coupling is a measure of the dependency between subdomains – the amount of “wiring” to and from them. Good partitioning results in little coupling and so the subdomains become less dependent on each other. Obviously, coupling required by the cooperation among applications in one subdomain and applications in other subdomains or the environment are necessary – but unnecessary wiring should be avoided by good subdomain decomposition ([Spinellis_09]).
3. Comprehensive *business-IT alignment*: Because cohesion is mainly rooted in the way business conducts its processes, the business cohesion strongly supports good decomposition. This synergy must be exploited, which automatically leads to a comprehensive business-IT alignment, especially if “business terminology” is used as much as possible in the domain model. A good domain model also

separates dedicated *business* functionality, such as payments or trading from *general* functionality, such as enterprise resource planning, electronic archive, communication channels.

One trap to avoid while constructing a domain model is thinking in products: If a domain model is structured around products, synergies in their production or processing will not be identified. The domain model must become – and remain – the functional and high-level data model of the ideal organization. As such it survives business strategy changes, company reorganizations, mergers and acquisitions and forms a stable, reliable base for the evolution of the application landscape.

Developing a domain model is a highly cooperative, consensus-based effort of business and IT resources under the strong leadership of application architecture. As such, it involves a broad range of experts.

The domain model benefits four areas:

1. *IT architecture*, supporting complexity management, providing the structural foundation for the *enterprise service oriented architecture* (integration architecture), defining the rules for various disentangling programs for *legacy applications* and providing the “ideal”, redundancy-free functional and data *model of the organization*. Once all applications have been assigned to the “best fit” subdomain, redundancy and misfits can easily be identified. A big complexity reduction potential is made accessible!
2. *System evolution*: Enabling independent development, deployment and operation of *components* improves the agility of the system. Better locality and isolation of applications in domains reduces effort for coordination of changes touching many applications. Higher software quality due to reduced cross-component impacts will result. Generation of new, unwanted or unknown *redundancy* is avoided.
3. *IT management*: Leads to improved *transparency* by using domain-based management and reporting and creation of a powerful domain architect organization with defined responsibilities assigned to architectural roles.
4. *Business reasons*: Better understanding of the IT landscape is achieved with domain-model based investment planning, resulting in better coordination and less effort duplication. Complexity reduction leads, as a direct consequence, to shorter time to market and lower development cost for new functionality. Finally, there is an improvement in business-IT alignment.

The domain approach enables the possibility to define *domain-specific architectures* ([Duffy_04]) and *domain specific languages* (DSL, [Kleppe_09]) based on the domain’s business concepts, such as business objects. Together, this leads to *domain specific modeling* (DSM, [Kelly_08]). Domain-specific modeling raises the level of abstraction beyond current modeling approaches and increases application productivity.

“Throughout the history of software development, raising the level of abstraction has been the cause of the largest leaps in developer productivity”
Steven Kelly, 2008

A successful domain model requires the full and consistent implementation of five elements:

1. The *structure* (Fig. 2.4): The main topic of this Side-Story 2.3,
2. A *representation*: Mapping of all domains/subdomains and applications, including their information flows, attributes and properties in an on-line repository,
3. A “*federated architecture organization*”, including the domain architects and other cross-business unit, cross-domain architecture roles (see Sect. 1.9),
4. A set of *processes* for the evolution, maintenance and optimization of the *application landscape* based on domains, including change request management for the domains, management of the domain architects, processes for the periodic domain assessment,
5. Strong and dedicated *governance* supporting and leading the domain architecture and the domain activities.

Side-Story 2.3: Credit Suisse Domain Model with the Main Design Considerations

Between 1995 and 2005 the amount of information and know-how related to the application landscape of Credit Suisse became far too large to be understood and controlled by any single person or even a team. Over the years, solutions were developed without always being correctly aligned with the existing application landscape. This led to unnecessary complexity, like unknown and unwanted redundancy, to silo'd applications, to a loss of conceptual integrity, strong architecture erosion and various other issues.

In addition, starting in 2006 the strategy of Credit Suisse was oriented towards “*OneBank*”, i.e. the global integration and cooperation of all business units (Private banking, retail banking, corporate banking, asset management, investment banking) into one business model under one common brand. This again increased the challenge of managing the combined application landscapes.

This situation called for the development of a *model* of the “target system for the integrated bank (One Bank)”. Such a model could then be used to partition, organize, understand and control the consolidation and the evolution of the very large application landscape. A central architecture team was mandated to develop the model – this was the start of the Credit Suisse *Combined Domain Model* (CDM). The combined domain model relied on an earlier application domain model developed and used in Credit Suisse private banking. Part of the organization therefore was familiar with domain models and had experienced their value for the management of the application landscape.

Due to the large size and the many different stakeholders, the development

“The key to controlling complexity is a good domain model”
Eric Evans, 2004

of the domain model was started with an enterprise model. This enterprise model is

shown in Fig. 2.3: It consists of seven categories. Category 1 “Partners and Persons” covers all the parties with which the bank is exchanging services or information. This is done via category 5 “Communication and Collaboration”. The categories 2 “Finance, Investment & Sales”, 3 “Trading and Markets” and 4 “Cash and Asset Operations” are the “production lines” of the bank – here the products and services are “manufactured”. Categories 6 “Accounting, Controlling and Reporting” and 7 “Enterprise Common Services” cover enterprise support functions, such as human resource management, compliance to all legal and regulatory requirements, enterprise content management etc. Note that the business context of this domain model is *banking* – other industries will have different domain models.

Once the enterprise model (Fig. 2.3) was agreed, a high-level decomposition of the functionality and data was laid down. The next step was to define and assign *domains*.

Again, the collaborative effort of business and IT resources – this time on a detailed level – was required. The current domain model of Credit Suisse is shown in Fig. 2.4: It contains 22 domains (and covers the full business

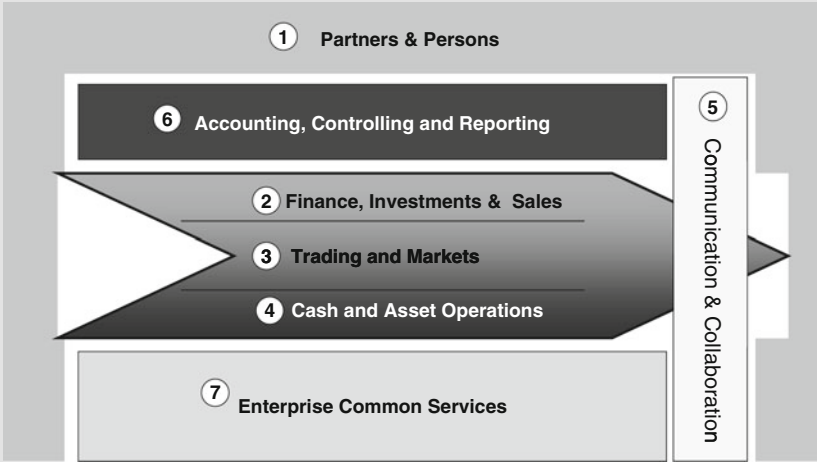


Fig. 2.3 Categories of the Credit Suisse 2009 domain model

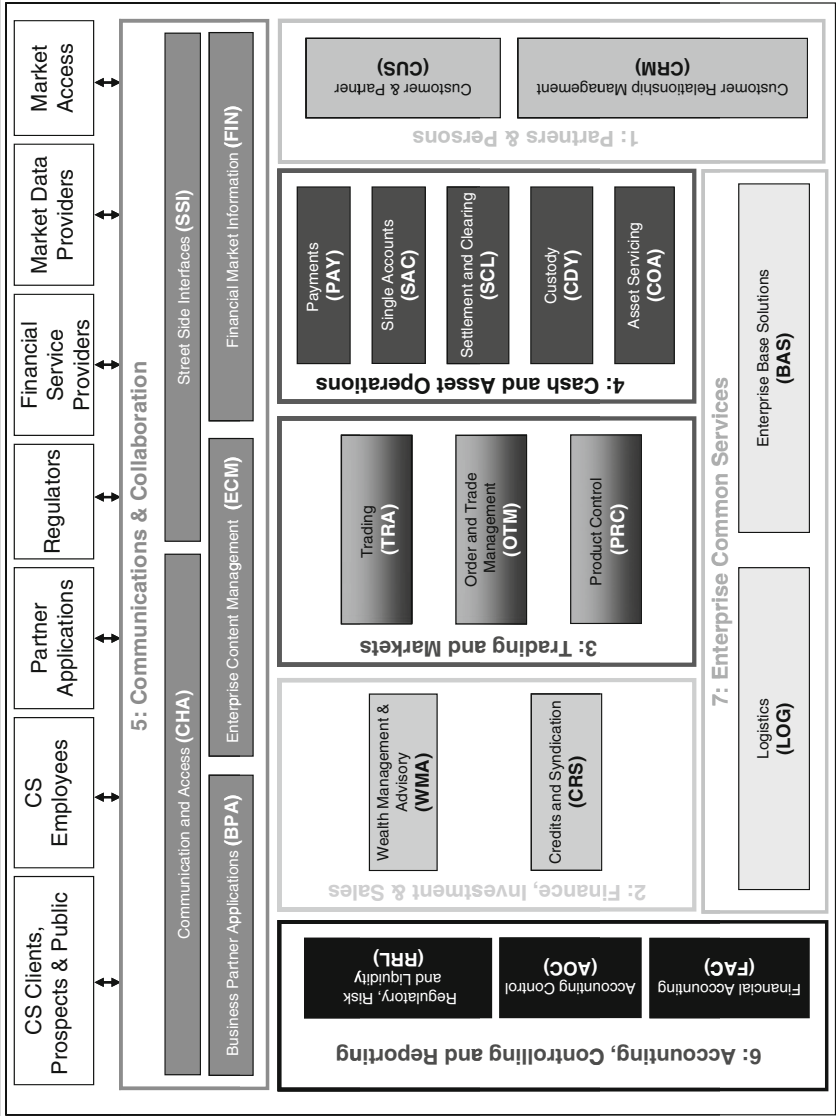


Fig. 2.4 The Credit Suisse 2009 domain model

offering of Credit Suisse worldwide). The domain model has proven to be an invaluable tool for a number of processes.

