Chapter 2
State of the Art

Marc M. Lankhorst, Maria-Eugenia Iacob, and Henk Jonkers

First, we position enterprise architecture relative to a number of well-known standards and best practices in general and IT management. Second, we outline the most important frameworks and methods for enterprise architecture currently in use. Next, we discuss service orientation, the most important architectural paradigm that has emerged over the last few years. Finally, we describe a number of relevant languages for modelling organisations, business processes, applications, and technology.

Based upon this state of the art, in the next chapter we will describe what we see as missing in current methods and techniques, and how our own approach tries to fill some of these gaps.

2.1 Enterprise Architecture and Other Governance Instruments

Enterprise architecture is typically used as an instrument in managing a company's daily operations and future development. But how does it fit in with other established management practices and instruments?

Here, we describe how enterprise architecture is positioned within the context of corporate and IT governance by relating it to a number of well-known best practices and standards in general and IT management, as outlined in Fig. 2.1. In the next

M.M. Lankhorst (✉)
BiZZdesign, Capitool 15, 7521 PL Enschede, The Netherlands
e-mail: m.lankhorst@bizzdesign.com

M.-E. Iacob
University of Twente, Enschede, The Netherlands

H. Jonkers
BiZZdesign, Enschede, The Netherlands

M. Lankhorst et al., *Enterprise Architecture at Work*, The Enterprise Engineering Series, DOI 10.1007/978-3-662-53933-0_2

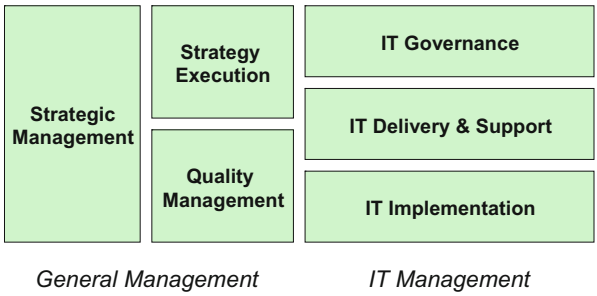

Fig. 2.1 Management areas relevant to enterprise architecture

subsections, we will outline the relation of enterprise architecture with some well-known management practices in each of these areas, not to be exhaustive but to show the position and role of enterprise architecture in a management context:

- Strategic management: Balanced Scorecard
- Business model development: Business Model Canvas
- Business architecture: BIZBOK® Guide and O-BA
- Quality management: EFQM and ISO 9001
- IT governance: COBIT
- IT delivery and support: ITIL
- IT implementation: CMM and CMMI

Others have also written extensively on this role of enterprise architecture as a governance instrument; see e.g. (Ross et al. 2006).

2.1.1 Strategic Management

Kaplan and Norton (1992) introduced the Balanced Scorecard (BSC) as a management system that helps an enterprise to clarify and implement its vision and strategy. Traditionally, management focus has strongly been on financial aspects. Kaplan and Norton argue that financial measures alone are inadequate to guide the future development of an organisation, and that they should be supplemented with measures concerning customer satisfaction, internal processes, and the ability to innovate.

The BSC therefore suggests viewing an enterprise from four perspectives. The *Customer* perspective asks how the enterprise should appear to its customers, with measures like customer satisfaction. The *Financial* perspective is focused on the business value created by the enterprise, entailing measures such as shareholder value. The *Internal Business Processes* perspective looks at the effectiveness and efficiency of a company's internal operations, paying special attention to the primary, mission-oriented processes. Finally, the *Learning and Growth* perspective addresses the corporate and individual ability to change and improve, which is critical to any knowledge-intensive organisation. For each of the four perspectives the BSC proposes a three-layered structure:

1. mission (e.g., to become the customers' preferred supplier);
2. objectives (e.g., to provide the customers with new products);
3. measures (e.g., percentage of turnover generated by new products).

To put the BSC to work, a company should first define its mission, objectives, and measures for each perspective, and then translate these into a number of appropriate targets and initiatives to achieve these goals. Strategy maps (Kaplan and Norton 2004) are often used as layered depiction of these elements and their relationships.

What is important in the BSC is the notion of double-loop feedback. First of all, one should measure the outputs of internal business processes and not only fix defects in these outputs but also identify and remedy the causes of these defects. Moreover, such a feedback loop should also be instituted for the outcomes of business strategies. Performance measurement and management by fact are central to the BSC approach.

If we look at the role of enterprise architecture as a management instrument, it is especially useful within the Internal Business Processes perspective of the BSC. Many operational metrics can be tied to a well-defined enterprise architecture and various performance analyses might be carried out. However, enterprise architecture has a broader use. In the Learning and Growth perspective, a company's ability to evolve, to anticipate, and to respond to a changing environment is vital. To determine an organisation's agility, it is important to assess what the impact and feasibility of future changes might be. Impact analysis of an enterprise architecture may assist in such an assessment.

In Sect. 6.3, we describe how the ArchiMate modelling language for enterprise architecture (The Open Group 2016a) introduced in Chap. 5 can be used to describe the Balanced Scorecard.

2.1.2 Business Model Development

The Business Model Canvas (Osterwalder and Pigneur 2010) is a template to create high-level descriptions of new or existing business models. It is conceptually rooted in the business model ontology described in Osterwalder's PhD thesis (Osterwalder 2004). The Business Model Canvas consists of seven parts:

- **Value proposition**: the centre of the canvas, describing what products and services an organisation has to offer to its different customers
- **Key activities**: what the organisation needs to do to provide its value propositions
- **Key resources**: the resources needed for these activities
- **Customer segments**: the typical customer groups the organisation distinguishes
- **Customer relationships**: the kind of links the organisation has with its customers
- **Channels**: how the organisation gets in touch with its customers.

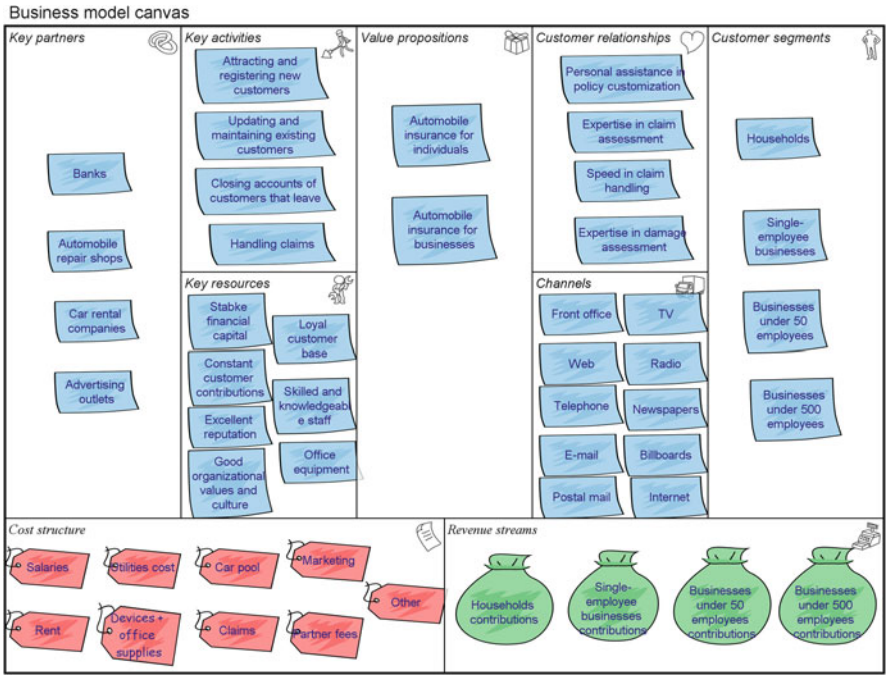

Fig. 2.2 Example business model canvas

- **Key partners**: others with which the organisation cooperates in delivering value to its customers
- **Cost structure**: the financial view of the means employed by the organisation
- **Revenue streams**: the way the organisation makes money from various revenue flows from its customer segments

The Business Model Canvas lays out these elements in a user-friendly, intuitive way. It is often used in a brainstorming or workshop context, sometimes using simple 'sticky notes', sometimes in a tool-supported fashion. Figure 2.2 shows a small example of such a canvas.

