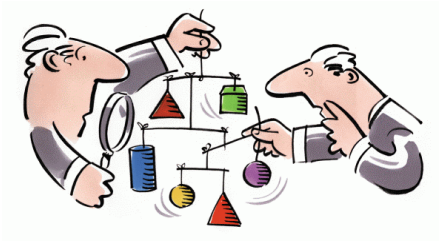
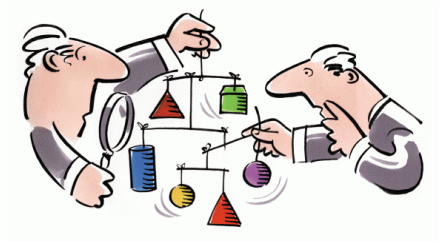


# State of the Art in Architecture Concepts and Description

*ArchiMate Deliverable D2.1*



# Colophon


| | |
|---|---|
| Date : | 20-12-2002 |
| Version : | 1.0 |
| Change : | Finalized |
| Project reference: | ArchiMate/D2.1 |
| TI reference : | TI/RS/2002/113 |
| Company reference : | |
| URL : | https://doc.telin.nl/dscgi/ds.py/Get/File-27883/ |
| Access permissions : | Project |
| Status : | Final |
| Editor : | Henk Jonkers |
| Company : | Telematica Instituut (TI) |
| | Leiden Institute of Advanced Computer Science (LIACS) |
| Authors : | Maria-Eugenia Iacob (TI) |
| | Henk Jonkers (TI) |
| | Marc Lankhorst (TI) |
| | René van Buuren (TI) |
| | Luuk Groenewegen (LIACS) |
| | Kok Hing Cheung (LIACS) |
| | Marcello Bonsangue (LIACS) |
| | Niels van Kampenhout (LIACS) |



**Synopsis:**

*This document is concerned with current practice in architectural modelling of organisations and applications, either in separation or combined. Special attention is paid to how the elements of existing frameworks and languages can be used to help to reach the goals of the project.*

# ArchiMate

Organisations need to adapt increasingly fast and flexible to changing customer requirements and business goals. This need influences the entire chain of activities of a business, from the organisational structure to the network infrastructure. How can you control the impact of these changes? Architecture may be the answer. The ArchiMate project will develop an integrated architectural approach that describes and visualises the different business domains and their relations. Using these integrated architectures aids stakeholders in assessing the impact of design choices and changes.

Architecture is a consistent whole of principles, methods and models that are used in the design and realisation of organisational structure, business processes, information systems, and infrastructure. However, these domains are not approached in an integrated way, which makes it difficult to judge the effects of proposed changes. Every domain speaks its own language, draws its own models, and uses its own techniques and tools. Communication and decision making across domains is seriously impaired.

The goal of the ArchiMate project is to provide this integration. By developing an architecture language and visualisation techniques that picture these domains and their relations, ArchiMate will provide the architect with instruments that support and improve the architecture process. Existing and emerging standards will be used or integrated whenever possible. ArchiMate will actively participate in national and international forums and standardisation organisations, to promote the dissemination of project results.

The project will deliver a number of results. First of all, we will provide a visual design language with adequate concepts for specifying interrelated architectures, and specific viewpoints for selected stakeholders. This will be accompanied by a collection of best practices and guidelines. Furthermore, ArchiMate will develop techniques that support architects in visualisation and analysis of architectures. Finally, project results will be validated in real-life cases within the participating organisations.

To have a real impact on the state of the art in architecture, the ArchiMate project consists of a broad consortium from industry and academia. ArchiMate's business partners are ABN AMRO, Stichting Pensioenfonds ABP, and the Dutch Tax and Customs Administration (Belastingdienst); its knowledge partners are Telematica Instituut, Ordina Institute, Centrum voor Wiskunde en Informatica (CWI), Leiden Institute for Advanced Computer Science (LIACS), and Katholieke Universiteit Nijmegen (KUN).

More information on ArchiMate and its results can be obtained from the project manager Marc Lankhorst (Marc.Lankhorst@telin.nl) or from the project website, archimate.telin.nl.

# Table of Contents

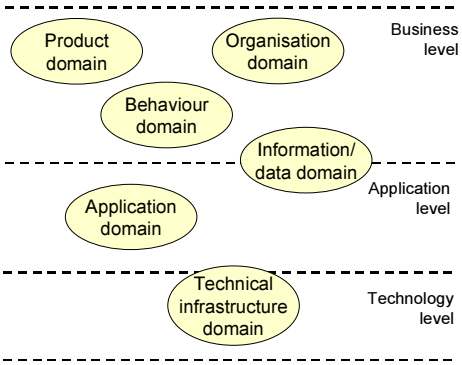
# Management summary

This report surveys the current state of the art in architectural modelling of organisations and applications, either in separation or combined. The topics include architectural *frameworks*, languages for *organisation and process modelling*, and languages for *application and technology modelling*. Although there is a trend towards considering the relationship between the organisational processes and the information systems and applications that support them (often referred to as “business-IT alignment), modelling techniques to really express this relationship hardly exist yet.

Frameworks and languages are not described in full detail, but their basic role and context are given. The large number of related concepts does not leave much space for extensive discussions, but we provide references to more detailed information, both in this document and in our web-based application – The ArchiMate Resource Tree (see Section 1.3). We have also refrained ourselves from doing detailed comparisons between the subjects we have surveyed. However, we have tried to digest the major issues and trends from the information we had at our disposal, and to give our opinion on the relevance of these subjects for ArchiMate.

### Frameworks

Frameworks structure architectural description techniques by identifying and relating different architectural viewpoints and the modelling techniques associated with them. They typically define a number of conceptual domains or aspects to be described. The figure below shows a number of domains that are often distinguished, divided in three ‘levels’. Frameworks do not provide the concepts for the actual modelling, although some frameworks are closely connected to a specific modelling language or set of languages. Many frameworks are associated with a design method. A *metamodel* (formally) defines the concepts of a language and their relationships. As a metamodel at a certain level of abstraction may also define the conceptual domains covered by a language, it could be considered an extension of a framework.




Well-known examples of architectural frameworks are:

- Zachman's "framework for enterprise architecture" is widely known and used. It identifies 30 or 36 views on architecture ("cells"), based on five or six levels (scope, enterprise, logical system, technology, detailed representations and functioning enterprise) and six aspects (data, function, network, people, time, motivation). The large number of views is an obstacle for the practical applicability of the framework.
- The Reference Model for Open Distributed Processing (RM-ODP) is an ISO/ITU Standard, which defines a framework for architecture specification of large distributed systems. It defines, among others, five viewpoints on a system and its environment: enterprise, information, computation, engineering and technology.
- The architectural framework of The Open Group (TOGAF) is completely incorporated in the TOGAF methodology. TOGAF has four main components, one of which is a high-level framework defining four views: Business Architecture, Data/information architecture, Application Architecture and Technology Architecture.

### Organisation and process modelling languages

A wide variety of organisation and process modelling languages are in use: there is no standard for models in this domain. The conceptual domains that are covered differ from language to language. In many languages, the relations between domains are not clearly defined. Also, most languages are not really suitable to describe *architectures*: they provide concepts to model, e.g., detailed business processes, but not the high-level relationships between different processes. Software tools are an important success factor for a language; some of the most popular languages are proprietary to a specific tool. Relevant languages in this category include:

- Amber is the business process and organisation modelling language in Testbed Studio, a tool used by among others Belastingdienst and ABP. Amber models appear to be easy to use and understand, also for non-expert users.
- IDEF, originating from the US Ministry of Defence, is a collection of 16 diagramming techniques. However, only three of them are widely used: IFEF0 (function modelling), IDEF1/IDEF1x (information/data modelling) and IDEF3 (process description).
- ARIS ("Architecture of Integrated Information Systems") has an academic origin, but is now part of the widely used ARIS Toolset. Although, in principle, ARIS also covers other conceptual domains, there is a clear focus on business process and organisation modelling.

### Application and technology modelling languages

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, and its use is expanding into other areas. This makes UML an important language not only for modelling software systems, but also for business processes and for the general business architecture. UML has either incorporated or superseded most of the older ICT modelling techniques still in use.

However, UML is not easily accessible and understandable for managers and organisational specialists; therefore, special visualisations and views of UML models should be provided. Given the importance of UML, other modelling languages will likely provide an

interface or mapping to it. This is also advisable to ArchiMate; it should be possible to describe the ArchiMate concepts in UML, or to map them to UML by using e.g. stereotypes and profiles.

Architecture description languages (ADLs) define high-level concepts for architecture description, such as components and connectors. Most of them have an academic background, and their application in practice is limited. However, they have a sound formal foundation, which makes them suitable for unambiguous specifications and amenable to different types of analysis. The next version of UML (UML 2.0), will likely comprise more concepts on the architectural level as well, drawing inspiration from ADLs. This obviates the need for a separate ADL for modelling software systems.

Although UML provides some (limited) support for modelling technical infrastructure, well-defined modelling languages at this level are nearly non-existent.

# 1 Introduction

This document is concerned with current practice in architectural modelling of organisations and applications, either in separation or combined. Figure 1-2 gives an overview of topics covered and their relationships. We pay special attention to how the elements of existing frameworks and languages can be used to help to reach the goals of the ArchiMate project.

## 1.1 Target group

This state of the art is intended for domain architects, enterprise architects, project managers of projects that use and create architecture, and for their managers (senior management). The goal of this report is to inform ArchiMate participants about the current state of the art in architectural frameworks and description techniques, and their relevance for ArchiMate.

## 1.2 Position of this work in ArchiMate

Because architectures are often complex and hard to understand, architects need ways to express these architectures as clearly as possible: both for their own understanding and for communication with other stakeholders, such as system developers, end-users and managers. To date, there is no standard language for describing architectures; they are often described in informal pictures that lack a well-defined meaning. This leads to misunderstandings, and makes it very difficult to provide tools for visualisation and analysis of these architectures.

Figure 1-1 illustrates the position of models within the scope of the project. Models described in a common ArchiMate language are the basis for different types of visualisation and analysis, which are the primary means for stakeholder communication. Different models and descriptions currently in use by architects, both at the business level and the application level, can be either mapped onto the common language or linked to the ArchiMate models.

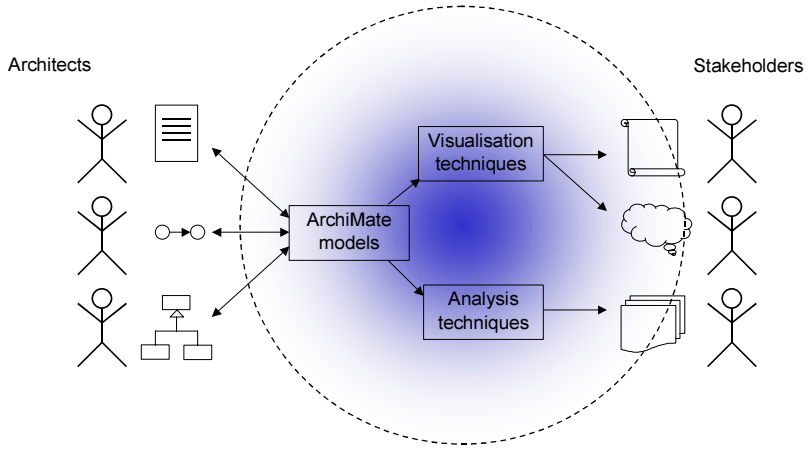



Figure 1-1. Scope of the ArchiMate project

Because of this central position of the ArchiMate language amidst other languages, it is important to have a clear overview of the different languages used by architects. The most

useful modelling concepts of these languages, or generalisations thereof, are incorporated in our language. Also, the ArchiMate language should be sufficiently generic and flexible to allow for mappings to the concepts of other languages. To obtain this overview, this report presents the current state of the art in architecture description, and as such it is the starting point for the conceptual work in ArchiMate. In particular, it is input for the definition of concepts (deliverable D2.2.1), as well as their representation and viewpoints (deliverable D2.2.2) and the mapping to partner-specific concepts (deliverable D2.2.3).