The domain model became the key structural ordering element: Most applications were assigned to subdomains. An inventory and planning tool was introduced which listed all applications, together with a large number of properties and attributes of the individual applications and the *information flows* between the applications.

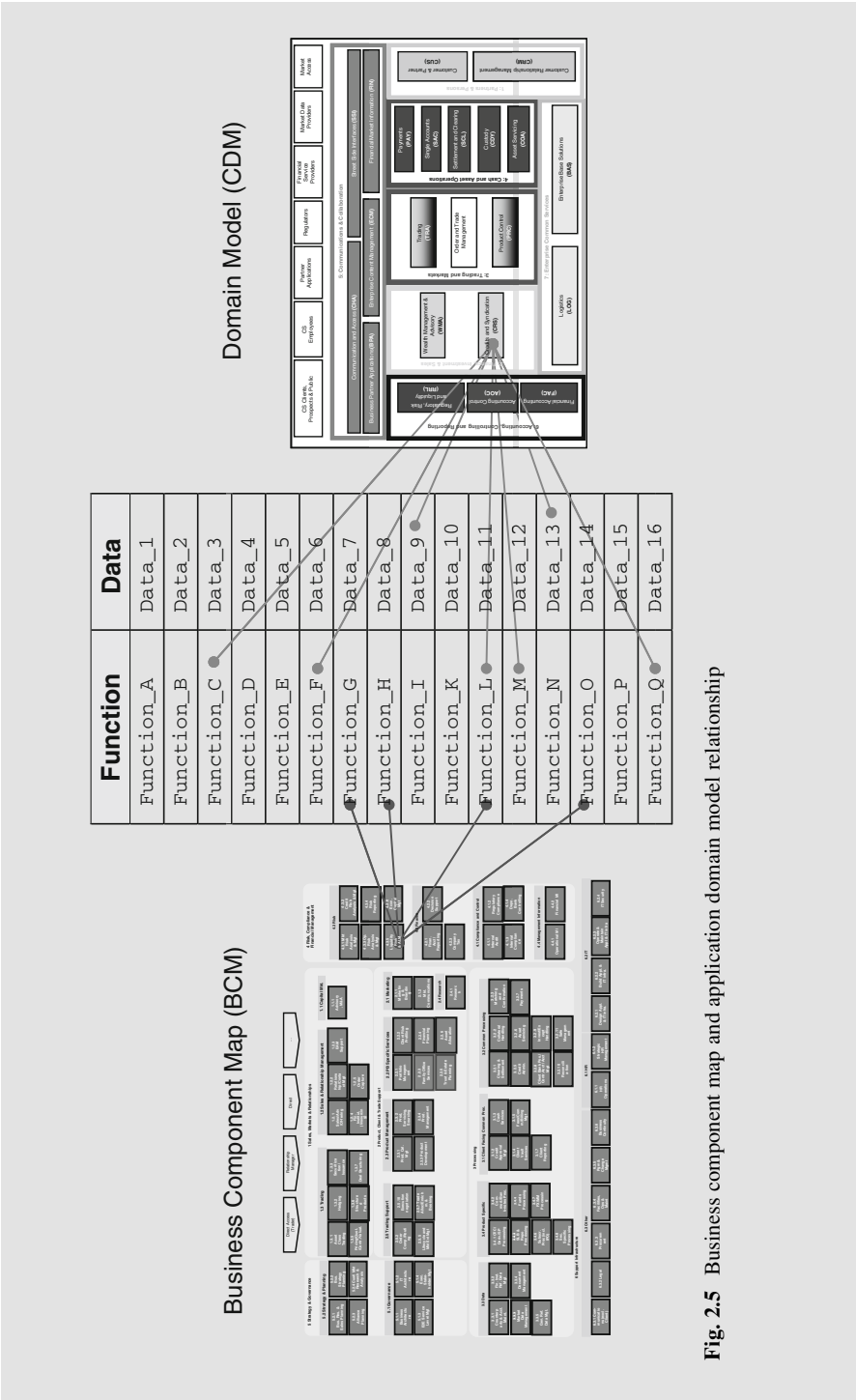
The *domain model* (see Side-Story 2.3) is linked to the *business component map* as presented in Side-Story 2.2. They are, however, not the same. Side-Story 2.4 explains their relationship.

Side-Story 2.4. Relationship Between CDM and BCM

In the middle of Fig. 2.5 a list of functions and data is shown: This list contains the *complete functionality* and all the *data* required to operate the business. The list is unstructured. There is only one such list: The list is complete, free of redundancy and consistent – it represents the “atomic” breakdown of the business to be supported by the system. An *atomic business function* is the lowest level of functionality in a system that is still recognizable to the business (see [Sessions_09], [Sessions_08]). Note that all the functions and data are included, not only the IT-supported functions and data, but also the functions executed manually and the data used in manual processes.

On the left side of Fig. 2.5 the *BCM (Business Component Map)* is presented. The BCM relates to the list of functions via a mapping of individual functions into the business components. The mapping rules are according to the business view. The aggregation into business components is optimized for reuse in business processes. It is therefore possible that some basic functionality or data required to operate the system (such as reference data, access protection functions, etc.) is not mapped to the BCM, but only to the domain model.

On the right side of Fig. 2.5 the *CDM (Combined Domain Model)* is drawn. The CDM relates to the list of functions and data via a mapping of individual functions and data elements into the domains. The mapping rules are according to optimal IT implementation. All functions and data are uniquely assigned. Following the principle of functional and data business cohesion this partitioning provides the foundation for the optimized, effective and efficient IT implementation of the business functionality and data.



Business component maps and domain models are *structural models*. Structural models are necessary, but not sufficient to model an information system. In a very large information system with many stakeholders and a distributed system development activity the *semantics* – the exact *meaning* ([Vygotsky_02], [Portner_05]) of all concepts, terms and expressions used in the system – must be defined ([McComb_04]) so that all architects, programmers and users of the system rely on a common semantic definition of the shared objects. Semantic ambiguity is one of the prime reasons for redundancy in systems. The same concept is implemented slightly differently in different domains. Unclear semantics lead to integration problems because two sides of an interface do not share the understanding of data fields. They may, for example, assume different units of measurement. This creates the need for *semantic models*.

Semantics can be modeled on different levels of consistency. The simplest form is a *glossary* of terms, including allowable values, units etc. Often used semantic definition instrument are *taxonomies*, which specify the hierarchy and the meaning of terms used in an enterprise ([Stewart_08]). A *business object model* defines semantics on an even higher level of consistency. All business entities are modeled as objects, including their definitions, attributes,

“As people start forming communities and attacking a problem, their approaches to the solution will vary. As these groups separate, they must develop more detailed communication within each subgroup, and the language diverges. New words aren't usually invented, rather new meanings are imposed on the words and phrases already being used”
Dave McComb, 2004

operations and relationships ([Daum_03a], [Lhotka_08]). The richest level of specification is *ontologies* ([Fensel_07], [Allemang_08], [McComb_04]). An ontology is a logical model, defining all the objects, attributes and relationships in a formal, machine-readable way. Ontologies – especially based on the standardized Web Ontology Language OWL – form the base of the semantic web and thus of future, semantically interoperable business systems ([Dietz_06], [Lacy_05], [Stuckenschmidt_05], [Haase_06], Financial Ontology: [Montes_05]). The advantage of ontologies is that they can be fully checked for internal consistency.

As an example, a small extract of the Credit Suisse business object model (BOM) is presented in Side-Story 2.5. The business object model is an important instrument of application architecture.

Side-Story 2.5: Credit Suisse Business Object Model

Business object models (BOMs), [Daum_03a], [Lhotka_08]) are highly industry-specific models of the business concepts, their properties and their relationships. Developing a sound, useful business object model for a very large information system is a major endeavor which requires the massive collaboration of both business knowledge and IT knowledge resources.