Enterprise architecture is typically used as a next stage in strategic development. Many of the elements in a canvas can be detailed out using enterprise architecture models. In Sect. 6.4, we describe how the concepts in the Business Model Canvas can be mapped onto those of the ArchiMate language introduced in Chap. 5.

2.1.3 Business Architecture

In recent years, business architecture has gained an increasing audience and has established itself as a distinct discipline. Partially fueled by often rather IT-focused

enterprise architecture approaches and efforts, it has developed its own methods and body of knowledge, exemplified by "A Guide to the Business Architecture Body of Knowledge®" (BIZBOK® Guide) (Business Architecture Guild 2016) and the Open Business Architecture (O-BA) method being developed by The Open Group at the time of writing (2016c).

The Business Architecture Special Interest Group (BASIG) of the OMG has defined business architecture as follows:

Business architecture: a blueprint of the enterprise that provides a common understanding of the organisation and is used to align strategic objectives and tactical demands.

A business architecture provides a business-oriented abstraction of the enterprise in its ecosystem, which helps to translate strategy into action. The role of model-based support for design, analysis and decision making is becoming increasingly important in the business architecture discipline.

Key input for business architecture is the organisation's strategy, which includes its business model, for example, described using the Business Model Canvas (Sect. 2.1.2), and its operating model (Sect. 1.4.1). Commonly used design techniques in business architecture include, among others, describing its value network and value streams (Sect. 6.5), developing and improving customer journeys (Sect. 6.6) and creating service blueprints (Sect. 6.7). Analysis and decision making in business architecture are supported by, for example, risk analysis (Sect. 9.4), portfolio management (Sect. 9.5) and capability-based planning (Sects. 8.7.1 and 9.6).

Typical concepts and aspects that the domain of business architecture concerns itself with are shown in Fig. 2.3. Basically all of these can be represented directly or indirectly in the ArchiMate language, which will also be illustrated in Chap. 6 on combining ArchiMate with other standards and approaches.

In particular, the focus on the capabilities of the enterprise, with capability-based planning (Ulrich and Rosen 2011; The Open Group 2016b) as a core technique, is a key contribution of the business architecture discipline. This allows an organisation to focus on *what* the current and desired abilities of the enterprise are, before diving into the details of *how* it achieves these. This implementation- and technology-independent view provides a crucial connection between strategy and realisation: It links the often rather high-level, coarse-grained descriptions of an organisation's strategy and business model with more detail- and technology-oriented other domains within the EA scope, such as business process, application and infrastructure architecture.

In the context of this book, we consider business architecture to be an important domain within the broader scope of enterprise architecture, hence the inclusion of business architecture concepts such as capability, outcome and course of action in the current version of the ArchiMate modelling language for enterprise architecture described in Chap. 5. Others take a more IT-oriented view of enterprise architecture, considering it to be enterprise-wide IT architecture and position business architecture next to it as a separate discipline. A third group see business and enterprise architecture as largely synonymous. But no matter which stance you

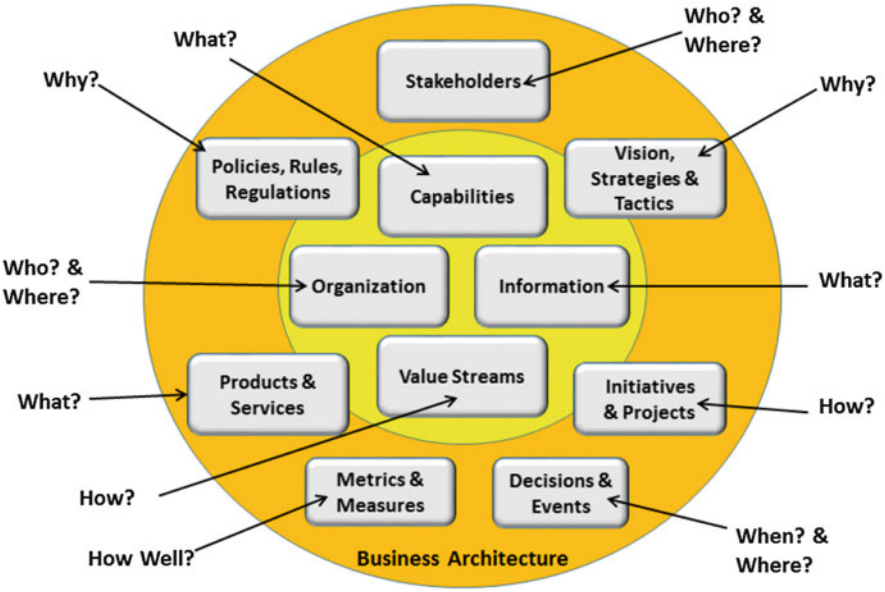

Fig. 2.3 The business ecosystem as represented by business architecture (Business Architecture Guild 2016)

take in this debate, the techniques used in the domain are an important addition to the toolbox you can use in designing and managing your enterprise.

2.1.4 Quality Management

Another important management approach is the EFQM (European Foundation for Quality Management) Excellence Model (EFQM 2003). This model was first introduced in 1992 as the framework for assessing applications for The European Quality Award, and was inspired by the Malcolm Baldridge Model in the USA and the Deming Prize in Japan.

The EFQM model has a much broader scope than ISO 9001 (discussed later in this section). It not only focuses on quality management, but provides an overall management framework for performance excellence of the entire organisation. The EFQM model consists of nine criteria for excellence, five of which are 'enablers', covering what an organisation does, and four are 'results', covering what that organisation achieves. These criteria and their mutual relationships are shown in diagrammatic form in Fig. 2.4. Leadership and Policy & Strategy determine the direction and focus of the enterprise; based on this, the People of the enterprise, its Partnerships & Resources, and its Processes make it happen; stakeholders of the results achieved are its Customers, its People, and Society in general; and these

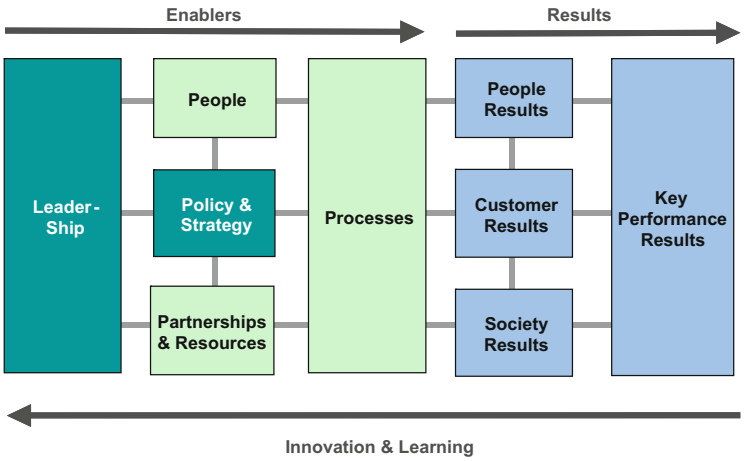

Fig. 2.4 The EFQM excellence model (EFQM 2003)

stakeholder results contribute to the enterprise's Key Performance Results, which comprise both financial and non-financial aspects. The EFQM model provides principles, measures, and indicators for assessing the performance of an enterprise in all of these aspects, and these measurements are the basis for continuous learning, innovation, and improvement.

All this also points to the main difference between the EFQM model and the BSC: whereas the latter is focused on developing effective strategic management, the former concentrates on measuring and benchmarking the performance of an organisation with respect to a number of best practices. Both are complementary: the BSC helps to make strategic choices, and the EFQM model assists in continuous improvement necessary to execute this strategy.

Positioning enterprise architecture with respect to the EFQM model, we view it especially as an important instrument for the Policy & Strategy and the Processes aspects. Based on its mission and vision, an organisation will determine the policies and strategies needed to meet the present and future needs and expectations of its stakeholders. An enterprise architecture is a valuable instrument in operationalising and implementing these policies and strategies. First of all, it offers insight into the structure and operation of the enterprise as a whole by creating a bird's-eye view of its organisational structure, business processes, information systems, and infrastructure. Such an overview is indispensable when formulating a coherent strategy. Furthermore, an enterprise architecture helps in developing, managing, and communicating company-wide standards of operation, needed to ensure that company policies are indeed implemented. Finally, by providing a better understanding of the effects of changes, it is of valuable assistance in creating roadmaps for the future, needed to assess and execute the longer-term enterprise strategy.