## 1.3 Working process

The work compiled here is the result of a two-stage process. In the first stage the basic material was gathered and assembled, in the second stage this report was created and supplemented with a management summary and an introduction. The basic material will be made available in a web-based application (called the ArchiMate Resource Tree - ART) combined with the basic material collected for the State of the Art in Architecture Concepts and Descriptions (D3.1).

## 1.4 How to read this document

From the very start, we have to stress that this document (and especially the web-based application the ArchiMate Resource Tree) should be regarded more as an “architecture encyclopaedia”. Our goal was to make available, in very condensed manner, information (and proper references to more detailed information) about a large number of topics that will be beneficial not only for target group of this document, but also for the research work in all the tasks within ArchiMate. Apart from the fact that the topics and the topic-related information were selected to meet the specific needs of ArchiMate (as stated in the project plan), we have also tried to express our opinion about the value and relevance of this information for the project. However, since many of the topics are indirectly related, there is a high risk that, while reading this document from the beginning to the end, the reader might miss the continuity of the discourse, and might find some of the subjects more interesting than others. Therefore, we recommend the reader to look at this document as a structured collection of various subjects, and possibly to select only chapters/sections (topics) that he/she finds interesting. In general, each chapter can be read in isolation from the others (this is even more obvious for ART). This also holds for the sections of one particular chapter, but in this case the reading of the introduction of that chapter should precede the reading of any of its sections.

## 1.5 Document structure

Figure 1-2 shows the topics addressed in this document and the relationships between them.

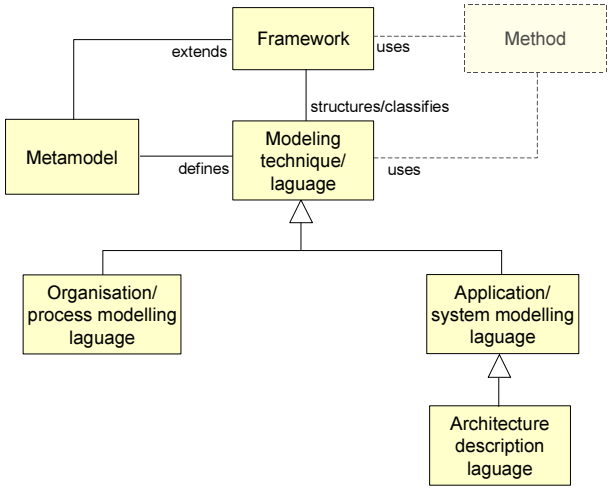



Figure 1-2. Overview of topics

*Frameworks* provide a structure to classify or compare modelling techniques. They typically define a number of conceptual domains or aspects to be described. Figure 1-3 shows a number of domains that are often distinguished, divided in three 'levels'. Many frameworks are associated with a (development) methodology (see ArchiMate deliverable D3.1 see Iacob et al. 2002). A *metamodel* (formally) defines the concepts of a language and their relationships. As a metamodel at a certain level of abstraction may also define the conceptual domains covered by a language, it could be considered an extension of a framework. The next chapter gives an overview of a number of well-known architectural frameworks.

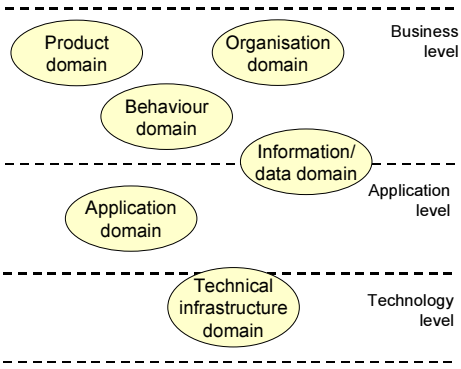



Figure 1-3. Conceptual domains

Traditionally, business modelling and software architecture are separate disciplines. Although there is a trend towards considering the relationship between the organisational processes and the information systems and applications that support them (often referred to as "business-IT alignment"), modelling techniques to really express this relationship hardly exist yet.

Therefore, we describe the two classes of modelling techniques in two separate chapters. Chapter 3 focuses on organisation and process modelling languages, while Chapter 4 considers application and technology modelling languages. Although UML, the dominant language in the latter category, is also claimed to be suitable for business process modelling, its practical application is still very much restricted to the software domain.

For each language that we describe, the aspects that we take into account include:

- Their background, scope and current usage.
- Their concepts and the conceptual domains (see Figure 1-3) that they cover.
- The structuring mechanisms that they provide. Structuring mechanisms are essential for readable, scalable models, and it can be argued that they form one of the most important properties of architectures.
- The availability of a formal basis, which is needed for an unambiguous meaning of models, as well as for analysis, visualisation and mappings to other languages.
- Their support by means of methods and software tools.

# 2 Architectural frameworks and standards

Frameworks structure architectural 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. Many frameworks have a close link with a design method.

This chapter gives an overview of the available architecture frameworks. We are going to present in more detail the ones that we consider of higher relevance for ArchiMate (the Zachman Framework, RM-ODP, OMG-MDA, RSD and TOGAF). For the others (4+1, Nolan Norton, GERAM, C$^4$ISR) one can refer to Section 2.6 for brief descriptions. More information about all these frameworks the reader can find in ART. Finally, in Section 2.7 we give a framework summary overview (in the form of Table 2-2).

## 2.1 Zachman's framework

In 1987, Zachman introduced his "*Framework for Enterprise Architecture*" (see 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.

ENTERPRISE ARCHITECTURE - A FRAMEWORK ™

| | DATA *What* | FUNCTION *How* | NETWORK *Where* | PEOPLE *Who* | TIME *When* | MOTIVATION *Why* | |
|---|---|---|---|---|---|---|---|
| SCOPE (CONTEXTUAL) *Planner* | List of Things Important to the Business<br>ENTITY = Class of Business Thing | List of Processes the Business Performs<br>Function = Class of Business Process | List of Locations in which the Business Operates<br>Node = Major Business Location | List of Organizations Important to the Business<br>People = Major Organizations | List of Events Significant to the Business<br>Time = Major Business Event | List of Business Goals/Strat<br>Ends/Means=Major Bus. Goal/ Critical Success Factor | SCOPE (CONTEXTUAL) *Planner* |
| ENTERPRISE MODEL (CONCEPTUAL) *Owner* | e.g. Semantic Model<br>Ent = Business Entity<br>Reln = Business Relationship | e.g. Business Process Model<br>Proc. = Business Process<br>I/O = Business Resources | e.g. Business Logistics System<br>Node = Business Location<br>Link = Business Linkage | e.g. Work Flow Model<br>People = Organization Unit<br>Work = Work Product | e.g. Master Schedule<br>Time = Business Event<br>Cycle = Business Cycle | e.g. Business Plan<br>End = Business Objective<br>Means = Business Strategy | ENTERPRISE MODEL (CONCEPTUAL) *Owner* |
| SYSTEM MODEL (LOGICAL) *Designer* | e.g. Logical Data Model<br>Ent = Data Entity<br>Reln = Data Relationship | e.g. Application Architecture<br>Proc. = Application Function<br>I/O = User Views | e.g. Distributed System Architecture<br>Node = I/S Function (Processor, Storage, etc)<br>Link = Line Characteristics | e.g. Human Interface Architecture<br>People = Role<br>Work = Deliverable | e.g. Processing Structure<br>Time = System Event<br>Cycle = Processing Cycle | e.g., Business Rule Model<br>End = Structural Assertion<br>Means =Action Assertion | SYSTEM MODEL (LOGICAL) *Designer* |
| TECHNOLOGY MODEL (PHYSICAL) *Builder* | e.g. Physical Data Model<br>Ent = Segment/Table/etc.<br>Reln = Pointer/Key/etc. | e.g. System Design<br>Proc.= Computer Function<br>I/O = Data Elements/Sets | e.g. Technology Architecture<br>Node = Hardware/System Software<br>Link = Line Specifications | e.g. Presentation Architecture<br>People = User<br>Work = Screen Format | e.g. Control Structure<br>Time = Execute<br>Cycle = Component Cycle | e.g. Rule Design<br>End = Condition<br>Means = Action | TECHNOLOGY MODEL (PHYSICAL) *Builder* |
| DETAILED REPRESEN-TATIONS (OUT-OF-CONTEXT) *Sub-Contractor* | e.g. Data Definition<br>Ent = Field<br>Reln = Address | e.g. Program<br>Proc.= Language Stmt<br>I/O = Control Block | e.g. Network Architecture<br>Node = Addresses<br>Link = Protocols | e.g. Security Architecture<br>People = Identity<br>Work = Job | e.g. Timing Definition<br>Time = Interrupt<br>Cycle = Machine Cycle | e.g. Rule Specification<br>End = Sub-condition<br>Means = Step | DETAILED REPRESEN-TATIONS (OUT-OF-CONTEXT) *Sub-Contractor* |
| FUNCTIONING ENTERPRISE | e.g. DATA | e.g. FUNCTION | e.g. NETWORK | e.g. ORGANIZATION | e.g. SCHEDULE | e.g. STRATEGY | FUNCTIONING ENTERPRISE |

John A. Zachman, Zachman International (810) 231-0531

Figure 2-1: The Zachman Framework.

The framework (Figure 2-1) 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 *sub-contractor* and are included in the framework graphic that is commonly exhibited.

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 *simple* - it is easy to understand: not technical, purely logical; *comprehensive* - it addresses the enterprise as a whole and any issues can be mapped against it to understand where they fit; *neutral* - it is defined totally independently of tools or methodologies. 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 hardly specified. More about the Zachman framework can be found in Zachman (1987) and Sowa and Zachman (1997), and at the home page of the *Zachman Institute for Framework Advancement* (ZIFA, http://www.zifa.com/).

## 2.2 Reference Model for Open Distributed Processing

The Reference Model for Open Distributed Processing (RM-ODP) is an ISO/ITU Standard, which defines a framework for architecture specification of large distributed systems. The standard aims to provide support for inter-working, 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 (see 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 (see ITU 1995a).
- *Part 3: Architecture*, describing the ODP framework of viewpoints for the specification of ODP systems in different viewpoint languages (see ITU 1995b).
- *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 (see ITU 1997).

All parts of the RM-ODP are based on the object-oriented paradigm. In Figure 2-2 we present the structure of RM-ODP.