A business object model captures the essence of a business in a formal way and forms the basis for consistent IT implementation.

"A sound conceptual model documents that the participating analysts and engineers have understood the problem"

Berthold Daum, 2003

In order to keep the BOM manageable², Cre-dit Suisse used three levels of *aggregation* and three levels of *abstraction* (Fig. 2.6).

The *aggregation levels* (from top to bottom) are “enterprise level”, “domain level” and “component level”. Enterprise level business objects – labeled eBOs – are at the top level. eBOs are valid for the complete enterprise. The eBOs are refined by top-down transformation rules to the domain level (see Side-Story 2.3), where the domain business objects dBOs are maintained. The domain business object model dBOM is the core knowledge base of banking concepts and their relationships. The lowest level of aggregation is the “component business objects”, i.e. the cBOs. cBOs are modeled when individual components are defined. cBOs are then used as the basis for implementation.

The *abstraction levels* are “conceptual”, “logical” and “physical”. The BOM only handles the conceptual levels. The lower abstraction levels are deduced and refined from the conceptual models.

Figure 2.7 shows the enterprise level BOM of Credit Suisse: All the key concepts are captured and represented. All the dBOs – on the domain level – are refined from the eBOs by adhering to clearly specified refinement rules that are automatically checked by a model checker.

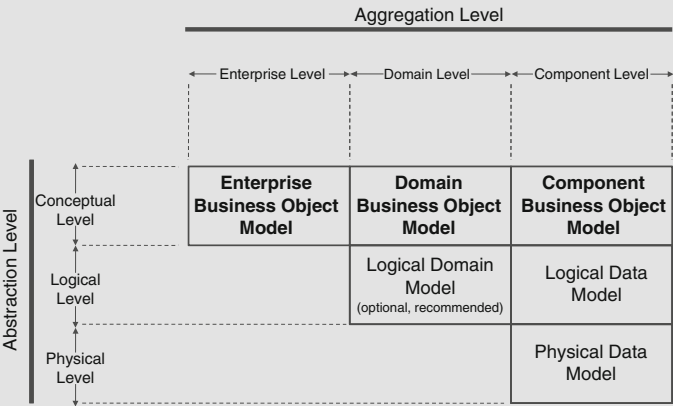


Fig. 2.6 BOM abstraction and aggregation levels in Credit Suisse

²The Credit Suisse BOM contains 21 Enterprise Business Objects (eBOs), in the order of 500 Domain Business Objects (dBOs) and several thousand Component Business Objects (cBOs).

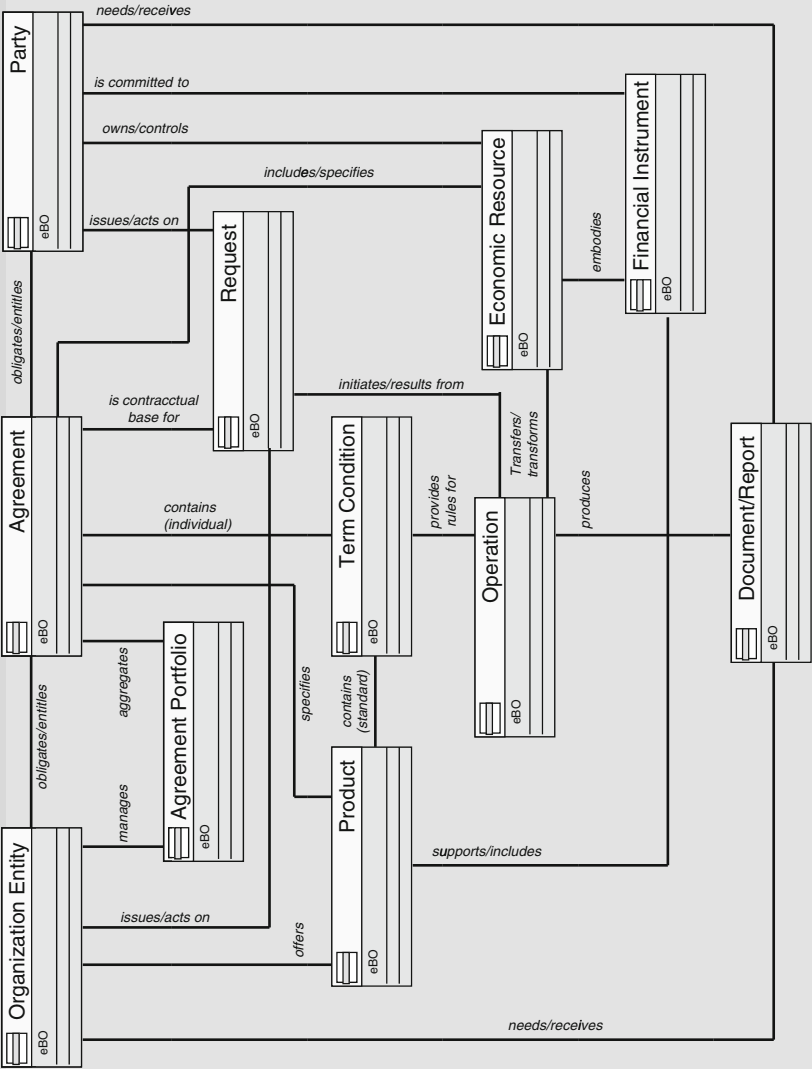


Fig. 2.7 Credit Suisse enterprise level business object model

A condensed specification of the eBOs is given in Table 2.2. The full specification of eBOs contains the properties (attributes), the associations and the BO management and versioning information.

Table 2.2 Description of Credit Suisse enterprise level business objects

Enterprise level business object eBO	Description
Organization Entity	An OrganizationEntity is any unit within Credit Suisse. These units may be ordered in a legal hierarchy, according to a line management hierarchy (or matrix) or in an organization chart
Party	Any internal or external entity with which an OrganizationEntity exchanges information, documents, goods or services based on an Agreement or a relationship. The entity can be a physical or legal person, a corporation, a government unit, a group of persons and any combination of these
Agreement	An explicit or implicit contract between two or many Parties, specifying rights, obligations and responsibilities of all Parties involved. Agreements have either a predefined validity in time (e.g. a credit duration) or have an indefinite temporal validity (e.g. a cash account)
AgreementPortfolio	The set of Agreements for which the bank has a responsibility based on this AgreementPortfolio to supervise and/or manage risk, investments, performance or other metrics and to report them to the stakeholders and authorities
FinancialInstrument	A FinancialInstrument is a standardized trading or exchange vehicle that defines rights and obligations of a Party using the FinancialInstrument. The value and price of a FinancialInstrument can be obtained from market makers (Party)
Product	A banking service or banking product offered and delivered by an OrganizationEntity to one or several Parties
TermCondition	Standard terms (e.g. prices or rates) and conditions (e.g. credit usage) for Products and specific details of individual Agreements
Request	A Request is any trigger (demand) to execute an action (Operation), exchange information (DocumentReport) or generate, modify or terminate an Agreement
Operation	Transaction performed generally in the context of an Agreement triggered by the arrival of a Request (e.g. order) and resulting in the transfer or exchange of EconomicResources, exchange or modification of information, or change of Agreement
EconomicResource	An EconomicResource is a value under the control of a Party. EconomicResources have measurable properties that can be used by valuation methods to determine their value, e.g. their monetary value
Document/Report	A DocumentReport is a container for information of any form (electronic, paper, message etc.) and of any content (text, figures, pictures, sound, film etc.). A DocumentReport can either be raw information or can be an assembly or aggregation of information from various sources and/or DocumentReports compiled and presented according to a defined set of rules

The “working level” of the BOM is the *conceptual domain level*. Each domain refines and specifies its key business concepts, attributes, associations and operations in their domain business object model dBOM.

As an example, the dBOM of the domain Customer and Partners (“CUS”, see Side-Story 2.3) is shown in Fig. 2.8 (without associations). Any project building a solution in the domain CUS – or using domain business objects from the domain CUS – has to start with the CUS domain objects and refine (enrich) them while constructing the component business objects cBOs.

One of the most demanding tasks related to the business object model of a very large information system is the *model consistency*. If the model is not consistent, it may cause more harm than benefit! The hierarchical consistency is assured by a clearly specified set of rules for how a dBO is enriched from an eBO and how a cBO is enriched from a dBO, including the refinement of associations, attributes and operations. Overall consistency as an example assures uniqueness in the namespace for all elements of the model. In a very large information system, business object model consistency can only be assured by a common repository and an automated model checker.

2.6 Technical Architecture

Technical architecture is the underlying layer of infrastructure comprising all elements of IT infrastructure such as hardware, system software, databases, networking components, system management software, database management systems and middleware components. Typically, this layer is not fundamentally dependent on the purpose the system is used for. We call the set of technology components used in this architecture layer the *technology portfolio*. Most of the elements in the technology portfolio can be found underneath any kind of very large information system. There is a gray zone between applications and infrastructure, which is commonly, but not very precisely, called *middleware*. These are technology components that can be considered part of the application or infrastructure in its own right. Depending on the maturity of the organization and the kind of applications on top of the infrastructure, this boundary may be chosen at different levels ([Masak_07], [Woods_03], [Britton_01]). Efficient organizations generally try to move this boundary upwards by standardizing more and more of the middleware and providing it as a set of common services to the applications. It is also a historic development in computing that ever more powerful abstractions have been developed and used over time. Examples include operating systems, virtual machines, databases and transaction monitors. More about the concepts and principles on how to manage the technology portfolio can be found in Chap. 4 (Infrastructure).