Another important standard in the domain of quality management is the ISO 9001:2000 standard (ISO 2000) of the International Organisation for

Standardisation (ISO). This outlines criteria for a good-quality management system (QMS). Based on a quality policy and quality goals, a company designs and documents a QMS to control how processes are performed. The requirements of the standard cover everything from how a company plans its business processes, to how these are carried out, measured, and improved.

Starting from general, overall requirements, the standard states the responsibilities of management for the QMS. It then gives requirements for resources, including personnel, training, the facility, and work environment. The demands on what is called 'product realisation', i.e., the business processes that realise the company's product or service are the core of the standard. Key processes, i.e., those processes that affect product or service quality, must be identified and documented. This includes planning, customer-related processes, design, purchasing, and process control. Finally, requirements are put on measurement, analysis, and improvement of these business processes. Once the quality system is installed, a company can request an audit by a Registrar. If it conforms to all the criteria, the company will be ISO 9001 registered.

Although the standard has earned a reputation as being very 'document-heavy', this mainly pertains to its previous versions of 1987 and 1994. Notwithstanding these criticisms, the business value of a good QMS is universally acknowledged. In Europe, industrial companies increasingly require ISO 9001 registration from their suppliers, and the universal acceptance as an international standard is growing.

Looking at enterprise architecture from the perspective of quality management in general and ISO 9001 in particular, we see its main contribution in the integrated design, management and documentation of business processes, and their supporting IT systems. A well-designed and documented enterprise architecture helps an organisation to conform to the ISO 9001 requirements on process identification and documentation; conversely, the need for a QMS may direct focus to an enterprise architecture initiative, by putting the emphasis on those processes and resources that are critical for the company's product or service quality. In this way, quality management and enterprise architecture form a natural combination: the former is concerned with *what* needs to be designed, documented, controlled, measured, and improved, and the latter determines *how* these high-quality processes and resources are organised and realised.

2.1.5 IT Governance

The COBIT (Control Objectives for Information and Related Technology) standard for IT governance was initially published in 1996 by the Information Systems Audit and Control Association. Now in its fifth edition (Stroud 2012), COBIT is an internationally accepted IT control framework that provides organisations with 'good practices' that help in implementing an IT governance structure throughout the enterprise. It aims to bridge the gaps between business risks, control needs, and technical issues. The basic premise of COBIT is that in order to provide the

information that the organisation needs to achieve its objectives, IT resources need to be managed by a set of naturally grouped processes.

The core of the COBIT framework is the control objectives and management guidelines for a set of IT processes, which are grouped into five domains:

1. Evaluate, direct and monitor
2. Align, plan and organise
3. Build, acquire and implement
4. Deliver, service and support
5. Monitor, evaluate and assess

Here, 'control' is defined by COBIT as the policies, procedures, practices, and organisational structures designed to provide reasonable assurance that business objectives will be achieved and that undesired events will be prevented or detected and corrected. The control objectives can help to support IT governance within an enterprise. For example, the control objectives of the 'Assist and advise IT customers' process consist of establishing a help desk, registration of the customer queries, customer query escalation, monitoring of clearance, and trend analysis and reporting.

Next to the framework of control objectives, COBIT provides critical success factors for achieving optimal control over IT processes, key goal indicators, which measure whether an IT process has met its business requirements, and key performance indicators, which define measures of how well the IT process is performing towards achieving its goals.

COBIT also offers a maturity model for IT governance, consisting of five maturity levels:

1. **Ad Hoc**: There are no standardised processes. Ad hoc approaches are applied on a case-by-case basis.
2. **Repeatable**: Management is aware of the issues. Performance indicators are being developed, and basic measurements have been identified, as have assessment methods and techniques.
3. **Defined**: The need to act is understood and accepted. Procedures have been standardised, documented and implemented. BSC ideas are being adopted by the organisation.
4. **Managed**: Full understanding of issues on all levels has been reached. Process excellence is built on a formal training curriculum. IT is fully aligned with the business strategy.
5. **Optimised**: Continuous improvement is the defining characteristic. Processes have been refined to the level of external best practices based on the results of continuous improvement with other organisations.

This maturity model closely resembles the Capability Maturity Model (CMM) for software development and its successor the CMMI (see Sect. 2.1.7).

According to COBIT, well-defined architectures are the basis for a good internal control environment. In many enterprises, the IT organisation will be responsible for establishing and maintaining the enterprise architecture. Whereas COBIT

focuses on how one should organise the (secondary) IT function of an organisation, enterprise architecture concentrates on the (primary) business and IT structures, processes, information and technology of the enterprise. Thus, enterprise architecture forms a natural complement to COBIT. Relative to the maturity levels of COBIT, enterprise architecture will of course be most relevant in the upper level. At the Repeatable level, a first awareness of the value of architecture may arise, but there is typically no established architectural practice at the enterprise level. Only from the Defined level upwards is it recognised and used as an important instrument in planning and managing IT developments in coordination with business needs.

2.1.6 IT Service Delivery and Support

ITIL (IT Infrastructure Library) (Hanna et al. 2008) is the most widely accepted set of best practices in the IT service delivery domain. It was originally developed by the UK Office of Government Commerce (OGC), to improve management of IT services in the UK central government. The OGC's objectives were on the one hand to create a comprehensive and consistent set of best practices for quality IT service management, and on the other hand to encourage the private sector to develop training, consultancy, and tools that support ITIL. Over the years, ITIL has gained broad support and has become the worldwide *de facto* standard for IT service management. The ITIL users group, the IT Service Management Forum (*it*SMF[1]), actively promotes the exchange of information and experiences to help IT service providers manage service delivery.

ITIL comprises a series of documents giving guidance on the provision of good IT services, and on the facilities needed to support IT. ITIL has a process-oriented approach to service management. It provides codes of practice that help organisations to establish quality management of their IT services and infrastructure, where 'quality' is defined as 'matched to business needs and user requirements as these evolve.' It does this by providing guidance on the design and implementation of the various processes within the IT organisation. The core of ITIL consists of two broad groups of processes:

- Service Delivery, comprising service-level management, availability management, financial management for IT services, IT service contingency management, and capacity management;
- Service Support, covering problem management, incident management, service desk, change management, release management, and configuration management.

ITIL is complementary to COBIT. The high-level control objectives of COBIT can be implemented through the use of ITIL. Its help desk module, for example,

[1] http://www.itsmf.com

complements and provides details on the help desk process including the planning, implementation, post-implementation, benefits and costs, and tools. So, COBIT's control objectives tell what to do and ITIL explains how to do it, i.e., what the best-practice processes are to realise these objectives.

Management of the IT assets of an organisation is central to ITIL. This is where a well-developed enterprise architecture is very valuable. It provides IT managers with a clear understanding of the IT applications and infrastructure, the related business processes, and the various dependencies between these domains. Nearly all of the core processes identified by ITIL will benefit from this.

2.1.7 IT Implementation

The Capability Maturity Model for Software (Paulk et al. 1993), also known as the CMM and SW-CMM, is a model for judging the maturity of an organisation's software engineering processes, and provides organisations with key practices required to help them increase the maturity of these processes. In 2000, the SW-CMM was upgraded to CMMI (Capability Maturity Model Integration), which addresses the integration of software development with other engineering activities and expands the scope to encompass the entire product life cycle, including systems engineering, integrated product and process development, and supplier sourcing. The CMM's popularity has sparked off the development of similar maturity models in other fields, including enterprise architecture; see, e.g., the NASCIO Enterprise Architecture Maturity Model (NASCIO 2003).

In the CMMI maturity models in their most common form, there are five maturity levels, each a layer in the foundation for ongoing process improvement, designated by the numbers 1–5 (CMMI Product Team 2002):

1. **Initial**: Processes are usually ad hoc and chaotic. The organisation does not provide a stable environment. Success in these organisations depends on the competence and heroics of the people in the organisation and not on the use of proven processes.
2. **Managed**: The projects of the organisation have ensured that requirements are managed and that processes are planned, performed, measured, and controlled. However, processes may be quite different in each specific instance, e.g., on a particular project.
3. **Defined**: Processes are well characterised and understood, and are described in standards, procedures, tools, and methods. These standards are used to establish consistency across the organisation. Projects establish their defined processes by tailoring the organisation's set of standard processes according to tailoring guidelines.
4. **Quantitatively Managed**: Quantitative objectives for quality and process performance are established and used as criteria in managing processes. Quantitative objectives are based on the needs of the customer, end users, organisation, and process implementers.