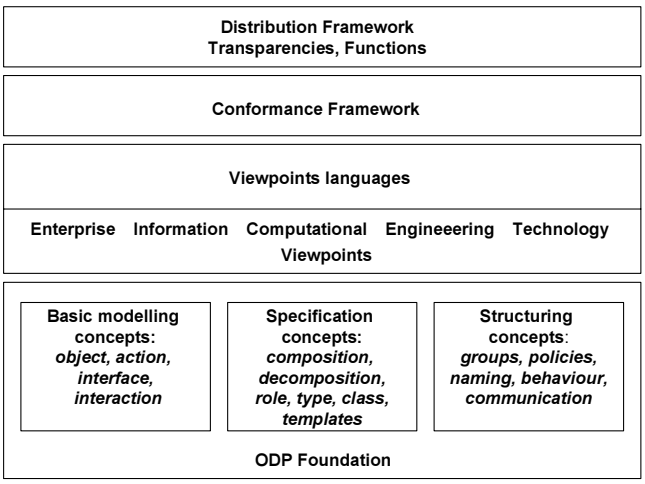



Figure 2-2. Structure of RM-ODP

The **ODP foundation** consists of a set of basic modelling concepts defining the general ODP object-model (e.g. object, action, interaction), a set of specification concepts (e.g. composition, role, type, class) and a set of structuring concepts addressing recurrent structures in ODP systems (e.g. groups, policies, behaviour).

Part 3 of RM-ODP defines five **viewpoints** on the system and its environment. A summary of the ODP viewpoints is presented in Table 2-1.

| Viewpoint | Enterprise | Information | Computational | Engineering | Technology |
|---|---|---|---|---|---|
| **Focus** | Enterprise needs with respect to IS | Information models, structures, flows, manipulation | Logical structuring of applications, components, interfaces and interactions; service oriented view of distributed applications | Distributed platform infrastructure, distribution transparency, communication support, system-oriented view of distributed applications | Technological artifacts for the underlying supporting infrastructure of the engineering mechanisms |
| **Main concepts** | Agents, artifacts, communities, contracts, roles | Invariant, static and dynamic schemas, relations, integrity, roles, etc. | Computational object, computational interface, operation stream, signal, actions | Channels, clusters, capsules, nodes, basic engineering objects, protocol object, nucleus | Technological solutions |
| **Language/ Notation** | UML use case diagrams, activity diagrams, stereotyped class diagrams | UML type/class diagrams, OCL, Entity relationships models, conceptual schemas | UML collaboration diagrams, OCL, Application and programming environments | UML collaboration diagrams, deployment diagrams, distributed platforms | Technology mappings |
| **Targets** | Capture of requirements and early design of ODP systems | Conceptual design and information modelling | Software design and development | System design and development | Technology identification, procurement installation |

Table 2-1. RM-ODP viewpoint summary

ODP defines a framework for assessment of a systems **conformance** to its specification. The purpose is to ensure well-defined behaviour of ODP components possibly delivered from different vendors. A conformance assessment may consider the conformance between specifications and implementations and the compliance and consistency between specifications, and it is based on a set of *conformance points* that can be observed and tracked during execution.

ODP also presents a framework for defining infrastructures **supporting distribution transparencies** for applications. The goal is to mask complexity of distribution for client

applications. ODP defines the following transparencies: access transparency, failure transparency, location transparency, migration transparency, relocation transparency, replication transparency, persistence transparency, and transaction transparency.

## 2.3 Rapid Service Development framework

The Rapid Service Development (RSD) methodology defines an integrated framework for the specification and development of e-business services. The development of such services is highly complex, as it involves many different aspects ranging from high-level strategic business concerns to low-level protocol definitions. In order to deal with this complexity, the 'separation of concerns' principle is applied. The RSD framework distinguishes seven different aspect areas, called *cornerstone*s, from which models and specifications can be made. In this way, one can focus on one set of concerns at a time, resulting in a lower (perceived) complexity. The RSD methodology provides well-defined links between the cornerstones and offers an integrated framework for business-driven design of transaction services.

The seven cornerstones of the RSD are structured along two dimensions, as illustrated in Figure 2-3. A distinction is made between business-oriented models (on the left) and technology or system oriented models (on the right). Secondly, models can vary in scope or granularity. They range from high-level, broad-scope, coarse-grained models (at the top) to low-level, narrow scope, fine-grained (at the bottom).

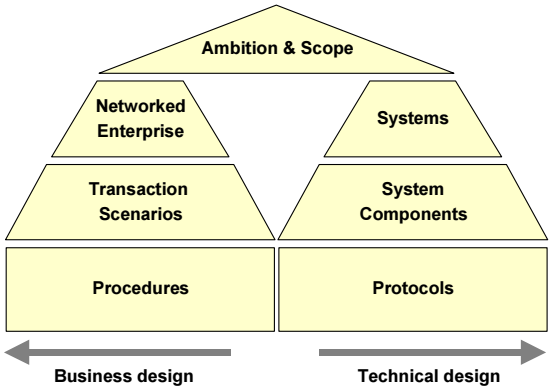



Figure 2-3. The RSD architectural framework

In addition, there is an implicit third dimension, the development dimension, ranging from analysis, through design, to realisation, which is visible in the lifecycle models proposed by RSD (see Fielt et al. 2000, Janssen and Steen 2000). The third dimension is orthogonal to the other two: development can take place in any of the cornerstones. On the business-oriented side, several types of models are considered describing the way organisations co-operate in a networked enterprise: *networked enterprise models, transaction scenarios, and procedures*. On the system-oriented side, the technology that supporting the co-operation between organisations in a networked enterprise is modelled and designed: *system descriptions, component specifications, and protocol and code specifications*.

## 2.4 OMG's Model Driven Architecture

The Model Driven Architecture (MDA) (see Architecture Board ORMSC 2001) is a trademark of the Object Management Group (OMG, http://www.omg.org/), which aims to provide an open, vendor-neutral approach to interoperability. It builds upon OMG's modelling standards: the Unified Modelling Language (UML), the Meta Object Facility (MOF) and the Common Warehouse Metamodel (CWM). Platform-independent application descriptions built with these standards can be realised using different open or proprietary platforms, such as CORBA, Java, .NET, XMI/XML and web services. Figure 2-4 illustrates how the OMG standards fit together in the MDA.

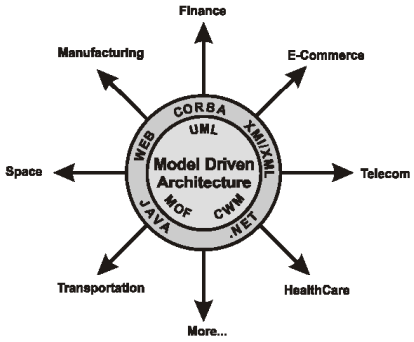



Figure 2-4. OMG's Model Driven Architecture

In the MDA, a distinction is made between *platform-independent models* (PIMs) and *platform-specific models* (PSMs). A PSM specifies, in a platform-specific way, how the functionality specified in a PIM is realised. The PIMs provide formal specifications of the structure and function of the system that abstract from the technical details. A *platform-independent component view* describes computational components and their interactions in a platform-independent manner.

UML is used as the modelling standard for both PIMs and PSMs. A complete MDA specification consists of a definitive platform-independent base UML model and one or more PSMs and interface definition sets, each describing how the base model is implemented on a different middleware platform. A complete MDA application consists of a definitive PIM, plus one or more PSMs and complete implementations, one on each platform that the application developer decides to support.

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. In most situations, however, human intervention is required. For the mapping to the different middleware platforms, UML profiles are used. An UML profile for OMG's CORBA is available, while profiles for several other platforms are under development. The UML profile for Enterprise Distributed Object Computing (EDOC) is expected to become a key element of MDA, providing a specification metamodel for enterprise systems (see section 4.1).

## 2.5 The Open Group Architectural Framework

The Open Group Architectural Framework (TOGAF) originated as a generic framework and methodology for development of technical architectures, but evolved into an enterprise architecture framework and method (http://www.opengroup.org/togaf/). Version 8 of TOGAF (which became public in December 2000) it is called the "Enterprise edition" and is dedicated to enterprise architectures.

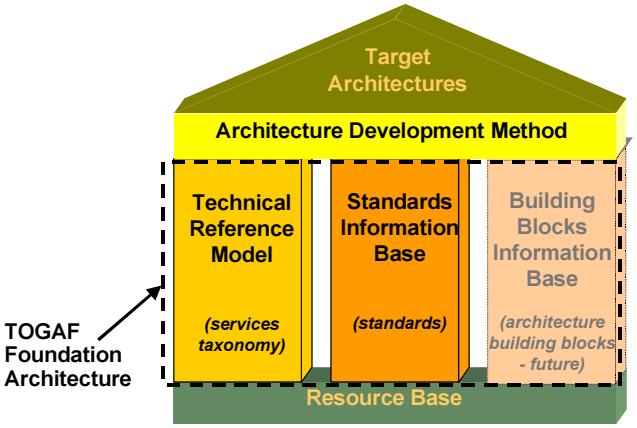



Figure 2-5. TOGAF

TOGAF has four main components:

- A high-level framework, based on some of the key concepts. The framework considers an overall Enterprise Architecture as composed of four closely interrelated Architectures: Business Architecture, Data/information Architecture, Application Architecture, and Technology (IT) Architecture.
- A methodology called Architecture Development Method (ADM), considered to be the core of TOGAF, and a step-by-step approach to developing an IT architecture.
- The TOGAF Foundation Architecture, which comprises a Technical Reference Model, The Open Group's Standards Information Base (SIB) and The Building Blocks Information Base (BBIB).
- The TOGAF Resource Base - a set of tools and techniques available for use in applying TOGAF and the TOGAF ADM (Architecture views, Business scenarios, ADML, Case studies, etc.).

The main components of the TOGAF framework are depicted in Figure 2-5. Apart from these components, TOGAF identifies a number of views, which are to be modelled in an architecture development process. The architecture views, and corresponding viewpoints fall into the following categories (the TOGAF taxonomy of views is compliant with the ANSI/IEEE Std 1471-2000, see IEEE Computer Society 2000):

- *Business Architecture Views*, which address the concerns of the users of the system, and describe the flows of business information between people and business processes (e.g. People View, Process View, Function View, Business Information View, Usability View, Performance View).
- *Technical Architecture Views*, which address the concerns of technicians responsible for developing, acquiring, and operating the system, and in turn comprise:

- *Engineering Views*, addressing the concerns of System and Software Engineers responsible for developing and integrating various components of the system (e.g. Security View, Software Engineering View, Data View, System Engineering View, Communications Engineering View).
- *Operations Views*, addressing the concerns of systems administrators and systems managers
- *Acquirers Views*, addressing the concerns of procurement personnel responsible for acquiring the commercial-off-the-shelf (COTS) software and hardware to be included in the system (e.g. The Building Blocks Cost View, The Standards View). These views typically depict building blocks of the architecture that can be purchased, and the standards that the building blocks must adhere.