The infrastructure layer introduced in Fig. 2.1 is organized as *technical domains*: The technical domains form the basis for managing the *technology portfolio*. In the federated architecture organization each technical domain is managed by a *technical domain architect*. It is the role of the domain architect to define standards and

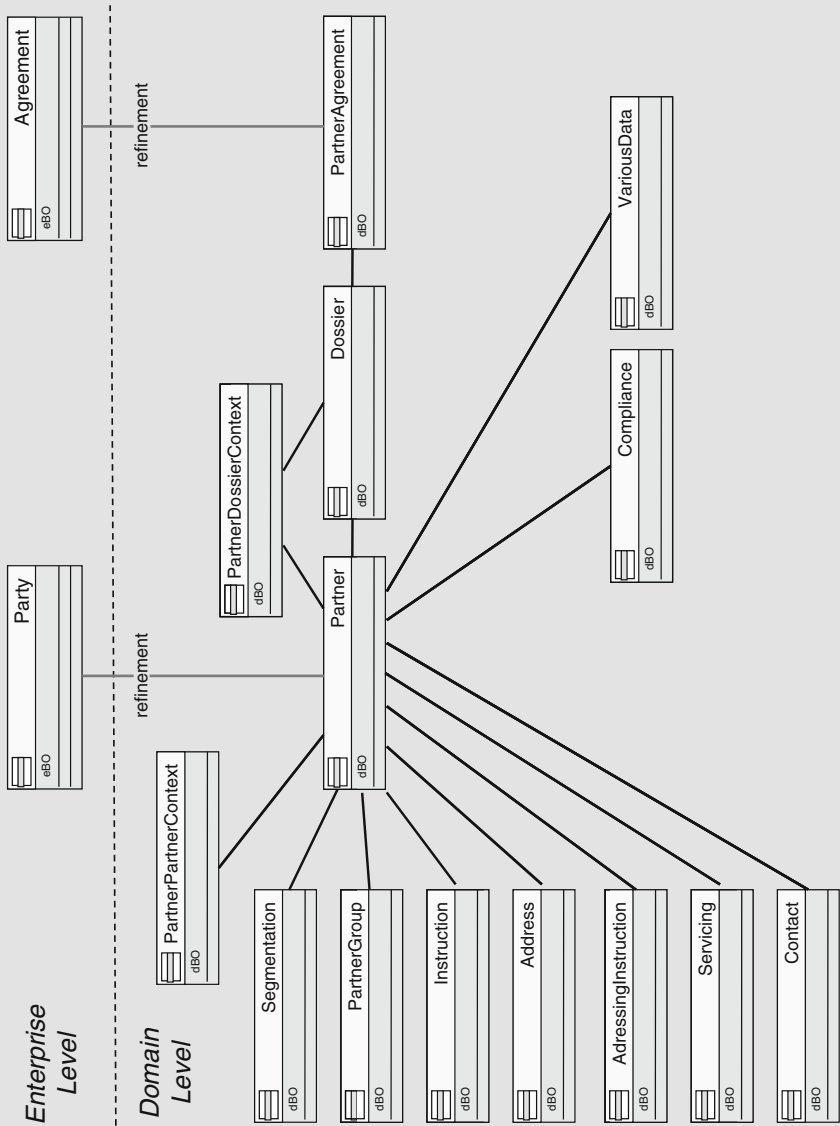


Fig. 2.8 (Side-Story-25) Business object model of the domain "customer and partner"

roadmaps for all the technical components and platforms in his or her domain. Careful, continuous and predictable management of the technology portfolio is fundamental for managed evolution. Replacement of a substantial part of the technology portfolio within a short timeframe is typically impossible due to the application adoption effort. Porting and retesting the large number of applications relying on a particular technical component in very large information systems is just too expensive. If the change in the technology portfolio is gradual and predictable, the applications can adapt within their natural lifecycles paced by the changing business needs.

The choice of technical domains is an important architectural decision reflecting the market segmentation for infrastructure products. A proven technical domain model is shown in Fig. 2.9. This technical domain model contains 11 *technical domains*:

- 1. *Platforms*: The platforms are specifically defined, engineered and implemented to present the infrastructure services to the applications. Platforms integrate software and hardware components and processes;
- 2. *Application Development Environments*: Tools and processes needed for the modelling, development, test, integration and documentation of applications;
- 3. *Integration Software*: Technical components for the communication between applications. Often this is known under the term middleware in the market;

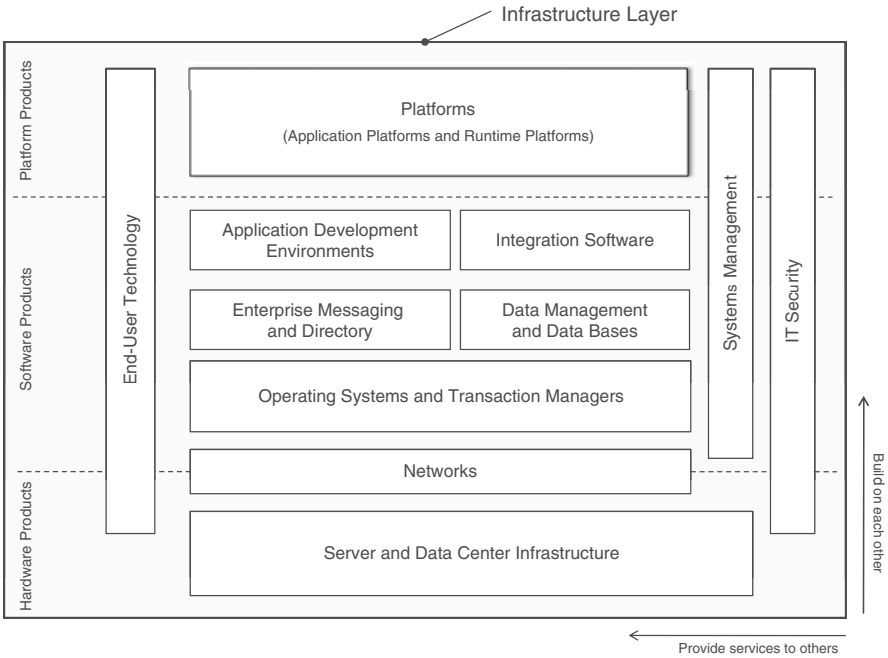


Fig. 2.9 Technical domain model

4. *Enterprise Messaging and Directory*: Enterprise infrastructure for employee and customer communication, including e-Mail, directory services, instant messaging;
5. *Data Management and Data Bases*: Databases and their management systems (e.g. database server software, database client drivers), database development tools (e.g. Schema modelling tools);
6. *Operating Systems and Transaction Managers*: Server operating systems, server virtualization, web and application servers (e.g. JEE container), transaction managers;
7. *Systems Management*: Systems and software required for monitoring, management, performance analysis and operating of the very large information system, such as capacity and performance management, configuration and inventory management, incident management;
8. *IT Security*: All the systems and software for the protection of the assets, such as security administration, data encryption, Internet boundary protection;
9. *End User Technology*: Tools provided to the end user to improve personal productivity, such as personal computers, office applications, collaboration technologies, mobile access and printing;
10. *Networks*: All the products to connect systems on the data transmission level, both within the information system and also across the boundaries, such as to the Internet;
11. *Server and Data Center Infrastructure*: Data center hardware and physical infrastructure, including emergency back up power, cabling, cooling, server hardware and storage facilities.

The technical domains (Fig. 2.9) are organized in three layers: “Hardware products”, “software products” and “platform products”. Products in higher layers build on top of products in the lower layers. Platforms form the highest layer, providing integrated infrastructure services to the applications. Some technical domains cover more than one layer because software and hardware are closely linked to each other. The technical domain *end-user technology* is such an example, because it contains all the necessary hardware and software products to deliver the end-user services and is in itself a platform.

2.7 Vertical Architectures

Some architecture aspects relate at the same time to business, application and technical architecture and have to be managed across all layers on the level of the entire global system. Such architectures are termed *vertical architectures* (see Fig. 2.10).