5. **Optimising**: Process performance is continually improved through both incremental and innovative technological improvements. Quantitative process-improvement objectives for the organisation are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement.

The CMMI provides numerous guidelines for assessing the maturity of an organisation and the improvements needed in various process areas to proceed from one level to the next. Next to this familiar staged representation of the maturity model in terms of consecutive maturity levels, there is now a continuous representation as well.

In any software engineering project of substantial size, software architecture plays an important role. The context of this software architecture may be given by an enterprise architecture, which provides constraints and guidelines for individual software projects. As such, enterprise architecture is something that becomes especially useful (or even necessary) at CMMI Level 3 and beyond, where projects have to conform to organisation-wide standards and guidelines.

2.2 Architecture Methods and Frameworks

To provide more insight into the different aspects that an enterprise architecture model may encompass, we will outline a number of well-known architecture frameworks, standards and approaches. Frameworks structure architecture description techniques by identifying and relating different architectural viewpoints and the modelling techniques associated with them. They do not provide the concepts for the actual modelling, although some frameworks are closely connected to a specific modelling language or set of languages.

Most architecture frameworks are quite precise in establishing what elements should be part of an enterprise architecture. However, to ensure the quality of the enterprise architecture during its life cycle the adoption of a certain framework is not sufficient. The relations between the different types of domains, views, or layers of the architecture must remain clear, and any change should be carried through methodically in all of them. For this purpose, a number of methods are available, which assist architects through all phases of the life cycle of architectures.

2.2.1 The IEEE 1471-2000/ISO/IEC 42010 Standard

In 2000, the IEEE Computer Society approved IEEE Standard 1471-2000 (IEEE Computer Society 2000), which builds a solid theoretical base for the definition, analysis, and description of system architectures. IEEE 1471, which has since been subsumed by the ISO/IEC 42010 standard (ISO/IEC/IEEE 2011), focuses mainly

on software-intensive systems, such as information systems, embedded systems, and composite systems in the context of computing. The standard uses the civil architecture metaphor to describe software system architectures. In this sense, it is similar to the framework of Zachman (see Sect. 2.2.2), although it does not try to standardise the system architecture by establishing a fixed number, or the nature of views (as in the case of the 36 cells of Zachman's framework). It also does not try to standardise the process of developing architectures, and therefore does not recommend any modelling languages, methodologies or standards. Instead, it provides, in the terms of a 'recommended practice', a number of valuable concepts and terms of reference, which reflect the 'generally accepted trends in practice for architecture description' and which 'codify those elements on which there is consensus'.

First of all, the standard gives a set of definitions for key terms such as acquirer, architect, architecture description, architectural models, architecture, life cycle model, system, system stakeholder, concerns, mission, context, architectural view, architectural viewpoint. As essential ideas we note a clear separation between an architecture and its architecture descriptions (defined as means to record architectures), and the central role of the relationship between architectural viewpoint and architectural view. The standard also provides a conceptual framework, which is meant:

– To explain how the key terms relate to each other in a conceptual model for architecture description (this model is shown in Fig. 2.5, using the UML notation for class diagrams; see also Sect. 2.3.3)

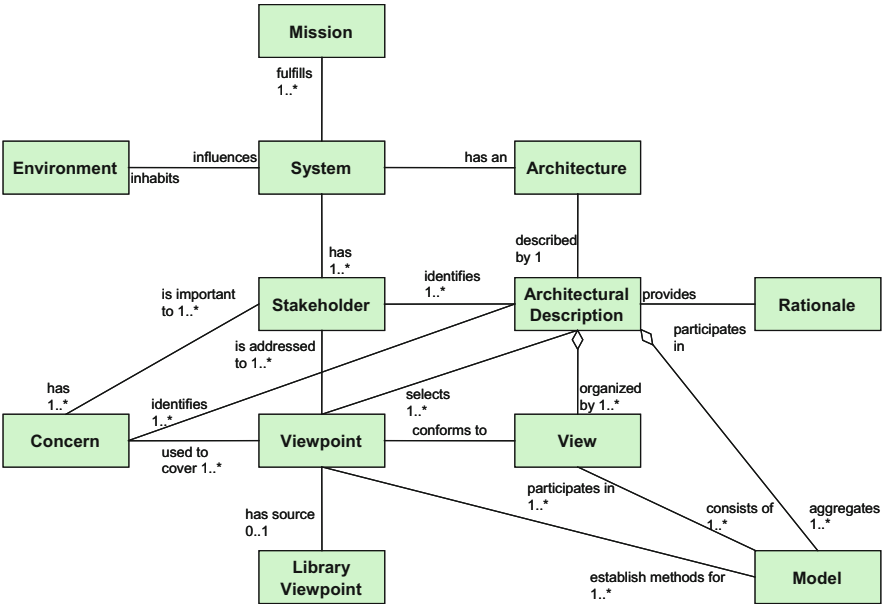

Fig. 2.5 Conceptual model of architecture description (based on IEEE Computer Society 2000)

- To explain the role of the stakeholders in the creation and use of an architecture description
- To provide a number of scenarios for the architectural activities during the life cycle: architectures of single systems, iterative architecture for evolutionary systems, architecture for existing systems and architectural evaluation

Furthermore, the standard gives six architecture description practices:

- Architectural documentation referring to identification, version, and overview information.
- Identification of the system stakeholders and of their concerns, established to be relevant to the architecture.
- Selection of architectural viewpoints, containing the specification of each viewpoint that has been selected to organise the representation of the architecture and the reasons for which it was selected.
- Architectural views corresponding to the selected viewpoints.
- Consistency among architectural views.
- Architectural rationale for the selection of the current architecture from a number of considered alternatives.

IEEE 1471 also provides a number of relevant architectural viewpoints together with their specifications in terms of concerns, languages, and modelling and analysis methods (see Annex D of the standard). It is important to note that architecture descriptions that are compliant with IEEE 1471 can be used to meet the requirements of other standards, like the Reference Model of Open Distributed Processing (described in Sect. 2.2.5).

2.2.2 The Zachman Framework

In 1987, John Zachman introduced the first and best-known enterprise architecture framework (Zachman 1987), although back then it was called 'Framework for Information Systems Architecture'. The framework as it applies to enterprises is simply a logical structure for classifying and organising the descriptive representations of an enterprise that are significant to the management of the enterprise as well as to the development of the enterprise's systems.

The framework (Fig. 2.6) in its most simple form depicts the design artefacts that constitute the intersection between the roles in the design process, that is, *owner*, *designer* and *builder*, and the product abstractions, that is, *what* (material) it is made of, *how* (process) it works and *where* (geometry) the components are relative to one another. Empirically, in the older disciplines, some other 'artefacts' were observable that were being used for scoping and for implementation purposes. These roles are somewhat arbitrarily labelled *planner* and *subcontractor* and are included in the framework graphic that is commonly exhibited.

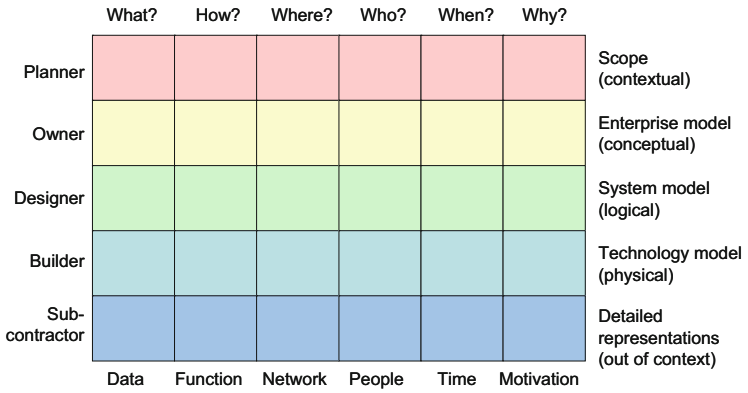

Fig. 2.6 The Zachman framework (Zachman 1987)

From the very inception of the framework, some other product abstractions were known to exist because it was obvious that in addition to *what*, *how*, and *where*, a complete description would necessarily have to include the remaining primitive interrogatives: *who*, *when* and *why*. These three additional interrogatives would be manifest as three additional columns of models that, in the case of enterprises, would depict: *who* does what work, *when* do things happen, and *why* are various choices made?