## 2.6 Other frameworks

**The 4+1 View Model of Architecture:** The 4+1 View Model (Kruchten 1995) describes the architecture of software-intensive systems using five concurrent views each of which addresses a specific set of concerns:

- *Logical View*: The logical view primarily supports the functional requirements; the services the system should provide to its end users.
- *Process View*: The process view addresses topics such as non-functional requirements (e.g. performance and system availability), concurrency and distribution, system integrity, and fault-tolerance, and it specifies which thread of control executes each operation of each class identified in the logical view.
- *Development View*: The development view focuses on the organization of the software modules in the software-development environment.
- *Physical View*: The physical view is mostly devoted to the system non-functional requirements such as system availability, reliability (fault-tolerance), performance (throughput), and scalability.
- *Scenario View*: This is the "+1" view of the framework. Its purpose is to illustrate and validate how the other four views are working together using a small set of scenarios.

Architects capture their design decisions in four views and use the fifth view to illustrate and validate the other four. The definition of each view consists of a description, a notation for the description of the view blueprint, a recommended style and an example.

**Nolan Norton Framework (**Zee, Laagland, and Hafkenscheid 2000): This framework is the result of a research project of the Nolan Norton Institute (that involved 17 Dutch large companies) on the 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 the reasons for the change, what information is needed and where lies the responsibilities 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.

**GERAM**: GERA - Generic Enterprise Reference Architecture - defines the enterprise related generic concepts recommended for use in enterprise engineering and integration projects. These concepts can be categorised as:

a) 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.
b) Process oriented concepts for the description of the business processes of the enterprise;
c) Technology oriented concepts for the description of the business process 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 lifecycle dimension (see the lifecycle model of GERAM), the instantiation dimension allowing for different levels of controlled particularisation, 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 (see IFIP-IFAC 1999)

**$C^4$ISR:** The Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance ($C^4$ISR) Architecture Framework (see C4ISR Architecture Working Group 1997) was developed in 1997, for the US Department of Defence, to ensure a common unifying approach for the Commands, military Services, and Defence Agencies to follow in describing their various architectures. Although $C^4$ISR has a rather specific target, it can be extended to system architectures that are more general. $C^4$ISR 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.

## 2.7 Summary

Table 2-2 summarises the properties of the frameworks that we studied. Although there is a wide variety in, in particular, the *number* of views or domains that they distinguish, there is a large overlap in the *types* of domains that they include. For ArchiMate, the most important frameworks to consider are Zachman's framework (because of its wide acceptance) and TOGAF (because its views and ideas match with the project goals).

Table 2-2. Summary of frameworks

| Name | Purpose | Architectural Views/Viewpoints | Viewpoints representations | Main concepts | Comments | Relevance for Archimate |
|---|---|---|---|---|---|---|
| Zachman's Framework | "Framework for Enterprise Architecture" | 2-dimensional (column - aspects, rows - perspectives) 30 cell collection of viewpoints | A certain type of diagram has been associated to each cell, no languages or particular notations are specified. | aspects (data-what, function-how, network-where, people-who, time-when, motivation-why), perspectives (scope, enterprise model, system model, technology model, detailed representations), rules. | simple, comprehensive, neutral, well-known | ●●●●○ |
| RM-ODP | Framework for architecture specification of large distributed systems | enterprise, information, computational, engineering, technology | UML use-case, activity, stereotyped class, class, entity relationships, collaboration, deployment diagrams and technology mappings | conformance points, transparancies, agents, roles, communities, contracts, artifacts, relations, schemas, stream, signals, actions, channels, clusters, capsules, nodes etc. | standardised (ISO/IEC 10746-1,2,3,4), highly technical and complex, object-oriented | ●●○○○ |
| RSD | Framework for specification and development of e-business(B2B) (transaction) services | cornerstones: ambition&scope, networked enterprise, transaction scenarios, procedures, systems, system components, protocols | the networked enterprise models are organised along five domains (actors, roles, functions, processes, data), each domain having its own notation. | cornerstones, actors, roles, functions, processes, items | simple, neutral, limited to e-business | ●●●○○ |
| TOGAF | Framework for development of technical and enterprise architectures | Business Architecture Views (People, Process, Function, Business Information, Usability, Performance), Engineering Views (Security, Software, System, Communications), Operations Views, Acquirerers Views (Cost, Standards) | ADML is recommended by TOGAF as a language for the development of views | system, architecture, architecture description, views and viewpoints, model, stakeholders, concerns | comprehensive, neutral, standard compliant (IEEE 1471), well-known | ●●●●○ |
| "4+1" | Framework for software-intensive systems architectures | Logical View, Development View, Process View, Physical View, Scenario View | Each view is accompanied by a specific notation for the development of a "view blueprint". Some of the views use the Rational (Booch) notations or derivates | object, class, network, process, message flow, task, node, scenario, use-case, view, style | very much related to the Rational approach, simple, limited to software architectures, object-oriented | ●●○○○ |
| Nolan Norton | Dutch framework for enterprise architectures | 5 perspectives: Content and goals, Architecture development process, Architecture process operation, Architectural competencies, Cost/Benefits | For the product, process, organisation, and information architectures (parts of Content and Goals perpective) a 2D template is provided. | product architecture, process architecture, organisation architecture, functional information-architecture, and technical information architecture | used (known) only in The Netherlands, neutral, comprehensive | ●●○○○ |

# 3 Organisation and process modelling languages

A wide variety of organisation and process modelling languages are in use: there is no standard for models in this domain. The conceptual domains that are covered differ from language to language. Software tools are an important success factor for a language; some of the most popular languages are proprietary to a specific tool. This chapter gives an overview of the available organisation and process modelling languages. We are going to present in more detail the ones that we consider of higher relevance for ArchiMate (the Amber and NEML, IDEF, ARIS). For the others (RAD, MEMO) one can refer to Section 3.4 for brief descriptions. More information about all these frameworks can be found in ART. Finally, in Section 3.5 we give a summary overview of the surveyed languages (in the form of Table 3-1).

## 3.1 AMBER and NEML

***Background, scope, and usage:*** Amber and NEML (Networked Enterprise Modelling Language) are organisation/process modelling languages aiming to cover the business level. In Amber the focus is on a single organisation, particularly from the financial sector, while NEML is targeting inter-organisational e-business processes in networks of organisations. They are mostly suited for business consultants and intended for business process and organisation modelling. In this section we primarily describe Amber, as it is the more mature of the two. The references to NEML emphasise the differences between the two languages, or the additional features embedded in NEML.

Both languages have a research background, developed at the Telematica Instituut. The Testbed business partners, including ABP and Belastingdienst, have contributed with requirements and took part in the validation of Amber. Currently there is commercially available software, Testbed Studio, that supports Amber. NEML is also supported by a tool called RSD Studio, but this still is solely a research product. Consequently, the usage of the two languages is significantly different: Testbed Studio currently runs in a number of big and rather different Dutch companies (in terms of business model, size and structure), while RSD Studio has been used only internally within the Telematica Instituut.

***Concepts and Conceptual Domains:*** Amber recognises three aspect domains:

1. the *actor* domain: describes the resources for carrying out business activities;
2. the *behaviour* domain: describes the business processes performed by the resources;
3. the *item* domain: describes the data objects handled by business processes.

A mapping of Amber onto the domains described in the introduction (see Figure 1-3) results in the following:

- *Organisation domain*: Represented in Amber by the actor domain. Main concepts: actors (persons, organisational units), interaction points (see Figure 3-1 for the graphical representation of concepts). In NEML the organisation domain also includes the role diagrams (main concepts: actors, channels, roles and flows).

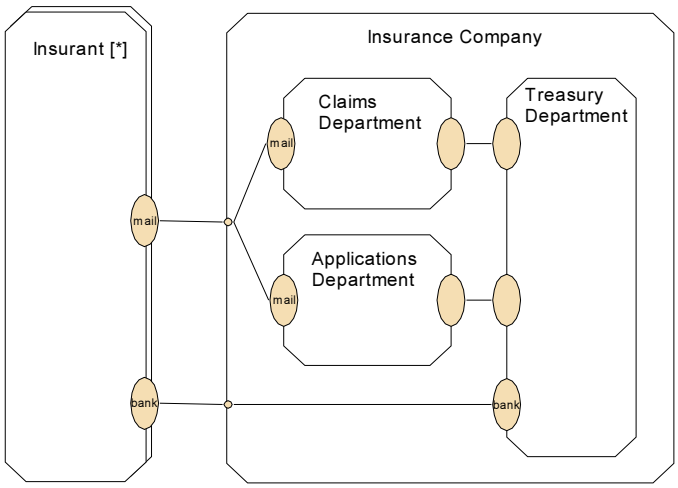



Figure 3-1. Amber actor diagram

- *Behaviour domain*: Represented in Amber by the behaviour domain. The basic concepts in the behaviour domain: actions, causality relations, and/or splits and joins, iterations, triggers, and enabling and disabling relations, behaviour blocks (processes) and interactions (see Figure 3-2 for graphical representations).

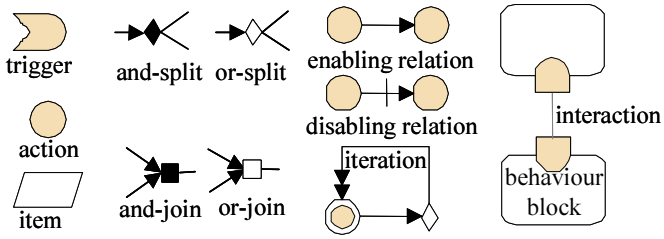



Figure 3-2. Amber basic concepts in the behaviour domain

The item concept (corresponding to data objects – e.g. databases) links the behaviour domain to the item domain. There are a number of operations that can be performed on items (see Figure 3-3).

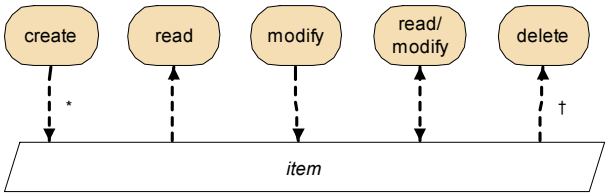



Figure 3-3. Items and actions

All these concepts are used for process modelling in the form of logical chains of event driven actions. NEML have introduced several important modelling concepts concerning behaviour, not covered by Amber: transfer (directed interaction), function, and flow. While process models in NEML are almost identical to the ones in Amber, the functional diagrams are completely new (see Figure 3-4).

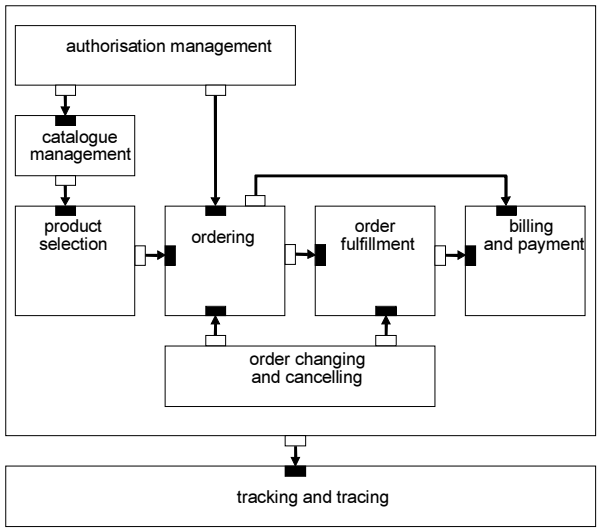



Figure 3-4. NEML function diagram

- *Information/data domain*: Represented in Amber by the Item domain. Main concepts: data type, attributes, specialisation, aggregation, and decomposition relations between data types. The notation used in this domain is a subset of UML class diagram notation.
- *Product domain, Application domain, and Technological infrastructure domain*: not covered.