Integration architecture is an important prerequisite for the managed evolution. The integration architecture (see Chap. 3) consists of principles, processes and technical solutions for managing the distribution and heterogeneity of the application

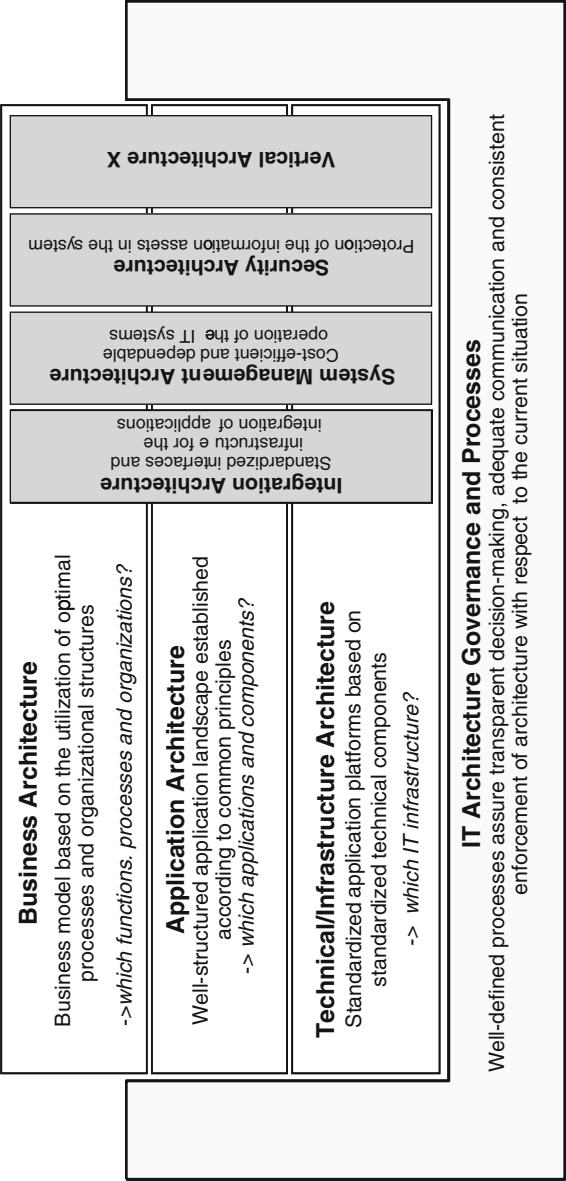


Fig. 2.10 Vertical architecture areas

landscape and the underlying technologies. Most importantly it defines the concept of interfaces, a process to manage interfaces and technologies to implement interfaces across heterogeneous technology platforms and application domains. This is particularly important for the managed evolution, since under this concept only parts of functionality or technology of the entire system can be replaced. So managing the interfaces among these parts, allowing for an independent lifecycle for each part is fundamental for the success of managed evolution of very large information systems. This is recognized by most architects of large systems under the term “Service-Oriented Architecture” (SOA), see ([Erl_04], [Masak_07], [Woods_03], [McGovern_06], [Chappell_04], [Woods_06]), although the contemporary SOA-discussion is still too much focused on the technology rather than *interface management*. It is really crucial to see integration architecture as a vertical architecture that overlaps all three architecture layers in the following ways:

- It overlaps with *business architecture* where services should be integrated in a flexible way to easily create new views on business objects or be orchestrated into new implementations of business processes,
- It overlaps with *application architecture* where the syntax and semantics of the interfaces and general principles of where and how to interface are defined,
- It overlaps with *technical architecture* where it defines the underlying integration technologies, such as middleware, and methods.

Other vertical architectures depend strongly on the context of the system. In the context of banking, where confidentiality of data and flawless operation without interruptions or data losses are mandatory for success, *security architecture* and *systems management architecture* have proven to be essential vertical architectures (Fig. 2.10). One can imagine, however, application areas like numeric simulation systems to define performance architecture or military systems to define resilience architecture. More generally, vertical architectures represent important non-functional aspects of very large information systems which are of universal impact across the whole system.

As a more detailed example, the Credit Suisse security architecture is described in Side-Story 2.6.

Side-Story 2.6. Credit Suisse Security Architecture

Security is an important property of a system of a financial institution. Customer data and client transactions must be well protected against unauthorized access. Because a large percentage of transactions are executed electronically in today’s modern banking and at the same time attacks from the outside become more and more frequent and dangerous dependable security architecture is mandatory.

Today the security of systems is a well studied and fast progressing field. A great number of security concepts, mechanisms, tools and methods exist. To assure sufficient security in a very large information system, a suitable, consistent and complete set of security measures must be defined, implemented, enforced and controlled: An important means to do so is the *security architecture*.

"Being able to detect, contain and eradicate security incidents is in many respects equivalent to defusing explosives - the sooner and better you do it, the less impact a security-related incident is likely to have"
Linda McCarthy, 2003

Security architecture ([Killmeyer_06]) is not described as objects and relationships: Security architecture consists of objectives, concepts, standards and security services that must be implemented throughout the complete system – hence the name “vertical architecture”. Vertical architectures – especially security architecture – are not static architectures: The range and scope of threats is ever-increasing, including more sophisticated cyber-crime (see e.g. [Deloitte_10], [CERT_04]). Therefore, the Credit Suisse security architecture and the protection measures are continuously improved and adapted. This is in the responsibility of a global group of security architects, directly reporting to the Chief Architect.

The structure of the Credit Suisse security architecture is shown in Fig. 2.11: It consists of three horizontal *functional layers* – the transportation layer, the transformation layer and the manipulation layer. A typical transaction starts in the manipulation layer in one part of the system, is propagated down through the transformation layer, transported via the transportation layer and arrives in another part of the system where it runs up through the

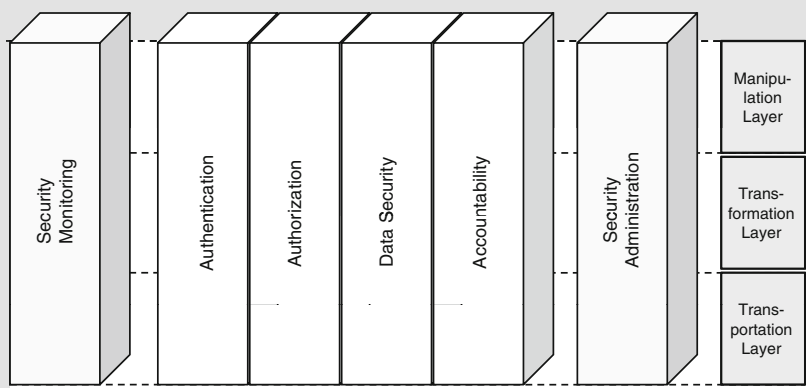


Fig. 2.11 Credit Suisse security architecture

transformation layer and is consumed and processed in the manipulation layer. Each layer is subject to various possible attacks.

The security architecture defines six *security technology towers* (Fig. 2.11): Four of them are *security mechanisms*, i.e. authentication, authorization, data security and accountability. The last two are *security controls*, i.e. security administration and security monitoring (see [Proctor_02], [Umar_04], [McCarthy_03] for details). The individual security technology towers provide the concepts, standards, processes, technologies and services for:

Authentication: Reliable identification and verification of the identity of system users, system components (servers and applications) and external partner systems,

Authorization: Granting (or denying) access of users and system components to resources based on the verified identity of a requestor and relying on explicit access control policies;

Data Security: Protection of data stored in the system or transmitted between systems against any unauthorized access or eavesdropping;

Accountability: Generation of a complete audit trail for all security-relevant transactions, including attacks and incidents;

Security Monitoring: Continuous supervision of the system activities related to security, including detection of real or attempted security breaches, such as intrusions and unauthorized accesses;

Security Administration: Management of identity credentials for authentication and access rights for authorization for all users, systems and resources.

For each functional layer the respective security technology towers specify security components. The security components must strictly be implemented in all parts of the Credit Suisse very large information system. Security is the subject of regular assessments by internal IT risk and audit, and is periodically scrutinized by independent external experts.

The impact of vertical security architecture on managed evolution is twofold: First, the weakest point in the very large information system determines to a large extent the overall security level of the system. It is therefore mandatory, to assure and maintain a consistent level of security throughout the complete very large information system, thus preventing security loopholes. Strong vertical security architecture allows the definition, implementation, checking and auditing of the security mechanisms implemented in the system. Second, individual applications and projects take advantage of the deployed security infrastructure and do not have to spend repeated effort and time on “reinventing security solutions”.

2.8 Architecture as an Activity: The Architecture Process

One view of IT architecture is to see it as one of the *management processes* governing a very large information system. The purpose of that process is defining a target architecture and to steer the evolution of the system towards this target architecture. As shown in Fig. 2.12 the *architecture process* can be separated into four *subprocesses*:

- 1. Architecture development
- 2. Architecture communication
- 3. Architecture implementation
- 4. Architecture controlling.