Advantages of the Zachman framework are that it is easy to understand, it addresses the enterprise as a whole, it is defined independently of tools or methodologies, and any issues can be mapped against it to understand where they fit. An important drawback is the large number of cells, which is an obstacle for the practical applicability of the framework. Also, the relations between the different cells are not that well specified. Notwithstanding these drawbacks, Zachman is to be credited with providing the first comprehensive framework for enterprise architecture, and his work is still widely used.

2.2.3 The Open Group Architecture Framework

The Open Group Architecture Framework (TOGAF) originated as a generic framework and methodology for development of technical architectures, but evolved into an enterprise architecture framework and method. From version 8 onwards, TOGAF (The Open Group 2011) is dedicated to enterprise architectures.

TOGAF has the following main components (Fig. 2.7):

– An Architecture Capability Framework, which addresses the organisation, processes, skills, roles, and responsibilities required to establish and operate an architecture function within an enterprise.

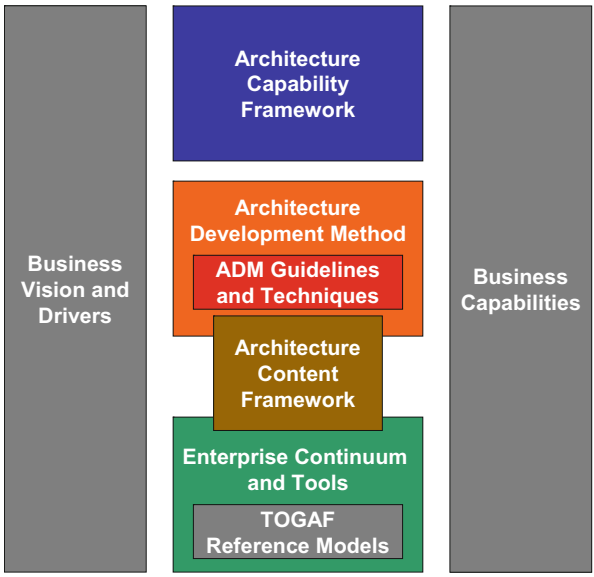

Fig. 2.7 TOGAF 9.1 (The Open Group 2011)

- The Architecture Development Method (ADM), which provides a 'way of working' for architects. The ADM is considered to be the core of TOGAF, and consists of a stepwise cyclic approach for the development of the overall enterprise architecture.
- The Architecture Content Framework, which considers an overall enterprise architecture as composed of four closely interrelated architectures: Business Architecture, Data Architecture, Application Architecture, and Technology (IT) Architecture.
- The Enterprise Continuum, which comprises various reference models, such as the Technical Reference Model, The Open Group's Standards Information Base (SIB), and The Building Blocks Information Base (BBIB). The idea behind the Enterprise Continuum is to illustrates how architectures are developed across a continuum ranging from foundational architectures, through common systems architectures and industry-specific architectures, to an enterprise's own individual architecture.

TOGAF's ADM (Fig. 2.8) is iterative, over the whole process, between phases and within phases. For each iteration of the ADM, a fresh decision must be taken as to:

- The breadth of coverage of the enterprise to be defined;
- The level of detail to be defined;
- The extent of the time horizon aimed at, including the number and extent of any intermediate time horizons;

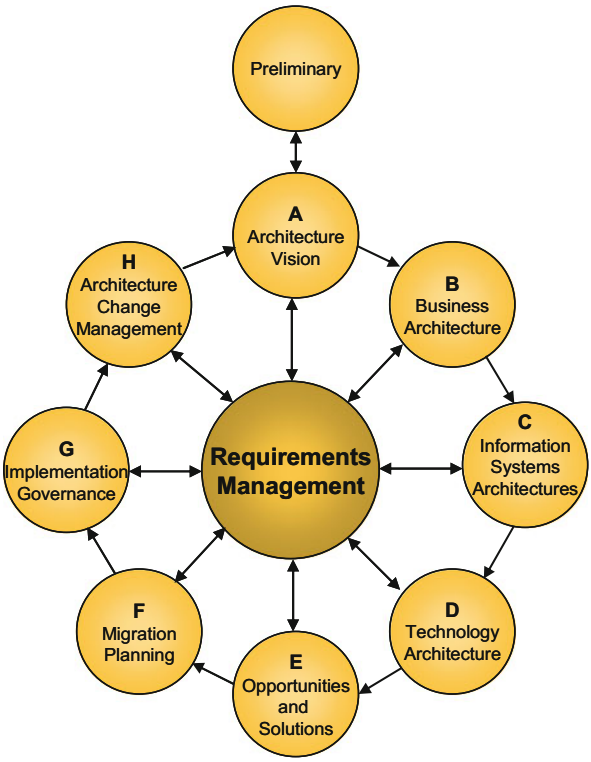

Fig. 2.8 TOGAF architecture development method (The Open Group 2011)

– The architectural assets to be leveraged in the organisation's Enterprise Continuum, including assets created in previous iterations of the ADM cycle within the enterprise and assets available elsewhere in the industry.

These decisions need to be made on the basis of a practical assessment of resource and competence availability, and the value that can realistically be expected to accrue to the enterprise from the chosen scope of the architecture work.

As a generic method, the ADM is intended to be used by enterprises in a wide variety of different geographies and applied in different vertical sectors/industry types. As such, it may be, but does not necessarily have to be, tailored to specific needs. For example:

It may be used in conjunction with the set of deliverables of another framework, where these have been deemed to be more appropriate for a specific organisation. (For example, many US federal agencies have developed individual frameworks that define the deliverables specific to their particular departmental needs).

It may be used in conjunction with the well-known Zachman framework, which is an excellent classification scheme, but lacks an openly available, well-defined methodology.

In Sect. 6.13, we will provide more detail on combining the ArchiMate language defined in Chap. 5 with the TOGAF framework.

2.2.4 OMG's Model-Driven Architecture

The Model-Driven Architecture (MDA) (Object Management Group 2014; Frankel 2003) aims to provide an open, vendor-neutral approach to interoperability. It builds upon the Object Management Group's modelling standards: the Unified Modeling Language (UML; see also Sect. 2.3.3), the Meta Object Facility (MOF) (Object Management Group 2015d) and the Common Warehouse Meta-model (CWM). Platform-independent application descriptions built with these standards can be realised using different open or proprietary platforms, such as Java, .NET, XMI/XML and Web services.

MDA wants to raise the level of abstraction at which software solutions are specified by defining a framework supported by a collection of standards that sets a standard for generating code from models and vice versa. Now, MDA-based software development tools already support the specification of software in UML instead of in a programming language like Java.

MDA comprises three abstraction levels with mappings between them (see Fig. 2.9):

1. The requirements for the system are modelled in a domain model or business model, historically called Computation-Independent Model (CIM) in MDA,

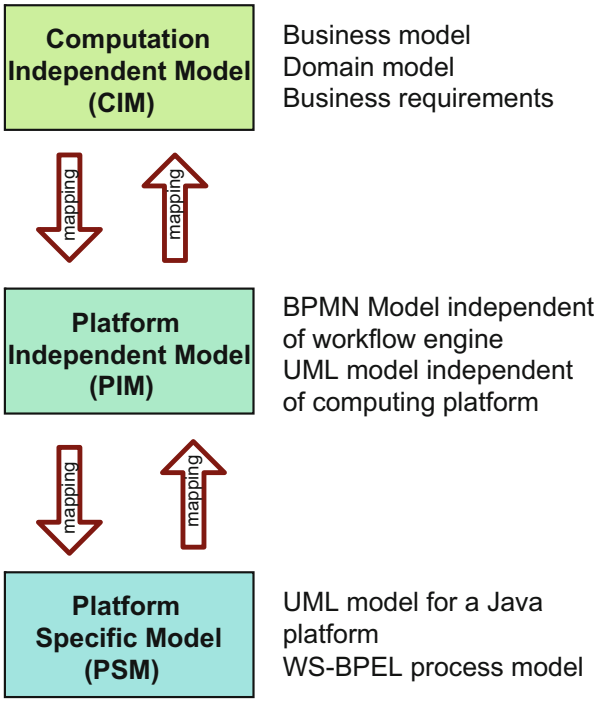

Fig. 2.9 MDA framework

which describes the situation in which the system will be used. It hides much or all information about the use of automated data processing systems.
2. The Platform-Independent Model (PIM) describes the operation of a system while hiding the details necessary for a particular platform. A PIM shows that part of the complete specification that does not change from one platform to another.
3. A Platform-Specific Model (PSM) combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

UML is endorsed as the modelling language for both PIMs and PSMs. At the CIM level, a language for business process specification such as BPMN (Sect. 2.3.2) may be used, and languages for the description of business rules and business models are also available.