In addition to the domains of Amber, NEML supports the function and role domains and defines a number of supplementary concepts (and corresponding graphical notations) like: function, role, transfer etc.

Amber and NEML are graphical languages. Part of their graphical notations is given in Figure 3-1, and Figure 3-2. However, one can also see an example of behaviour model in Figure 3-5.

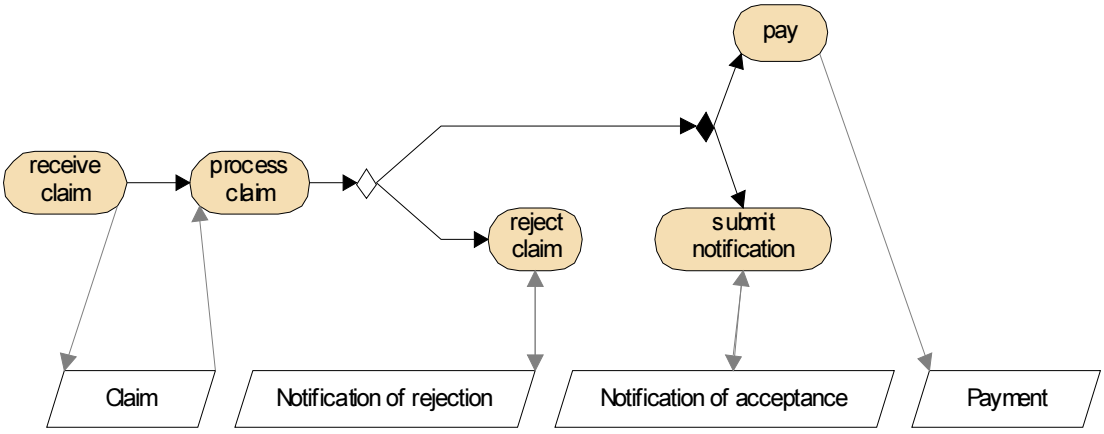



Figure 3-5. Example of process in Amber

Views are used to generate feature overviews of models. For instance, colour views emphasise certain aspects of the model. Other views generated by Testbed Studio visualise precedence relations, dataflow, or the assignment of behaviour to actors. Also structural transformations can illustrate different aspects of a model structure: e.g., an organigram shows the hierarchical structure of an organisation. A powerful concept is that of process lanes. A business process model can be automatically structured with respect to any

attribute. For example, the actions can be structured into a block (sub-process) for every actor involved, showing the change in responsibility in a process, e.g. to reveal the handovers between different organisations. Alternative process lane structures are for example based on the business function associated with an activity, or whether the activity belongs to the primary or secondary part of the process.

The three domains in Amber can also be seen as specific types of viewpoints. It is important to note that a complete model always contains representations of all these domains. Moreover these representations are not isolated from each other and they communicate via several mechanisms.

***Structuring mechanisms:*** In Amber structuring mechanisms are defined to tackle the complexity of behaviour models. Examples of such mechanisms are:

- Grouping (Amber allows actions to be grouped in behaviour blocks) using blocks and entry and exit points (used when a block separates behaviour between actions)
- Nesting (Actions, and actors can be nested)
- Decomposition
- Replications (for actors, actions, interactions and blocks)

Via these structuring mechanisms Amber favours the development of models at successive levels of detail. This feature makes the development and the understanding of large models easier, and induces the scalable character of the language. The same structuring mechanisms are present in NEML.

***Formal underpinnings:*** Amber and NEML both have formal descriptions of their meta-models (see Ferreira Pires 1998, Steen et al. 2002). The notation used in both cases is UML. The purpose of these meta-models is to provide an abstract representation for the language syntax. Furthermore, each concept is separately defined, using the same UML formalism. Apart from this, process models are endowed with a number of operational semantics, having different purposes such as stepwise simulation, model checking, and quantitative analysis.

***Support:*** Amber and NEML are the modelling languages embedded in two modelling tools: Testbed Studio and RSD Studio respectively. They are both accompanied by a methodology for modelling and analysis (see BiZZdesign (2000), Steen et al. (2002), and http://rsd.demo.telin.nl/analysis/).

***Discussion***: The focus of Amber is primarily on business process modelling: it misses the architectural perspective of information systems and the concepts related to this. The language can be extended to a certain extent through user-defined profiles. Amber was the starting point for the development of NEML, which inherited most of its features and added new elements to it. The way the language was defined (via a separate metamodel definition language) makes it relatively easy to modify the concepts and their representation (like in NEML).

## 3.2 IDEF

***Background, scope and usage:*** IDEF is the name of 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 have been developed by the US Air Force Program for Integrated Computer Aided Manufacturing (ICAM). Of these methods, IDEF0, IDEF3, and IDEF1X are the most commonly used, especially by US government agencies and subcontractors. IDEF0 and IDEF1x are published as standards of the National Institute of Standards and Technology. The number of participants in the meetings of the IDEF user group are evidence of the widespread usage of IDEF. Currently, there are 16 IDEF methods, running from IDEF0 to IDEF14:

| IDEF METHODS | | | |
|---|---|---|---|
| IDEF0 | Function Modelling | IDEF7 | Information System Auditing |
| IDEF1 | Information Modelling | IDEF8 | User Interface Modelling |
| IDEF1X | Data Modeling | IDEF9 | Scenario-Driven IS Design |
| IDEF2 | Simulation Model Design | IDEF10 | Implementation Architecture Modelling |
| IDEF3 | Process Description Capture | IDEF11 | Information Artifact Modelling |
| IDEF4 | Object-Oriented Design | IDEF12 | Organization Modelling |
| IDEF5 | Ontology Description Capture | IDEF13 | Three Schema Mapping Design |
| IDEF6 | Design Rationale Capture | IDEF14 | Network Design |

Of these methods, IDEF0, IDEF3, and IDEF1X ("the core") are the most commonly used. In the next paragraphs we will refer in particular to IDEF0 and IDEF3, as we consider them to be most relevant for ArchiMate. 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 is 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 diagram, multiple IDEF1X logical subject area diagrams, and multiple physical diagrams.

Overall, the IDEF family of languages are general-purpose modelling languages and intended to be used by business system designers. For the rest of IDEF languages we one can find detailed information at http://www.idef.com/default.html.

***Concepts and Conceptual Domains:*** In principle, IDEF covers most of the conceptual domains. However, only some of the methods are widely used: the 'core' of IDEF only truly covers the behaviour domain and the information/data domain. One can see below the distribution of the IDEF methods over the conceptual domains:

- *Organisation domain*: by IDEF12.
- *Behaviour domain*: IDEF0, IDEF3, IDEF2.
- *Information/data domain*: IDEF1, IDEF1x, IDEF11.
- *Products domain*: products, business services.
- *Application domain*: IDEF4-IDEF 10
- *Technological infrastructure domain*: partly covered by IDEF14.

*IDEF0: Main concepts.* There are five elements to the IDEF0 functional model (see Figure 3-6): the *activity* (or process) is represented by boxes, *inputs*, *outputs, constraints* or controls on the activities, and *mechanisms* that carries out the activity. The inputs, control, output and mechanism arrows are also referred as ICOMs.

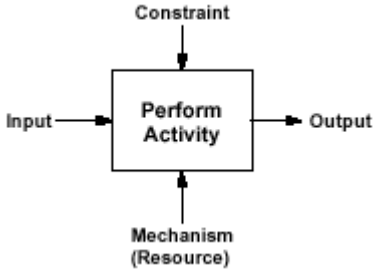



Figure 3-6. IDEF0 representation

Figure 3-7 shows an example of IDEF0 model.

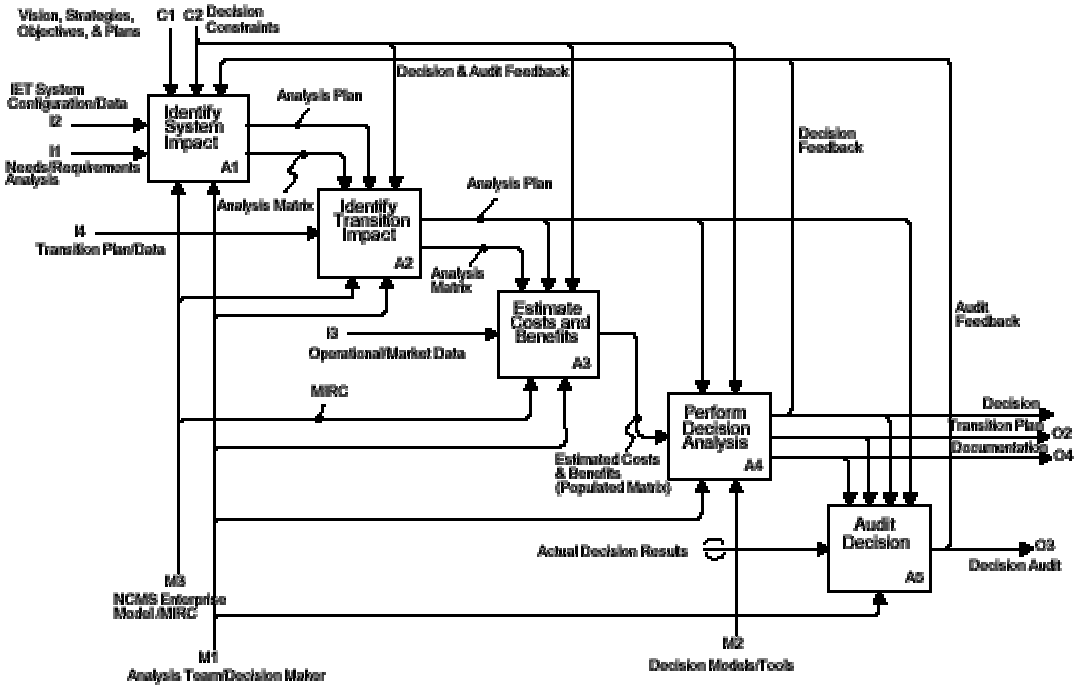



Figure 3-7. Example of IDEF0 model

*IDEF3: Main Concepts.* 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 description captures knowledge of "how things work" in an organization, 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 (see Figure 3-9).

Object state transition network (OSTN) diagrams capture object-centred views of processes, which cut across the process diagrams and summarise the allowable transitions. *Object states* and *state transition arcs* are the key elements of an OSTN diagram. In OSTN diagrams, object states are represented by circles, and state transition arcs are represented by the lines connecting the circles. Other main concepts are: strong transitions, conditions, transition junctions, and elaborations.

The notation used in IDEF0 and IDEF3 models is graphical. It appears that a disadvantage of IDEF is the visual appearance of IDEF diagrams (especially the IDEF0 diagrams). Presley and Liles (1995) mention that they have encountered expressions of aversion from some reviewers and end users when first presented with an IDEF0 diagram: “The network of boxes and arrows, along with the size of some models, can cause many users to reject the model. In our experience, most will overcome this initial reaction if the modelling syntax is explained to them.” Moreover, they state that beginner modellers might need preliminary training. In the context of ArchiMate, it is important to note that the IDEF family provides support for the modelling of several architectural views. However, there are no communication mechanisms between models. The fact they are isolated hinder the visualisation of all models as interrelated elements of an architectural system. This also means that a switch between views is not possible.