The architecture *definition* process results in ratified architecture standards, models, target architectures and guidelines, which are mandatory for all development work within scope. The process starts with studies, prototypes and pilot projects. Pilot projects apply non-standard technology in production applications in order to understand general applicability. The evaluation result is subsequently used for decision making. In the ideal IT organization described in Chap. 6, there are two areas where architecture decisions are made: In the architecture function within the application development units and in the infrastructure architecture

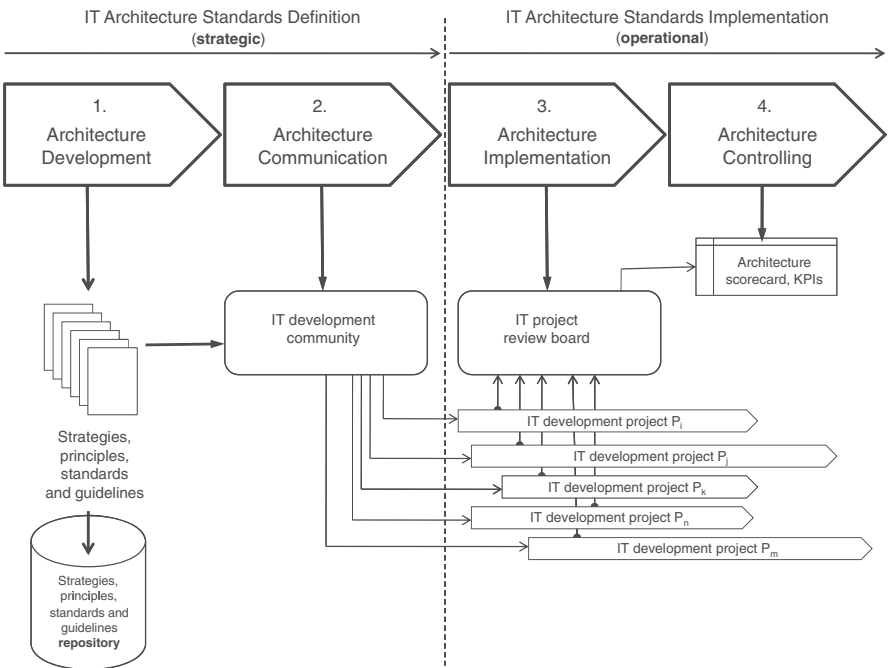


Fig. 2.12 Architecture process

function. These groups should have in place standing steering committees with representation across the architecture communities within their units. These steering committees are responsible for reviewing and approving all major architect decisions regarding strategies, programs, etc.

In addition, an IT-wide architecture steering committee led by the Chief Architect should be established to ratify the major decisions of the underlying committees. This IT-wide committee is also responsible for establishing overall strategy and many of the overarching elements of architecture management described in this book.

It is important to reinforce the binding nature of the decisions of these bodies by having well-defined membership and charters, regular scheduled meetings and published documentation and minutes. Where appropriate, the decisions should be documented as official standards for the IT organization.

The most important architecture strategies and decisions should be taken to the IT management committee for discussion and decision. For this to be effective, the Chief Architect should establish as part of the architecture planning cycle an agenda of key decisions required to support the managed evolution and guide these decisions through the governance process so they can be properly positioned for success with his or her peers.

The efficient operation of these steering bodies requires investment in syndication of the proposed decisions with the relevant committee members and other influential stakeholders prior to the meeting. In our experience, without such syndication and the integration of the resulting feedback into the proposal, many decisions will be sent back for further work or be rejected altogether.

As architecture work starts with every single developer, *architecture communication* is a very important first step to steer the system towards the target architecture.

"Critical structural decisions are often made on the side, without executing proper control - for example, an engineer might quickly write a small batch program to synchronize data between two systems, and 10 years later, a small army of developers is needed to deal with the consequences of this ad-hoc decision"
Dirk Krafzig, 2005

Architecture communication starts with a well structured *architecture documentation* that documents all the standards, guidelines, target architectures and the like. It is crucial that multiple targeted views of the same set of standards exist for different roles in the development process.

The Java software developer will generally not be interested in testing standards for mainframe software developers and vice versa. Architecture needs to be communicated over multiple channels including documentation on the web, formal developer education, question and answer sessions with the architects and management leadership by the senior architects. It is important that communication is not a one-way channel, as widespread acceptance in the community is paramount to success. The general experience shows that resources invested into good communication pay back several times in the architecture implementation process.

As an example the Credit Suisse architecture communication concept is presented in Side-Story [2.7](#).

Side-Story 2.7. Credit Suisse Architecture Communication Process

Architecture knowledge has become quite extensive in Credit Suisse. Much of it is documented in the form of binding IT standards. One key requirement is to make this knowledge available to the community in a suitable form. This is the objective of the *architecture communication process*. At Credit Suisse architecture communication happens through a number of different channels, such as classroom learning, consulting in projects, regular brush-ups, written communication and up-to-date webpages.

Classroom learning starts with the *architecture boot camp*: Each new hire entering a Credit Suisse Private Banking development department goes through a 2-day boot camp where the basics of Credit Suisse IT architecture are presented. Another instrument is *Fit-for-Architecture*, a full-day course mandatory to all solution architects at periodic intervals. In fit-for-architecture, new developments in architecture and IT standards are communicated. More than 20 architecture events in the form of auditorium and video broadcast lectures to a larger audience presenting new developments or emerging concepts are carried out each year.

Written communication includes the IT standards. In addition, some periodic publications, such as *InsideIT*, *Transfer* and an *architecture newsletter* are available. These publications are comprehensive and address a large audience.

Last but not least, architects from central architecture and domain architects are involved as consultants in every important project, bringing their expertise into the project at an early stage and accompanying the projects until their completion.

Formal project reviews by central architecture (see Table 2.3) are also used as a communication channel: Any architecture deficiency found during a review is discussed with the project architects, thus educating them on the job.

The *architecture implementation process* ensures that the standards and guidelines are adhered to and that exceptions to standards are managed at the appropriate level. If we presume that the system is changed by the portfolio of projects, this is best ensured by *reviewing* all projects at well-defined points in the project lifecycle with regard to their adherence to architecture standards, the quality of their design and the fit into the remainder of the application landscape. *Project reviews* are executed according to a fixed procedure and are based on formal review sheets. Side-Story 2.8 shows part of the architecture review check list used in the project reviews by Credit Suisse. In addition to that, senior architects are appointed as solution architects for critical projects, where substantial architecture impact is to be expected.

The other element of the architecture implementation process is the design and the sponsoring of *architecture programs*. Architecture programs are collections of IT architecture-driven projects that implement infrastructure in support of the target architecture, restructure applications that are no longer structurally fit for extensions and help retire outdated technology solutions and applications.

Side-Story 2.8. Credit Suisse Check List for Architecture Reviews

Every project of a certain size (measured by its development cost) or of architectural importance is reviewed four times according to the Credit Suisse project methodology by a team of IT architecture reviewers. Credit Suisse Private Banking uses two project methodologies: The classical *waterfall model* and the Rational Unified Process (RUP). For each project, the adequate methodology is chosen.

As an example, the waterfall model reviews correspond to the four project phases:

- *PC*: Project concept. The initial project concept is described in a document. It contains the basic project justification, the main project objectives and the project setup.
- *PO*: Project offer. The planned implementation – possibly including some options – is described in a document. Risk analysis and project planning is presented.
- *RO*: Realization offer. The project documents its activities, the solution architecture, the interaction with other applications and projects and the impact on the application landscape. The project planning and risk analysis is refined.
- *RC*: Request for conclusion. The project reports on the achievements, the open issues and the recommendations for the future.

The review is conducted by 2–5 reviewers based on an IT architecture project review checklist. Table 2.3 shows a short check list extract covering general questions, setup and banking applications. The full checklist covers all systems management, infrastructure, etc. Reviewers agree on common findings, resulting in a consolidated review report being communicated to the project. The conclusion of the review report is either “OK” which means that the project can continue as proposed, or “OKA” (“Auflagen” in German, hence “OKA”), which means that the project has to accept obligations, or “NOK” (Not OK) meaning that the project cannot enter the next phase before the raised issues are resolved and reviewed again.

The most important part of the review report is the *obligations*: Here the reviewers express requirements for architectural integrity of the modifications done by the project to the application landscape. Any obligation entered by the reviewers is recorded in a centralized obligation management system and is tracked. The mechanism of associating obligations by architecture reviewers to projects is important for managed evolution: It assures that each project is contributing to – or at least not damaging – the architectural integrity of the application landscape and thus improves the agility of the system (as shown in Fig. 1.7.).

Table 2.3 (Side-Story 2.8) Credit Suisse IT architecture review checklist (Extract)

IT Architecture Project Review Board of the 27.05.2009									
	OK	OKA	NOT OK	Reasons for NOK					
Evaluation Architecture		X							
Architecture Reviewers	Date	Findings	Condition	Deadline					
Hans Muster Peter Beispiel Jürg Modell	19.05.2009	The proposed data migration concept leads to unmanaged data redundancy	Propose a new data migration concept which completely eliminates data redundancy	PO					
Exceptions, accepted deviation from standards									
none									
Conditions of Previous Reviews			Y	N					
Have the conditions of the previous review(s) been met?									
If conditions have not been met, discuss further actions with KSCD									
Review Details									
Part 1: Sufficient/adequate documentation of architecture relevant requirements/descriptions									
Nr.	PC	PO	RO	RC	Legend: Y = Yes / N = No / I = Irrelevant	Y	N	I	Comment / Statement
V01	X	X			Has the relevance for the strategy been described <ul style="list-style-type: none"> Overall strategy/IT strategy Strategy of domain/area 				
V02	(X)	X	X		Have the functional requirements been described in a way that allows to derive and assess the implications on the IT-architecture and design?				
V03	(X)	X	X		Are the non-functional requirements described in a way that allows to derive and evaluate implications for the IT architecture and design .				
V04	(X)	X	X		Is the architecture relevant information available and are the relevant architecture decisions documented, e.g. concerning <ul style="list-style-type: none"> Available options (PO) Integration into overall system Interfaces to other projects/domains/applications Application and technical solution Security concept, risks Systems management, production Phase out of 'obsolete' architecture (e.g. technical or application components) 				
V05		(X)	X		Entries in Information System and Dictionaries: <ul style="list-style-type: none"> Application parameters entered in the applications inventory? Portfolio of technical standards (TIA) updated? 				