One of the key features of the MDA is the notion of *mapping*. A mapping is a set of rules and techniques used to modify one model to get another model. In certain restricted situations, a fully automatic transformation from a PIM to a PSM may be possible, and software development tools will support these automated mappings. To what extent automation of mappings between CIMs and PIMs is feasible is still a topic of research. If these mappings are performed in a predefined (formal) way, relations between models of different abstraction levels can be assured. The ArchiMate language introduced in Chap. 5 would typically be positioned at the CIM level of the MDA. In Sects. 6.1.7 and 6.1.8, we describe how several ArchiMate concepts can be mapped to BPMN and UML, respectively.

The Meta Object Facility (MOF) (Object Management Group 2015c) is a standard for repositories that plays a central role in the MDA framework. A MOF-compliant repository makes it possible to manage models in an integrated fashion, even when the models are expressed in different languages. In order to make a repository effective for EA, it must be possible to model relations between models in the repository. MOF in itself does not offer a solution for this, but models in a modelling language like ArchiMate can be added in order to model these relations. In addition to MOF, OMG has developed the QVT (Queries, Views, and Transformations) specification (Object Management Group 2016b), which addresses the way mappings are achieved between models whose languages are defined using MOF and defines a standard way of querying MOF models and creating views of these models.

2.2.5 Other Frameworks

DoDAF/C^{4}ISR The Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C^{4}ISR) Architecture Framework (C4ISR Architecture Working Group 1997) was originally developed in 1996, for the US Department of Defense, to ensure a common unifying approach for the commands, military services, and defence agencies to follow in describing their

various architectures. The framework was retitled Department of Defense Architecture Framework (DoDAF) in 2003 (Department of Defense 2007). Although DoDAF has a rather specific target, it can be extended to system architectures that are more general. DoDAF sees the architecture description as an integration of three main views: operational view, system view, and technical view. A number of concepts and fundamental definitions (e.g., architecture, architecture description, roles, and interrelationships of the operational, systems, and technical architecture views) are provided. Some framework-compliant guidelines and principles for building architecture descriptions (including the specific product types required for all architecture descriptions), and a Six-Step Architecture Description procedure, complement them.

RM-ODP The Reference Model for Open Distributed Processing (RM-ODP) is an ISO/ITU Standard (ITU 1996) which defines a framework for architecture specification of large distributed systems. The standard aims to provide support for interworking, interoperability, portability and distribution, and therefore to enable the building of open, integrated, flexible, modular, manageable, heterogeneous, secure, and transparent systems (see also Putman 1991). The standard has four parts:

- *Part 1: Reference,* containing a motivational overview of the standard and its concepts (ITU 1996).
- *Part 2: Foundations,* defining the concepts, the analytical framework for the description of ODP systems, and a general framework for assessment and conformance (ITU 1995a).
- *Part 3: Architecture*, describing the ODP framework of viewpoints for the specification of ODP systems in different viewpoint languages (ITU 1995b). It identifies five viewpoints on a system and its environment: enterprise, information, computation, engineering, and technology.
- *Part 4: Architectural semantics,* showing how the modelling concepts from Part 2 and the viewpoint languages from Part 3 can be complemented in a number of formal description techniques, such as LOTOS, Estelle, SDL, and Z (ITU 1997).

GERAM The Generic Enterprise Reference Architecture and Methodology (GERAM) (IFIP-IFAC Task Force 1999) defines the enterprise-related generic concepts recommended for use in enterprise engineering and integration projects. These concepts can be categorised as:

- *Human-oriented concepts* to describe the role of humans as an integral part of the organisation and operation of an enterprise and to support humans during enterprise design, construction, and change.
- *Process-oriented concepts* for the description of the business processes of the enterprise;
- *Technology-oriented concepts* for the description of the supporting technology involved in both enterprise operation and enterprise engineering efforts (modelling and model use support).

The model proposed by GERAM has three dimensions: the life cycle dimension, the instantiation dimension allowing for different levels of controlled particularisation, and the view dimension with four views: Entity Model Content view, Entity Purpose view, Entity Implementation view, and Entity Physical Manifestation view. Each view is further refined and might have a number of components.

Nolan Norton Framework (Zee et al. 2000) This framework is the result of a research project of the Nolan Norton Institute (which involved 17 Dutch large companies) on current practice in the field of architectural development. Based on the information collected from companies the authors have defined a five-perspective vision of enterprise architecture:

- *Content and goals*: which type of architecture is developed, what are its components and the relationships between them, what goals and requirements has the architecture to meet? More precisely, this perspective consists of five interconnected architectures (they correspond to what we have called architectural views): product architecture, process architecture, organisation architecture, functional information-architecture, and technical information architecture.
- *Architecture development process*: what are the different phases in the development of an architecture, what is their sequence and what components have to be developed in each phase?
- *Architecture process operation*: what are the reasons for the change, what information is needed, and where do the responsibilities lie for decision making?
- *Architectural competencies*: what level of expertise should the organisation reach (and how) in order to develop, implement, and use an architecture?
- *Cost/Benefits:* what are the costs and benefits of developing a new architecture?

2.3 Description Languages

In subdomains such as business process design and software development, we find established description languages for modelling these domains. For software modelling, UML (described in Sect. 2.3.3) is of course the single dominant language. In organisation and process modelling, on the other hand, a multitude of languages are in use: there is no standard for models in this domain.

Here, we describe a number of languages for modelling business and IT We do not describe 'languages' that are merely abstract collections of concepts, such as the RM-ODP viewpoint languages, but focus on languages that either find widespread use or have properties that are interesting from the perspective of our goals in developing an enterprise architecture language.

2.3.1 IDEF

IDEF is the name of a family of languages used to perform enterprise modelling and analysis (see http://www.idef.com/ and Mayer et al. 1995; IDEF 1993; Menzel and Mayer 1998). The IDEF (Integrated Computer-Aided Manufacturing (ICAM) DEFinition) group of methods have a military background. Originally, they were developed by the US Air Force Program for Integrated Computer Aided Manufacturing (ICAM). The numbers of participants in the meetings of the IDEF user group are evidence of the widespread usage of IDEF.

Currently, there are 16 I.E. methods. Of these methods, IDEF0, IDEF3, and IDEF1X ('the core') are the most commonly used. Their scope covers:

- Functional modelling, IDEF0: The idea behind IDEF0 is to model the elements controlling the execution of a function, the actors performing the function, the objects or data consumed and produced by the function, and the relationships between business functions (shared resources and dependencies).
- Process modelling, IDEF3: IDEF3 captures the workflow of a business process via process flow diagrams. These show the task sequence for processes performed by the organisation, the decision logic, describe different scenarios for performing the same business functions, and enable the analysis and improvement of the workflow.
- Data modelling, IDEF1X: IDEF1X is used to create logical data models and physical data models by the means of logical model diagrams, multiple IDEF1X logical subject area diagrams, and multiple physical diagrams.

There are five elements to the IDEF0 functional model (see Fig. 2.10**): the *activity* (or process) is represented by boxes, *inputs*, *outputs, constraints*, or controls on the activities, and *mechanisms* that carry out the activity. The inputs, control, output and mechanism arrows are also referred to as ICOMs.** Each activity and the ICOMs can be decomposed (or exploded) into more detailed levels of analysis. The decomposition mechanism is also indicated as a modelling technique for units of behaviour in IDEF3.