***Structuring mechanisms:*** Another characteristic of the IDEF 0 modelling technique is that 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 IDEF 3 (see Figure 3-8 and Figure 3-9 below).

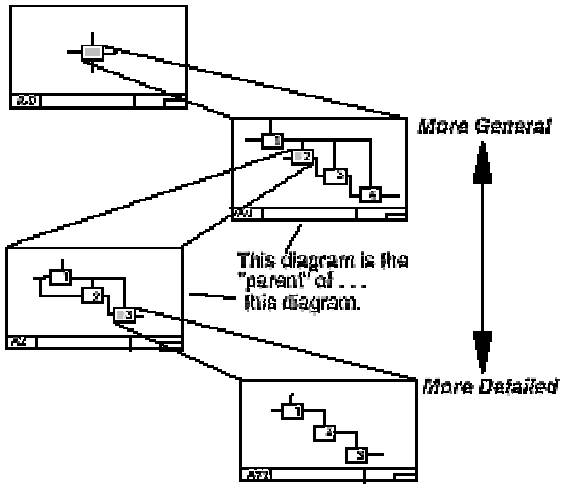



Figure 3-8. IDEF0 decomposition

***Formal underpinnings:*** No formal description of a metamodel or of formal semantics is provided for IDEF0 and IDEF3.

***Support:*** Method support: Apart from being a collection of languages, IDEF also describes a method that goes with a particular language. Tool support: IDEF0, IDEF1x and IDEF3 notations are supported by Popkin’s “System Architect” tool. Other IDEF Tool Suppliers: IDEFine Ltd, Computer Associates, Knowledge Based Systems, Wizdom Systems Inc., Meta Software, Advantage Software Limited, Logic Works.

***Discussion***: IDEF is widely used in the industry. This indicates that it satisfies within acceptable limits the needs of the users. However, we want to refer to a rather critical opinion on IDEF1x (see http://www.aisintl.com/case/idef.html), that raises questions regarding the suitability of IDEF for the modelling of large systems. The IDEF family is subject of a continuous process of development and improvement. Still IDEF0, IDEF1x and IDEF3 are rather stable and rigid languages (http://www.aisintl.com/case/idef.html).

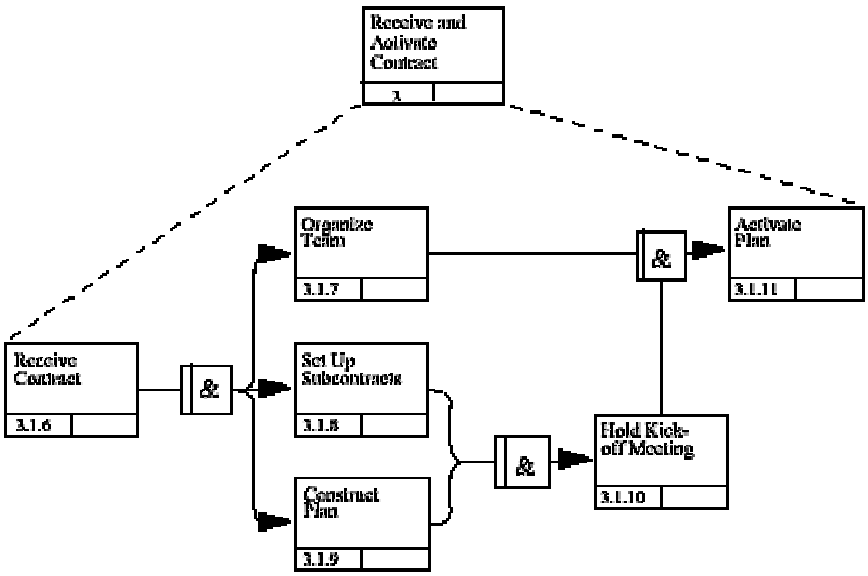



Figure 3-9. IDEF3 decomposition

## 3.3 ARIS

***Background, scope, and usage:*** ARIS ("Architecture of Integrated Information Systems", Scheer 1994) is a well-known approach to enterprise modelling. Although ARIS started as the academic research of Prof. A.W. Scheer, it has now an explicit industrial background. It is not a standard, but it is very well sold and therefore widespread. In IDS Scheer AG has sold over 30000 ARIS licences all over the world. In addition to the high level architectural framework, ARIS is a business modelling method, which is supported by a software tool ("ARIS Toolset"). ARIS is intended to serve various purposes: documentation of existing business process types, blueprint for analysing and designing business processes and support for the design of information systems. The tool is intended for system designers.

***Concepts and Conceptual Domains:*** To model business processes within an enterprise model, ARIS provides a modelling language known as event-driven process chains (EPCs). An EPC is an ordered graph of events and functions. It provides various connectors that allow alternative and parallel execution of processes. Figure 3-10 gives an example of business process model made in ARIS and also presents the graphical notation used in these models. The main concepts defined in ARIS are: events, functions, control flows, logical operators, organisational units, interactions, output flows, environmental data, outputs, human output, message, goal, machine, computer hardware, application software.

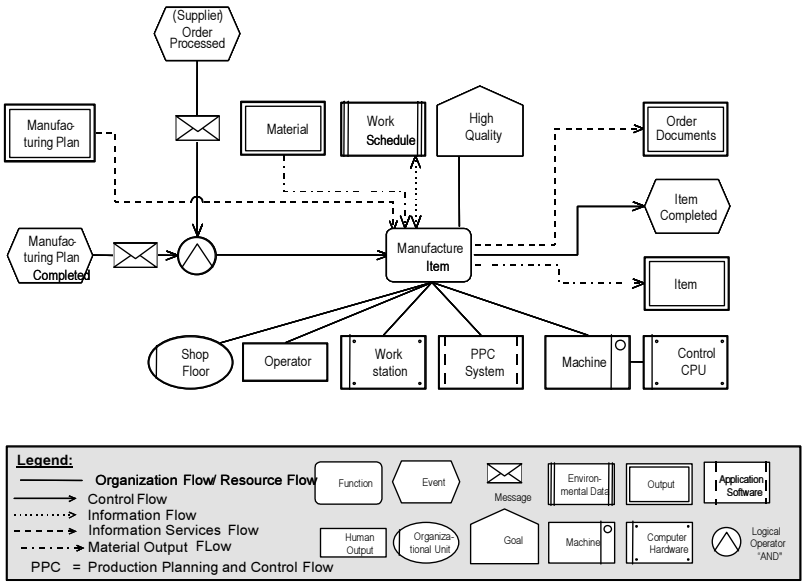



Figure 3-10. Events, functions and control flows in ARIS

The ARIS Toolset includes various editors that can be used to design and edit several types of diagrams. The most important are value added chain diagrams, organisational charts, interaction diagrams, function trees, and Event-driven Process Chains (EPCs). One can see an example of an EPC in Figure 3-12. The temporal order of functions and events is from top to bottom, starting with the events that trigger the process.

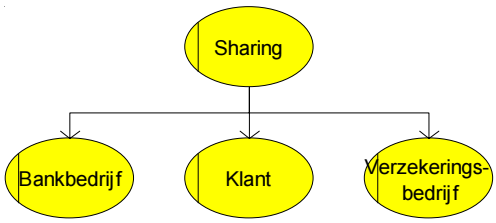



Figure 3-11. ARIS organisational chart

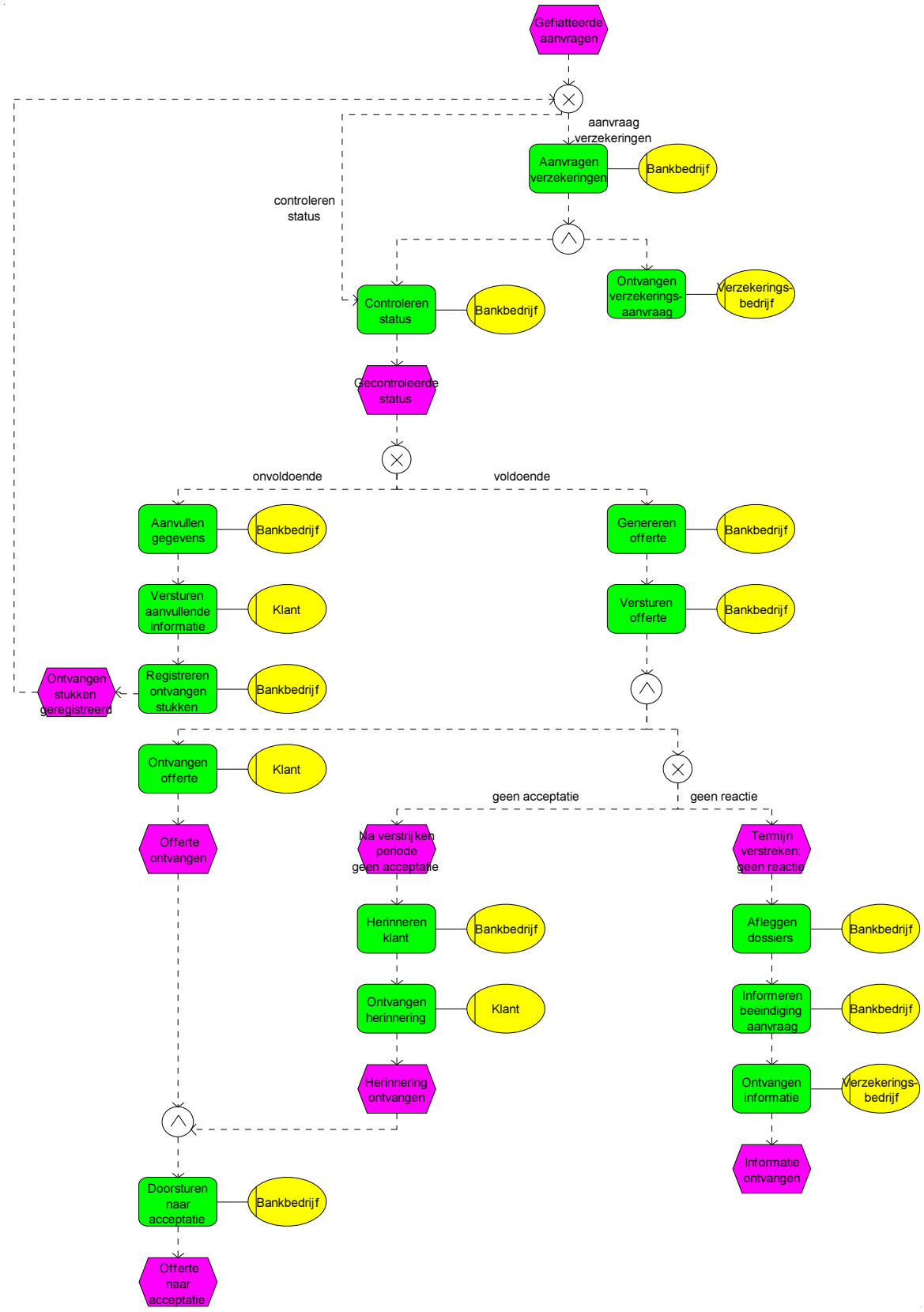



Figure 3-12.EPC model in ARIS

Figure 3-11 illustrates the visualisation of a small organisational chart within ARIS. In summary, the concepts defined in ARIS cover more or less all the conceptual domains. However, only the organisation, behaviour and information domain can be fully modelled using ARIS. For the others the coverage is only superficial.