(continued)

Table 2.3 (Side-Story 2.8) (continued)**Part 2: General setup, integration/delineation**

Nr.	PC	PO	RO	Legend: Y = Yes / N = No / I = Irrelevant	Y	N	I	Comment / Statement
G02	X	X	(X)	Integration into the overall system/ avoidance of redundancies <ul style="list-style-type: none"> Are the boundaries/interactions with other projects, processes, domains, applications and infrastructures clear and appropriate? Are potential redundancies, overlaps e.g. concerning infrastructures, services reasonable? Justified? Accepted? Is there a mix of old and new architecture? Reasonable? Justified? 				
G03		X	X	Migration to standard architecture / phase out of obsolete architecture/standards: <ul style="list-style-type: none"> Are the necessary actions for a migration to standard architecture planned, described and appropriate? Is it documented how/when old architecture will be phased out? 				
G04	(X)	X	(X)	Options: <ul style="list-style-type: none"> Are the proposed options appropriate and complete? Is the proposed option reasonable? 				
G05	(X)	X	X	Risks: <ul style="list-style-type: none"> Have all risks relevant for architecture been identified? Have they been mitigated accordingly? 				
etc.								

The fourth architecture process – *architecture controlling* – measures progress towards the target architecture. Typically this process is implemented as a balanced scorecard ([Kaplan_96], [Niven_06], [Kaplan_06]) of key performance indicators measuring various indicators (see Side-Story 7.3), including the following:

1. *Standard deviation indicators* measure on the one hand, whether the exception management process is sound and, on the other hand, whether the standards are appropriate to fulfil a majority of the needs.
2. *Strategy adoption indicators* measure how new architecture elements are being accepted in the projects. This is particularly important if the strategy requires the retirement of a widely used technology or interface. There, careful observation of the progress and immediate reaction to deviation from the plan are crucial for success.
3. *Architecture process quality indicators* measure stakeholder feedbacks, review efficiency, decision making efficiency, documentation quality and the like.

All performance indicators need to be reviewed regularly by the architecture process owner and the appropriate changes must be made to the other parts of the architecture.

2.9 Architecture Organization: Federated Architecture Process

Very large information systems are typically managed by large organizations. In order for the architecture process to be effective it needs to be embedded properly within the organization. The roles and responsibilities described below need to be defined and implemented.

At the top of the architecture organization there is the *chief architect* responsible for the architecture across the whole, global system. Alternatively, the chief architect is replaced by a committee of senior architects that make decisions. Given the strategic importance of architecture for managed evolution, the chief architect should, ideally, directly report to the CIO (Chief Information Officer) or other manager in charge of the overall IT organization in the company. Key tasks of the chief architect include the following:

- Lead the global architecture organization in its functional and regional dimensions
- Chair the chief architect's council that decides on architectural standards and concepts
- As a direct report to the Chief Information Officer (CIO) the chief architect represent architecture in the IT management team
- Lead the global architecture process, with a particular focus on reviewing projects for architectural alignment
- Drive improvement of architecture, propose IT architecture changes
- Provide IT architecture frameworks and standards
- Lead definition of strategy and roadmap for IT architecture
- Sponsor a project portfolio to help implement architectural standards and concepts, such as the execution of architecture programs
- Approve exceptions to standards where necessary.

Typically the chief architect is supported by a team that facilitates the architecture process by preparing the meetings, recording the decisions, facilitating communications like web conferencing or preparing events and measuring key performance indicators. In addition to that, a group for each of the architecture layers or vertical areas (application architecture, technical architecture, security architecture, etc.) directly reports to the chief architect. The heads

"The idea of Federated Enterprise Architecture is a practical way to employ architectural concepts in complex, multiunit enterprises; however, there is nothing magic happening. Someone has to figure out and actually create and manage what is to be federal, that is, centralized, optimized relative to the overall enterprise (or cluster), integrated, reused, etc. and conversely, someone has to figure out what is not to be federated, that is, what is to remain local, sub-optimized relative to the enterprise, unique to the business units"
John Zachman, 2006

of these groups take responsibility for the respective architecture area in the sense

that they are experts in the field, define relevant architectural concepts and standards for approval by the Chief Architecture Steering Committee and sponsor projects in their area. Typically the architecture area heads also chair teams of experts throughout the organization that help prepare architecture decisions.

Depending on the organization and on the importance of using the latest technology, the chief architect may also lead a research group that helps facilitate innovation in the system. This team would usually be aligned to the external research and industrial community in order to understand relevant technical developments and be able to transfer those into the organization. They generally work with advanced technology studies, prototypes and pilots in order to evaluate new technologies for general use in the system. The leader of the group is responsible for the *innovation portfolio*.

In order to avoid the “ivory tower syndrome”, it is important that all architects in these groups also perform real project work. Very often these architects act as solution architects in projects of high architectural importance. It has proven to be a good split of work, if these people dedicate one third of their time to projects and two thirds of their time to enterprise architecture work. In the experience of the authors, the chief architect’s direct organization should roughly represent 1% of the staff in the overall IT organization.

According to the size of the organization, the chief architect’s organization can be federated and have divisional (= business unit) or regional chief architects with the same responsibility for a business unit or a region of the IT organization as the chief architect has for the entire company. This organization is well suited for large IT organizations with a business unit or regional structure that share substantial parts of their systems. This is typically the case if all business units are in the same business and share a single domain model, or if the business units are heavily dependent on each other, by sharing a single infrastructure. To sum it up, the architecture organization needs to have the same scope as the system that is managed as a whole, which is tightly coupled and governed by the same principles.

In a two-level chief architect’s organization the architecture area groups can be distributed to the business units or the regions according to the main user principle or be kept centrally. Typically each business unit’s chief architect has his own architecture process group supporting the process in his business unit or region.

As architecture should be governed along the domain model, *domain architects* are needed that are responsible for all aspects in their domain with the following key tasks:

- Maintain, develop, communicate and enforce the *target architecture* for the assigned domain. Define roadmaps and strategies to move towards the target domain architecture,
- Together with the other domain architects continue to develop the overarching domain model, decide on assigning technical products or applications to the appropriate domains,

- Maintain the necessary data and object *models* for the assigned domain, align to overarching models, drive prototypes, pilots and reengineering projects in the domain,
- Regularly assess domain fitness, define necessary remediation steps.
- Maintain architectural information about the domain in order to facilitate system-wide analyses,
- Review all projects that affect the domain,
- Lead all solution architects in the domain.

Domain architects need to be senior architects with a broad background, a deep understanding of their domain and outstanding leadership skills. Domain architects should report to the head of the developer group responsible for developments in the domain. As the enterprise-wide architects, domain architects should be deeply involved in projects. In the experience of the authors, a good split between project and architecture governance work for the domain architects is 50% each. All domain architects together typically represent another 1% of the overall IT staff as a good benchmark, including one domain architect and one deputy for each domain.

There are two kinds of domain architects: *Technical domain architects* have responsibility for the architecture in a technical domain. Their second reporting line is into the head of technical architecture. *Application domain architects* have responsibility for application architecture in the domain and have a second reporting line into the head of application architecture. The responsibilities of application and technical domain architects are slightly different because technical domains are mainly buy-and-integrate, whereas application domains are often in-house development-based. One major role of technical domain architects is to understand and act on market trends. In contrast, application domain architects are more focused on the dialog with their business partners to understand requirements.

In the two-level chief architect model, each business unit has their own domain architects for all the domains relevant to the business unit. All business units work with the same domain model. If there are multiple domain architects for the same domain, but in different business units, one of them is the lead domain architect, in charge of system-wide issues in a domain.

The third and largest group of architects is the *solution architects*. Depending on the setup of the development organization, they are grouped in pools or assigned as secondary roles to experienced developers. Each project should have a solution architect assigned, in charge of the technical solution behind the project. Solution architects need to be well trained and guided in order to understand the relevant guidelines and standards applicable to their project. They need good leadership and communication skills to guide the work inside their project team and to communicate with the larger architecture organization to ensure a proper fit of their project work. As explained in Chap. 1 steering every project vector a little bit into the direction of better agility is the most effective way to evolve the system in the right direction. Solution architects are key enablers of this process, as they make most of the micro-architecture design decisions, which if added together constitute the

architectural progress. Solution architects are also responsible to fix issues that come up during the project reviews. But, one needs to be realistic here: Project reviews can identify bad design or certain design flaws. But they cannot turn bad design into good design by means of architecture obligations. The ultimate factor for good design quality is the quality of the work of the solution architects. If they don't master their jobs, the whole organization doesn't work. The architecture process can only help by providing templates and standards that reduce the design space and by reviewing the project results in order to ensure quality. Therefore, a focus is needed on the selection and the development of solution architects. Chap. 6 (People and Culture) will elaborate more on this aspect. Typically, solution architects constitute a few percent of the total IT staff.