The IDEF3 Process Description Capture Method provides a mechanism for collecting and documenting processes. There are two IDEF3 description modes: process flow diagrams and object state transition network diagrams. A process flow

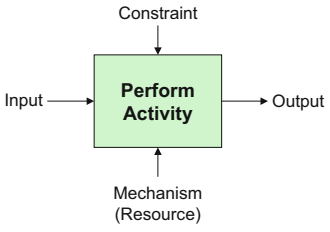

Fig. 2.10 IDEF0 representation

description captures knowledge of 'how things work' in an organisation, e.g., the description of what happens to a part as it flows through a sequence of manufacturing processes. The object state transition network description summarises the allowable transitions an object may undergo throughout a particular process. The IDEF3 term for elements represented by boxes is a Unit Of Behaviour (UOB). The arrows (links) tie the boxes (activities) together and define the logical flows. The smaller boxes define junctions that provide a mechanism for introducing logic to the flows.

The IDEF family provides support for the modelling of several architectural views. However, there are no communication mechanisms between models. The fact that they are isolated hinders the visualisation of all models as interrelated elements of an architectural system. This also means that a switch between views is not possible.

IDEF is widely used in the industry. This indicates that it satisfies the needs of the users within acceptable limits. The IDEF family is subject to a continuous process of development and improvement. Still, IDEF0, IDEF1X, and IDEF3 are rather stable and rigid languages, and IDEF0 and IDEF1X have been published as standards of the National Institute of Standards and Technology.

2.3.2 BPMN

The Business Process Modelling Notation (BPMN) was developed by the Business Process Management Initiative (BPMI), which has since merged with the Object Management Group. The BPMN standard (Object Management Group 2013) specifies a graphical notation that serves as a common basis for a variety of business process modelling and execution languages.

As the name already indicates, BPMN is restricted to process modelling; applications or infrastructure are not covered by the language. The main purpose of BPMN is to provide a uniform notation for modelling business processes in terms of activities and their relationships (Fig. 2.11).

The first version of BPMN only defined a concrete syntax, i.e., a uniform (graphical) notation for business process modelling concepts. However, there is a formal mapping to the XML-based business process execution language WS-BPEL. BPMN 2 (Object Management Group 2013) provides a semantics for execution of these models, and many business process management tool suites now support this.

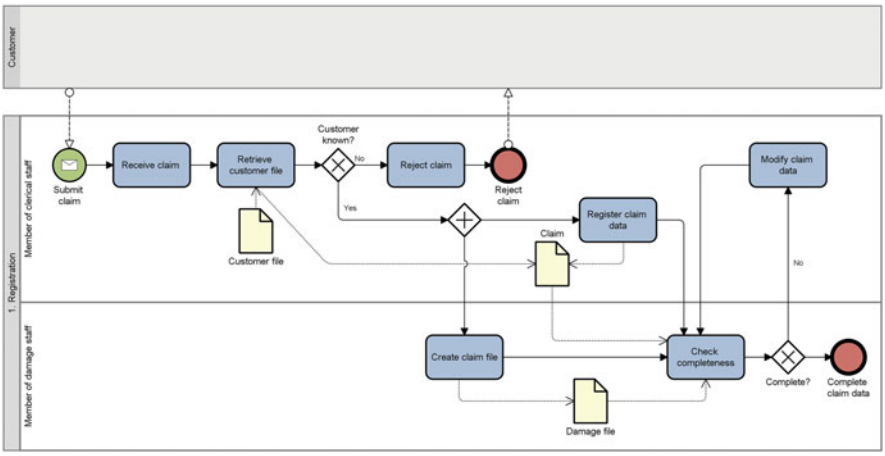

Fig. 2.11 Example model in BPMN

2.3.3 UML

The Unified Modeling Language (UML) (Booch et al. 1999; Object Management Group 2015a) is currently the most important industry-standard language for specifying, visualising, constructing and documenting the artefacts of software systems. The language's development is managed by the Object Management Group (OMG). It emerged from the combination of three existing notations, Booch, OMT, and Objectory, authored by the 'three amigos' Booch, Rumbaugh, and Jacobson. Other influences came from Harel's state charts and Shlaer-Mellor's object life cycles.

UML is intended to be used by system designers. Consequently, UML models are only clear to those who have a sound background in computer science, in particular in object orientation (see Fowler and Scott 1999). However, leaving out the more technical details, UML models should be sufficiently understandable for illustrative and explanatory purposes to business engineers and organisation specialists. Although UML was originally developed for the design of object-oriented software, its use has expanded to other areas, including architecture modelling. In its current version, UML 2 (Object Management Group 2015a), several architectural concepts are included.

Through object orientation, UML covers all possible modelling domains one can think of. From the point of view of UML, the world consists of only one kind of component-like thing, called *object*, together with a connection-like thing, called *link*. Examples of objects are persons, organisational units, products, projects, archives, and machines. The objects consist of a static part and a dynamic part. The dynamic part is a description of *how* such an object does what it should do.

The links reflect any kind of connection or relation between objects, varying from concrete ('is-boss-of') to abstract ('might-be-relevant-for'). In this way links

can express relations, connections, dependencies, relevancies of a physical, logical, temporal, structural, behavioural, similar, or complementary character, to mention a few examples.

UML is a disturbingly rich combination of 13 sublanguages each having its own (sub)scope of the complete UML scope, and each with its own diagram to model a specific aspect of a (software) system. The 13 diagrams can be grouped in three categories:

- structure: package diagrams, class diagrams, object diagrams, composite structure diagrams;
- behaviour: use case diagrams, state diagrams, sequence diagrams, timing diagrams, communication diagrams, activity diagrams, interaction overview diagrams;
- implementation: component diagrams, deployment diagrams.

Each diagram type describes a system or parts of it from a certain point of view, and contains its own symbols. However, the diagram types and UML meta-model are interrelated; no strict separation between views and meta-model concepts has been made. Consequently, the relations between modelling concepts in different diagrams are often ill-defined. We will not show the notation of all these diagrams and modelling concepts here; a good overview is given in Fowler and Scott (1999).

Moreover, apart from the package, component, and deployment diagrams, each of the other languages is in itself a disturbingly rich combination of visual building blocks. Some of these languages have large mutual overlap, e.g., activity diagrams and state chart diagrams. The advantage of such richness is the expressiveness of the language; a serious disadvantage is the readability and accessibility of the language. The large numbers of symbols and diagrams make the learning curve of UML pretty steep for new users.

Next to the graphical notation, UML contains the Object Constraint Language (OCL), a textual language for specifying constraints on model elements. The meaning of UML diagrams is not always very intuitive and sometimes requires quite careful study. For an experienced UML user, however, the language is not too difficult to use.

To extend the modelling vocabulary or give distinctive visual cues to certain kinds of abstractions that often appear, UML offers three kinds of mechanisms:

- A ‘stereotype’ is an extension of the vocabulary of UML that allows the creation of new kinds of building blocks, based on existing ones. A stereotype is used to define specialisations of existing elements of UML meta-model.
- A ‘stereotype attribute’ is an extension of the properties of a UML element that allows the creation of new information in that element’s specification. Stereotype attributes can be added to all existing meta-model elements.
- UML offers the possibility to define so-called profiles attuned to certain problem domains. A profile is a kind of dialect of the original modelling language, better suited to reflect the characteristics of a certain problem domain. A profile uses tagged values and stereotypes to express a specific and precise model.

Although these extension mechanisms give UML considerable flexibility, they also are a weak point of the language. Stereotypes, especially when applied too much, can confuse readers who are not familiar with them. In such cases stereotypes take away one of the strong points of UML, which is standardisation.

UML partially has a formal basis. Semantics for individual diagram types exist, in a more or less formal manner. However, a formalised integrated semantics for the whole language is still lacking. This lack of an integrated semantics makes it difficult to define rigorous analysis techniques.

Perhaps UML's most important asset is its broad tool support: there are many commercial as well as public domain modelling environments. As many of these environments offer means to translate a model into executable code, e.g. Java, some form of analysis is being provided: through the execution. Often also other means of analysis and verification are being provided, through partial consistency checking, or forms of animation or explicit translation to a different domain where a particular verification can be performed.

2.3.4 Architecture Description Languages

The term 'Architecture Description Language' (ADL) is used to refer to a (usually formal) language to describe a software architecture in rather general terms. A wide variety of ADLs exist, with several differences in the exact concepts that they offer: some focus on structural aspects of an architecture, while others pay more attention to the dynamic aspects. In general, their concepts are defined at a rather generic level: although they are usually intended for modelling the application level, the use of the concepts is not restricted to this. As a result of this high abstraction level, constructing and reading ADL specifications may be difficult for non-expert users. An advantage is the precise definition and formal foundation of the languages, which may make them suitable as an underlying language for more specific concepts. In Medvidovic and Taylor (2000) the basics of ADLs are described, and a large number of ADLs are compared.