- *Organisation domain*: organisational charts, organisational units.
- *Behaviour domain*: processes, functions, actions, events, goals, interactions, flows.
- *Information/data domain*: message, environmental data, output
- *Products domain*: human output, output.
- *Application domain*: application software.
- *Technological infrastructure domain*: machine, computer hardware.

The graphical notation of ARIS is unambiguous and easy to understand and use. While ARIS allows for various perspectives on the enterprise (the data view, the control view, the process/function view and the organisation view), the integration of these aspects remains on a low level. Therefore, the tool does not guarantee the overall integrity of interrelated models. The tailorability of ARIS is limited to business modelling, and more precisely to organisational, functional and process modelling. It is very well suited for large models. ARIS is not extensible.

***Structuring mechanisms:*** On a higher level of abstraction, ARIS allows to model decomposition of processes. Objects in ARIS have attributes, relations, and participate in hierarchies.

***Formal underpinnings:*** While there is a formal definition of the syntax of EPCs, EPCs lack a precise definition of their semantics. The semantics of EPCs is given only roughly (in a verbal form) in the original publication by Scheer (Scheer 1992). A comprehensive discussion of the semantic shortcomings of EPCs can be found in Rittgen (2000). This is also the case for corresponding object models which are specified in a rudimentary metamodel. For this reason, ARIS lacks a solid formal foundation and is of limited use for the design of (application) architectures.

***Support:*** ARIS is supported by an architecture framework (an architecture metamodel: “the ARIS house” and a methodology: the “ARIS phase model”) and a software package: the ARIS Toolset.

### 3.4 Other languages

***Role Activity Diagrams (RADs)*** originated through the study of coordination (see Holt, Ramsey, and Grimes 1983). Although RADS were based originally on Petri-Nets, they are a variation on the traditional state-transition chart. Designed originally for Business Process Modelling, they fit the task of modelling components, which realise business rules. RAD’s focus is on processes, which involve the co-ordination of inter-related activities carried out by people in organizations using a variety of tools. The notation supports four foundation classes: Roles - which represent the individual roles in a process, Actions - individual activities or actions carried out by a role, Entities - data, and structures plus collections of entities and tables of entities, and Interactions - which allow roles to communicate by object passing. Concurrent behaviour is modelled by giving a finite-state model for each Role and by allowing Roles to synchronise by letting them share transitions (Ould 1995, Murdoch and McDermid 2000).

***MEMO*** is a tool supported and object oriented methodology for analysing and (re-) designing of business information systems (http://www.uni-

koblenz.de/%7Eiwi/EM/MEMO/index.html). It is based on a set of modelling languages and on the various stages of a macro process that are supplemented with heuristics and techniques (most of them originate in strategic enterprise planning and organisational analysis/design). More precisely, MEMO proposes three object-oriented modelling languages: MEMO-OML, MEMO-OrgML and MEMO-SML. The graphical notation, and the main concepts are common to all languages. The main concepts defined in:

- MEMO-OML are class, object, attribute, interaction, services, constraints, guards, triggers, multiplicity.
- MEMO-OrgML are ProcessType, ProcessUse, ContextOfProcessUse, InputSpec, OutputSpec and Event.
- MEMO-SML (Strategy Modelling Language) are abstract strategy, abstract and total value chain, activity, business unit etc.

The notation used is graphical, very much resembling UML notation. In MEMO there is a clear separation between the modelling languages and the visual appearance of the models. The latter is taken care of by the so-called MEMO Center. The MEMO Center, basically is a user interface, having among other things, the role of providing navigation, simulation and retrieval mechanisms and of user-friendly diagramming tools. It is thus obvious that the diagrams (and the various accompanying textual editors) the user can design while using MEMO Center, are “light” replacements for the graphical notation used by the MEMO languages. Their sole purpose is to ease the creation and understanding of the models and to make their appearance more pleasant.

The MEMO modelling languages are highly integrated and support multiple views. The integration feature (carried out by the MEMO Center) permits the communication between models created in the different languages of MEMO, and ensures the overall integrity of the enterprise architecture.

MEMO has a very solid formal support. A meta metamodel defining the meta language used to specify each of the MEMO modelling languages is provided. Also the several MEMO languages are hierarchically organised according to a “MEMO Meta-Metamodel”. Further on, the integration of the MEMO languages is achieved via the sharing of common concepts and each of the MEMO languages is provided with its own metamodel and with formal semantic of concepts. MEMO is not commercially available and its use is limited primarily to scientific research purposes (although it has been applied in real-life cases).

***Paradigm*** (Groenewegen and De Vink, 2002) is a coordination modelling language, expressing coordination through behaviour (expressed in state transition diagrams) and behaviour influencing. Coordination as specified through Paradigm might be clarifying for the problems of behavioural consistency between components within various kinds of architecture. In addition, Paradigm’s notions can be used to model the coordination of (business) process migration, providing a smooth transition from the current situation to the desired situation. As these problems belong to the kernel problems to be addressed in ArchiMate, the ideas of Paradigm can provide useful input to the ArchiMate language.

Paradigm has been successfully integrated with OMT, one of the predecessors of UML. Research has shown that Paradigm can also be embedded in UML using statecharts extended with some new notions.

## 3.5 Conclusions

We have selected for this survey representative current languages in the area of organisation and process modelling. It is clear that none of them has succeeded to become "the language". Overall, 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 use a non-standard notation (except IDEF which is a standard);
- most languages have a weak formal support (except MEMO and Amber);
- most languages miss the overall architectural vision: therefore, from ArchiMate's point of view they are limited in scope.

However, ArchiMate can benefit from each of these languages. We highlight some of their qualities:

- Amber is used in two ArchiMate client organisations as the main business process modelling language. It is relatively easy to understand and use. Apart from this, ArchiMate can use the in-house expertise related to this language.
- ARIS uses a very simple and attractive notation, which made it very successful.
- IDEF and RAD are standardised notations.
- MEMO has a solid formal foundation.

Table 3-1 summarises the main features of all the surveyed languages.

Table 3-1. Organisation and process modelling languages overview.

| Name & Background | Scope | Main Concepts | Structuring Mechanisms | Model representation & flexibility | Formal underpinning | Support | Comments | Relevance for Archimate |
|---|---|---|---|---|---|---|---|---|
| AMBER & Neml. Research background. Amber is supported by a commercial tool and used in The Netherlands. | Business process and organisation modelling | actor, behaviour, item, role, action, flow, relation etc. | grouping, nesting, replication, decomposition | clear and intuitive graphical notation, multiple view languages, extensible via user profiles | both languages have formal meta-models described in the UML notation | Testbed Studio, RSD Studio, Testbed and RSD methods | simplicity and clarity of models, one-to-one mapping of models to domains, used by the clients, limited to business process modeling, and to The Netherlands, extandable, in house expertise | ●●●●○ |
| RAD Originators: Ould & Roberts 1986, commercialy promoted by Praxis & Coordination Systems | Business process modelling | state, action, interaction, role, external event, parallel action etc. | grouping | sigle view, simple graphical notation, labelling of states is an option, hardly extensible and tailorable | RAD is based on the formal language SPML | STRIM, PRAXIS, Grade Modeller | accessible, simple, simplistic,not extendable, widely known, single view, no solid formal underpinnings, not extensively used | ●●○○○ |
| IDEF family. Military background, standards, widely used in industry | Enterprise modelling and analysis | input, output, constraints, mechanisms, UOB, logical flows, object states, state transition arc | decomposition | graphical notation, the visual apearance of model is not attractive, big models are hard to follow, family of single view languages, IDEF1x not suitable for large systems, less tailorable and extesible | no formal underpinings | supported by System Architect and other tools | comprehensive, good structuring mechanism, standard notation, extensive use in the industry, supported by several tools, no integration between languages, models look crowded and ard to follow, not extensible and not suited for large systems. | ●●●○○ |
| ARIS Originator Prof. A.W. Scheer, Industrial background | Enterprise modelling and analysis | events, functions, control flows, logical operators, organisational units, interactions, output flows, environmental data, outputs, human output, message, goal, machine, computer hardware, application software | model decomposition, hierarchies of processes | attractive graphical notation, easy to follow, multiple view language, low level of integration, tailorable, not extensible | formal definition of the syntax, no formalism for semantics | ARIS Toolset, ARIS method | simplicity and attractivenes of models and notation, accessible, very well suited for modelling, commercially successful, multiple view, limited for architectural purposes, no one-to-one mapping of models to domains, low integration, weak formal underpinnings, rudimentary structuring mechanisms | ●●●○○ |
| MEMO, academic background, not commercially available | Modelling and analysis of enterprise information systems | class, object, attribute, interaction, services, constraints, guards, triggers, multiplicity | associations , specialisation, generalisation (single and multiple inheritance), subtyping, decomposition diagram, process generalisation diagram | graphical notation resembling UML, separation of modelling languages from visual appearance of models, user-friendly diagrammig, extensible, multiple view | solid formal support | MEMO framework and methodology, MEMO Center (modelling environment) | multiple-view, integrated modelling languages, solid formal foundation, separation of visualisation and modelling concerns, good structuring mechanisms, coverage of most of the domains, complex, very abstract graphical notation, heavily OO, usage limited to academic research purposes | ●●●○○ |

# 4 Application and technology modelling languages

## 4.1 Unified Modelling Language

The Unified Modelling Language (UML) is an important industry-standard language for specifying, visualising, constructing, and documenting the artefacts of software systems, managed by the Object Management Group (OMG). It emerged from the combination of three existing languages for object-oriented modelling (hence “unified”) with an industrial origin. In this section, we include the architectural concepts proposed for the next version of the standard, UML 2.0.

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. However, the current version of the language, v1.4, lacks native support for many architectural concepts.

In response to UML Infrastructure and Superstructure Requests for Proposals (RFP), several proposals have been submitted. Among these, we judge the proposals by U2 Partners (http://www.u2-partners.org), a consortium of major vendors and users of UML, as belonging to the most likely candidates.

***Concepts*:** UML is a disturbingly rich combination of eight different visual languages each having its own (sub)scope of the complete UML scope. Moreover, apart from the component diagrams and the deployment diagrams, each of the other six 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 statechart diagrams. The advantage of such richness is the expressiveness of the language; a serious disadvantage is the readability and the 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. Especially the extensive tool support is very supportive.

The nine types of diagrams in UML 1.4 can be grouped according to three aspects:

- Structure: class diagrams, object diagrams;
- Behaviour: use case diagrams, Statechart diagrams, sequence diagrams, collaboration diagrams, activity 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 metamodel are

interrelated; no strict separation between views and metamodel 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; an good overview is given in Fowler and Scott (1999).

*Domains*: 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 Infrastructure proposal (see U2 Partners 2002a) defines the foundational language constructs for UML 2.0. We will not go into this foundation, but rather concentrate on the higher-level concepts defined in UML Superstructure (see U2 Partners 2002b). Given UML's orientation towards software development, these concepts seem especially useful in modelling application architectures and to some extent the technical infrastructure.