Side-Story 2.9: Credit Suisse IT Architecture Organization

Information technology within Credit Suisse is organized as a global unit, serving all business units, with the CIO reporting to the CEO of the company. As information technology is crucial to the bank's success, roughly 20% of the bank's staff belongs to the information technology unit. The bulk of the IT staff, on the one hand, belongs to four business-aligned departments building and maintaining applications for the three core businesses: investment banking; private banking; and for the shared services area. On the other hand, all infrastructure activities and the operation of data centers and applications sit with the technology infrastructure department. Credit Suisse is managed in the four regions Americas, Europe Middle East Africa, Asia Pacific and Switzerland. Each region is managed by a regional CIO in charge of local IT matters, such as human resources, regulatory reporting and day-to-day operation (Fig. 2.13).

The architecture organization is federated and includes a small central team in charge of global tools and processes to maintain the necessary data to manage architecture. These systems include an application and technology portfolio, which holds the current architecture of the global system as well as the formal target architecture, organized along the application and the technical domain model (see Side-Story 2.3). Other central teams manage *security architecture* and *integration architecture*. This reflects the view that security in a globally integrated system is only as good as its weakest link. Integration architecture maintains the necessary standards and interface contracts to enable interoperability across the enterprise. The central organization is lead by the bank's *chief architect*. The chief architect directly reports to the bank's CIO in charge of the information technology unit. The chief architect runs a global architecture steering committee to set standards and guidelines across the bank. In addition to that he/she also runs a global architecture program bundling bank wide *IT architecture investments*, as

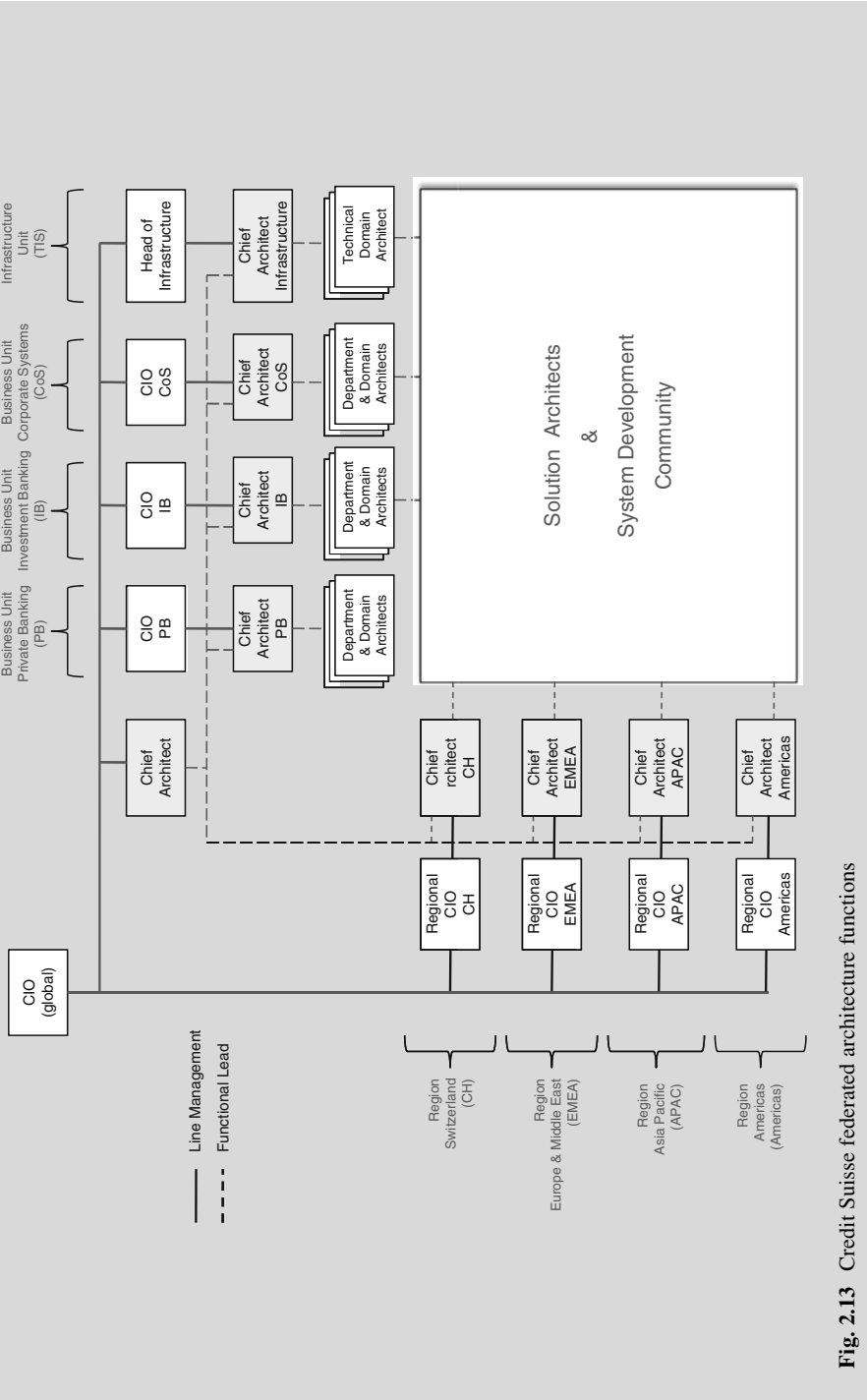


Fig. 2.13 Credit Suisse federated architecture functions

well as coordinating the departmental architecture programs. The chief architect is one of the CIO's two staff functions, the other being the chief operating officer (COO) mainly in charge of resource planning, risk management, financials, project portfolio, reporting and planning of the IT unit.

Furthermore, each of the business units mentioned above have their own architecture organizations, with a chief architect who reports directly into the business units' CIO with a functional management line into the bank's chief architect. The departmental chief architect runs a departmental architecture steering committee, in charge of setting departmental standards and guidelines according to the overarching guidelines. These groups run the day-to-day architecture implementation process by managing a departmental architecture program. Furthermore, they participate in a departmental project review board, reviewing all projects, thus ensuring adherence to all relevant standards. For the application oriented departments this mainly means data and application architecture. One particular case is the *infrastructure* chief architect, who is at the same time departmental chief architect of the technology infrastructure services department. In contrast to his application colleagues who cover the application domain model (see Side-Story 2.3) within their department, he covers the whole technical domain model (see Sect. 4.2). He has groups in charge of the technology portfolio and platform architecture managing the interface between applications and infrastructure, as well as system management architecture ensuring efficient operation of platforms and applications. As with his colleagues, infrastructure architecture maintains its own architecture program and enforces standards in infrastructure projects via the departmental project review board.

The most senior architect in a region – who is usually one of the departmental chief architects – acts as regional chief architect with a dotted line to the regional CIO. The regional responsibility is mainly for communication, culture and building up an architecture community in the region. Furthermore, he is responsible for finding synergy opportunities across business units from a regional perspective. Often it makes sense to use applications across business units in a local market, even if that isn't the case globally. Regional chief architects bring their unique regional perspective into the global architecture steering committee. Typical regional peculiarities in the banking business include local regulatory requirements or input on a function the region is particularly strong in, as for example the Asia Pacific region as a global software development center.

People in these functions are generally well-trained, experienced enterprise architects with strong technical, strategic and communication skills. As Credit Suisse believes that architects should be able to work hands-on, all architects on the first two levels have an obligation to spend 25% of their time on project work, often in the role of a solution architect in an important project.

The departmental chief architects have functional lines to the domain architects, which organizationally sit with the corresponding development and engineering departments. If the same domain is being worked on in multiple departments, each has its own domain architect. Domain architects of the same domain coordinate with each other. Appointing a lead domain architect has been considered in cases where there are particularly strong synergy opportunities among business units. Domain architects spend roughly 50% of their time in strategic planning, coordination and alignment of their domain, while they spend the other 50% on projects, typically as solution architect.

The *domain architects* in turn lead the solution architects responsible for architecture of the individual projects touching a domain. Each project has the role of an architect in charge of the solution design in the project. In large projects, solution architect is usually a distinct role, while a senior developer might take it on in smaller projects.

The organization described in Side-Story [2.9](#) is appropriate to Credit Suisse's IT unit with more than 10'000 headcount. For a smaller organization one layer less would be sufficient.



<http://www.springer.com/978-3-642-01632-5>

Managed Evolution

A Strategy for Very Large Information Systems

Murer, S.; Bonati, B.

2011, XXIV, 264 p., Hardcover

ISBN: 978-3-642-01632-5