Although the concepts used in ADLs are very generic, they are mainly applied in the field of software architecture. In addition to ADLs with a general applicability, there are ADLs with a much more specific application area (e.g., MetaH, for the guidance, navigation, and control domain). Because of the formal nature and high abstraction level of the concepts, ADLs are mainly suitable for users with a technical background. They are unsuitable as a means for communication at the organisational level.

In principle, ADL concepts are sufficiently flexible to create models in several domains. However, they are mainly applied, and are most suitable, for the application domain (i.e., to describe software architectures). As Acme (1998) is claimed to be suitable as a general architecture description and interchange language, we believe its concepts can be used as a representative for ADLs. The core concepts are:

- component;
- connector;
- system (a configuration of components and connectors);
- port (a point of interaction with a component);
- role (a point of interaction with a connector);
- representation (used to model hierarchical composition);
- rep-map (which maps a composite component or connector's internal architecture to elements of its external interface).

ADLs like Acme generally have an academic background, and limited usage. However, some of these concepts have been included in UML and SysML (Object Management Group 2015e). In this way, these concepts are made available to a large user base and will be supported by a wide range of software tools.

2.3.5 Suitability for Enterprise Architecture

In the previous sections, we have given an overview of several languages for modelling in the area of organisations, business processes, applications and technology. It is clear that none of these has succeeded in becoming 'the language' that can cover all domains. In general, there are a number of aspects on which almost all of these languages score low:

- The relations between domains (views) is poorly defined, and the models created in different views are not further integrated.
- Most languages have a weak formal basis and lack a clearly defined semantics.
- Most languages miss the overall architectural vision and are confined to either the business or the application and technology subdomains.

In contrast to organisation and business process modelling, for which there is no single dominant language, in modelling applications and technology UML has become a true world standard. UML is the mainstream modelling approach within ICT. This makes UML an important language not only for modelling software systems, but also for business processes and for the general business architecture. However, UML is not readily accessible and understandable for managers and business consultants; therefore, special visualisations and views of UML models should be provided.

2.4 Service-Oriented Architecture

The emergence of the service-oriented computing (SOC) paradigm and Web services technology, in particular, has aroused enormous interest in service-oriented architecture (SOA). Probably because such hype has been created around it, there

are a lot of misconceptions about what SOA really is. Numerous Web services evangelists make us believe that if you could divide the world into service requestors, service providers and a service registry, you would have an SOA (e.g., Ferris and Farrell 2003). Others emphasise that SOA is a way to achieve interoperability between distributed and heterogeneous software components, a platform for distributed computing (e.g., Stevens 2002).

Even though dynamic discovery and interoperability are important benefits of Web services, a purely technological focus would be too limited and would fail to appreciate the value of the (much more general) service concept. SOA represents a set of design principles that enable units of functionality to be provided and consumed as services. The interesting thing is that the service concept applies equally well to the business as it does to software applications. Services provide the 'units of business' that represent value propositions within a value chain or within business processes. This essentially simple concept can and should be used not just in software engineering, but also at all other levels of the enterprise architecture, to achieve ultimate flexibility in business and IT design.

The idea of systems (applications or components) delivering services to other systems and their users has really caught on in software engineering. Moreover, in other relevant disciplines there is also an increasing focus on services. In fact, economic development is to an increasing extent driven by services, not only in traditional service companies but also in manufacturing companies and among public service providers (Illeris 1997). In the service economy, enterprises no longer convert raw materials into finished goods, but they deliver services to their customers by combining and adding value to bought-in services. As a consequence, management and marketing literature is increasingly focusing on service design, service management, and service innovation (e.g., see Fitzsimmons and Fitzsimmons 2000, or Goldstein et al. 2002).

Another area in which the service concept plays a central role is IT service management. This discipline is aimed at improving the quality of IT services and the synchronisation of these services with the needs of their users (Bon 2002). The ITIL approach described in Sect. 2.1.6, for example, puts great emphasis on services and service-level agreements.

The service concept is the result of a separation of the 'external' and 'internal' behaviour of a system. As such, it should be self-contained and have a clear purpose from the perspective of its environment. The internal behaviour, on the other hand, represents what is required to realise this service. For the 'consumers' of a service, the internal behaviour of a system or organisation is usually irrelevant: they are only interested in the functionality and quality that will be provided.

2.4.1 Service-Oriented Technologies

Web services are a large body of industry standards developed and managed by organisations such as W3C, UN-CEFACT, OMG, The Open Group and OASIS.

Next to these 'classical' and rather heavy-weight standards-based Web services, lighter service-oriented protocols based on Representational State Transfer (REST) have become widely used.

A parallel development in service orientation is the ability to access ICT resources, such as computing power, storage capacity, devices, and applications as services over the Internet. This provisioning of commoditised computing and storage capabilities over the Internet is collectively called Cloud Computing, with Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) as important categories. This gives large and small organisations access to ICT resources otherwise out of reach and provides advantages regarding cost and scalability. This development has its origin in e-science environments (computing grids), but has found extensive usage for a variety of other application areas like healthcare, education, finance, life sciences, industry and entertainment.

These service developments strengthen the impact of service orientation on business architectures, because they extend the application of service-oriented technology to the domain of utility computing and ASP, while its focus on sharing of ICT resources has additional impact on the way ICT infrastructure services are managed within organisations.

Several tool vendors recognise the importance of integrating real-time IT service management with operational business processes and customer services. They provide tools that propagate events at the IT level to process owners and customers; conversely, problem reports from users and customers can be propagated to the IT service level. Such integration should offer operational business–IT alignment giving insight into real-time performance and service levels. These developments create a strong case for service-oriented methods, since they apply service orientation in real-time operational service management allowing services to be used for on-line decision making and problem solving.

2.4.2 Relevance and Benefits for Enterprise Architecture

One might ask why we should focus on services for architecting the enterprise and its IT support. What makes the service concept so appealing for enterprise architecture practice? First, there is the fact that the service concept is used and understood in the different domains making up an enterprise. In using the service concept, the business and IT people have a mutually understandable 'language', which facilitates their communication. Second, service orientation has a positive effect on a number of key differentiators in current and future competitive markets, i.e., interoperability, flexibility, cost effectiveness, and innovation power.

Of course, Web services and the accompanying open, XML-based standards are heralded for delivering true interoperability at the information technology level (Stevens 2002). However, service orientation also promotes interoperability at higher semantic levels by minimising the requirements for shared understanding: a service description and a protocol of collaboration and negotiation are the only

requirements for shared understanding between a service provider and a service user. Therefore, services may be used by parties different from the ones originally perceived, or used by invoking processes at various aggregation levels.

Interoperability and separation of internal and external behaviour provide new dimensions of flexibility: flexibility to replace or substitute services in cases of failure, flexibility to upgrade or change services without affecting the enterprise's operations, flexibility to change suppliers of services, flexibility to reuse existing services for the provision of new products or services. This will create new opportunities for outsourcing, rendering more competition and more efficient value chains.

By focusing on services, many opportunities for reuse of functionality will arise, resulting in more efficient use of existing resources. In addition, outsourcing and competition between service providers will also result in a reduction of costs. From a macroscopic point of view, costs will be reduced as a result of more efficient distribution of services in value chains.

The ability to interoperate and collaborate with different partners, including partners not familiar with the enterprise, provides new opportunities for innovation. Existing services can be recombined, yielding new products and services, ad hoc liaisons with new partners become possible that exploit emerging business opportunities, and newly developed services can easily be advertised and offered all over the world, and integrated in the overall service architecture.

Finally, service orientation stimulates new ways of thinking. Traditionally, applications are considered to support a specific business process, which in turn realises a specific business service. Service orientation allows us also to adopt a bottom-up strategy, where the business processes are just a mechanism for instantiating and commercially exploiting the lower-level services to the outside world. In this view, the most valuable assets are the capabilities to execute the lower-level services, and the business processes are merely a means of exploitation.

Some organisations have already started to implement service-oriented enterprise architectures, but the future will determine whether service orientation really can deliver on all its promises of increased interoperability, flexibility, and innovation power.