*Structural concepts*: A *Class diagram* in UML is used to model the static structure of a system. A class is a description of a set of objects that share the same attributes, operations, relationships and semantics, and implements one or more interfaces. The relations between the classes are dependencies, generalisations and associations, which represent structural relationships among objects.

Two other static diagrams exist in UML, both for modelling physical aspects of object-oriented systems. A *component diagram* models the static implementation view of a system. This involves modelling the physical things that reside on a component, such as executables, libraries, tables, files, and documents. A *deployment diagram* models the static deployment view of a system, and models the configuration of run time processing nodes and the components that live on them. This involves modelling the topology of the hardware, i.e., it models the technological infrastructure domain. Both diagrams are essentially *class diagrams*, but a *component diagram* will focus on system's components and *deployment diagram* on system's nodes.

*Behavioural concepts*: UML provides four different views on the behaviour of a system or organisation. (1) A *functionality* view by means of use cases, which essentially model activities without explicitly showing the flow between them. (2) A *scenario*-oriented view, using interactions to model communication. (3) *Local behaviour* view, modelling the full behaviour of specific elements in the model with state machines. These first three views come together in the fourth, in which (4) *activity diagrams* provide an overview of the flow in the system from activity to activity. A weak point in UML is that it does not address the consistency between the different behavioural concepts. To show relations between the

diagrams, UML depends on the reader's intuition – based on things like similar labels for corresponding actions, events, transitions, etc. – rather than semantics.

In modelling cooperation between system elements, two aspects need to be covered: the structural description of the participants and their communication, i.e., the behaviour they exhibit. In UML, the structure of the participants, their roles and their relationships are modelled by a *collaboration*, and the communication pattern is described by an *interaction*. It comprises a set of *messages* exchanged among a set of objects within a context to accomplish a purpose. An interaction can be visualized in two ways: emphasising the time ordering of messages in a *sequence diagram*, or emphasising the structural organisation of objects in a *collaboration diagram*. In UML 2.0 proposal, interactions can also be modelled in *timing diagrams* and *activity diagrams with interactions*. This wide variety of notation for the same underlying concept allows the designer considerable flexibility. However, the resulting models (especially the activity diagrams) might confuse the uninitiated. Although both interactions and collaborations are used to model aspects of the same cooperation, it appears to be no formal link between the two in UML 2 metamodel.

***Structuring mechanisms*:** UML provides concepts for modelling decomposition of object-oriented models via aggregation and composition relations between classes (and objects). UML has different means of refining or globalizing its model parts:

- the static things in UML can be refined as groups (aggregations) of smaller static things, or they can be refined as special representatives (specializations);
- the dynamic things in UML can also be refined (states within statecharts can become superstates comprising one or more smaller statecharts).
- although consistency of the different descriptions still is a problem, these refinements and globalizations allow for zooming in as well as zooming out with respect to a model: on a detailed scale as well as on a global scale the same kind of descriptions occur.

In addition, UML allows for differentiating between descriptions from outside – black-box-like – and from inside – white-box-like; interfaces usually make up for outside descriptions; in this way technical details can be hided without getting lost.

Furthermore, UML has packages and subsystems, allowing for selecting any part of a model while everything else is being left out; this provides for much freedom in choosing a view onto a model: a view can comprise everything relevant for a certain user or a certain usage of the model, or a view can comprise everything relevant for a certain aspect as e.g. statics or security or coordination.

In particular the combination of the possible structurings is very useful in the light of scalability, reuse, accessibility for different stakeholders.

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

- 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 metamodel.

- *Tagged value* is an extension of the properties of a UML element that allows the creation of new information in that element's specification. Tagged values can be added to all existing metamodel 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.

A profile which is particularly relevant for ArchiMate is the profile for Enterprise Distributed Object Computing (EDOC). Its goal is to provide architecture and modelling support for collaborative or Internet computing, with technologies such as web services, Enterprise Java Beans, and Corba components. The EDOC profile was adopted by the OMG as a standard in November 2001 and will provide model-driven development of enterprise systems based on the Model-Driven Architecture (MDA, Section 2.4). The EDOC profile provides a business collaboration architecture, a technology-independent business component architecture and modelling concepts for describing business processes, applications, and infrastructure.

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 that are not familiar with them. In such cases stereotypes take away one of the strong points of UML, which is standardisation.

Furthermore, the (formal) consistency between various parts of a UML model being not defined, the consistency of UML models is still an issue of study and discussion. On an informal level only, the problem whether some model part fits into the rest of the model, is now easier to handle.

***Formal underpinnings*:** UML has, only partially, a formal basis. What it is missing is mainly semantics and consistency in particular. As this is a topic of ongoing and intensive research, one might expect substantial improvements concerning semantics and consistency within the not very far future. So analysis and verification of UML models are at the moment dependent on what tools provide. In particular, some behavioural analysis via animation is what current tools offer. In other cases, explicit transformations to other formalisms are being done in order to facilitate verification of properties one wants to establish.

Semantics for individual diagram types exist, in a more or less formal manner. However, a formalised integrated semantics is still lacking. Work is progressing on this, e.g. by the precise UML (pUML) group and the Action Semantics task force of the OMG.

The lack of an integrated semantics makes it difficult to provide analytical support for UML. Analysis is limited to what is permitted within a single diagram, and since the semantics of UML has not been specified very well in the standard (at least up to v.1.4), rigorous analysis techniques are difficult to define.

***Support*:** For UML many commercial as well as public domain modelling environments exist. As UML is so large, most of these do not (yet) cover everything. But they are certainly improving. 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.

UML 2.0 standardisation process has not yet been concluded. Consequently, tools do not support this version, and the new architectural concepts most interesting to ArchiMate have not yet been implemented.

***Summary/discussion*:** UML is the mainstream modelling approach within ICT, and its use is expanding into other areas. 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 accessible and understandable for managers and organisational specialists; therefore, special visualizations and views of UML models should be provided. These could be based on existing profiles, on profile ideas currently under study, or for example on Testbed or Testbed-like representations of business processes.

## 4.2 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. Typically, they describe an architecture in terms of components and connectors. A large number of ADLs have been proposed, some for a specific application area, some more generally applicable. In Medvidovic and Taylor (2000) the basics of ADLs are described, and a large number of ADLs are compared. Table 4-1 summarises the focus of a number of ADLs (derived from Medvidovic and Taylor 2000).

Table 4-1. Focus of several ADLs

| **ADL** | **Focus** |
|---|---|
| ACME | Architectural interchange, mainly at the structural level |
| Aesop | Specification of architectures in specific styles |
| C2 | Architectures of highly-distributed, evolvable and dynamic systems |
| Darwin | Architectures of highly-distributed systems whose dynamism is guided by strict formal underpinnings |
| Rapide | Modelling and simulation of the dynamic behaviour described by an architecture |
| SADL | Formal refinement of architectures across levels of detail |
| UniCon | Glue code generation for interconnecting existing components using common interaction protocols |
| Weaves | Data-flow architectures characterised by high-volume of data and real-time requirements on its processing |
| Wright | Modelling and analysis of the dynamic behaviour of concurrent systems |

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

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.

ADLs generally have an academic background, and limited usage. However, the ADL ACME is widely accepted as a standard to exchange architectural information, also between other ADLs. There are initiatives to integrate ACME in UML, both by defining translations between the languages and by a collaboration with OMG to include ACME concepts in UML 2.0. In this way, the concepts will be made available to a large user base and be supported by a wide range of software tools. In the remainder of this section, we will especially consider ACME, which can be regarded as a representative of ADLs.

***Concepts and representation*:** 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 is claimed to be suitable as a general architecture interchange language. Therefore, its concepts can be considered 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)

Many of these concepts are now also proposed for the forthcoming UML 2.0 standard. In ACME, other aspects of an architectural description are represented with property lists.

In ACME, other aspects of an architectural description are represented with property lists. Like most ADLs, ACME provides both a textual and a graphical representation of the concepts, as well as automated support to alternate between them. It also allows for both top-level and detailed views of composite elements. Support for other views is limited.

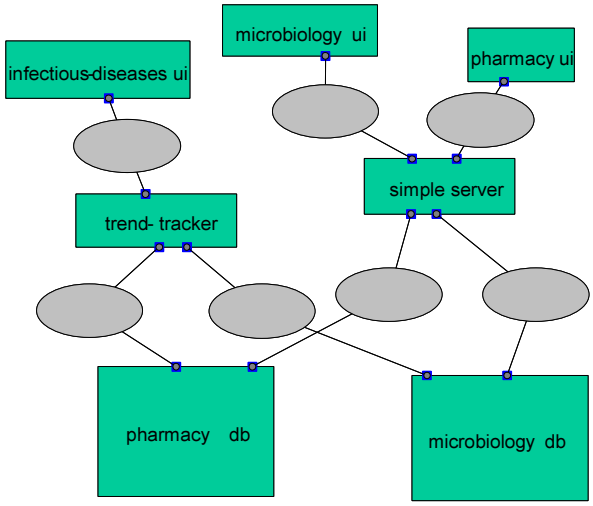



Figure 4-1. Example of an ACME specification

The *Architecture Description Markup Language* (ADML) has originally been developed as an XML encoding of ACME. It is being promoted by the OpenGroup as a standard for enterprise architectures.

***Structuring mechanisms*:** ACME provides features to support hierarchical composition of components. Architectural refinement is still an open research area, and support for it in ACME ass well as other existing ADLs is limited.

***Flexibility*:** The basic ACME concepts are very general, and can be made more specific by adding property lists. The tool AcmeStudio provides a number of different architectural styles, and allows for the use of specific symbols for specific types of components.

***Formal underpinnings*:** ACME focuses on describing the architectural structure of systems; it does not provide specific computational semantics for architectures. However, it uses a so-called *open semantic framework*, providing a basic structural semantics while allowing the inclusion of the semantics of specific other ADLs.

***Support*:** Tool support for existing ADLs varies widely. ACME is supported by AcmeStudio (http://www-2.cs.cmu.edu/~acme/AcmeStudio/), a research tool which is available free of charge. In addition to creation and editing of graphical ACME specifications, the tool supports a number of visualisations (e.g. based on the properties of concepts) and simple performance analysis based on queueing models.

***Summary/discussion*:** A wide variety of Architecture Description Languages (ADLs) exists, 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. ACME is of particular interest, as it can serve as a standard for the interchange of architectural descriptions, and its concepts have been proposed to be incorporated in UML. Therefore, it is important to make sure that the ArchiMate concepts are consistent with those of ACME, so that a mapping between them is possible.

### 4.3 Other application modelling languages

There are several other, usually older, techniques to model applications. Many of them are used for detailed description of programs or algorithms, and are less interesting from an architectural viewpoint. Many languages are either incorporated in UML or superseded by UML. An example of a language which is still in use in many organisations are data flow diagrams, which describe an information system or application in terms of data stores, processes and data flows between them.

Some techniques from the IDEF family (see Section 0) also fit into the application modelling level, in particular IDEF4 and IDEF10. However, as they are not part of the ‘core’ of IDEF, their use is limited.

## 4.4 Technical infrastructure modelling

As stated in Boar (1999), the technical infrastructure of an organisation is usually represented in informal pictures. At the very detailed hardware level, more or less formal description techniques such as VHDL exist. However, at the level of complete systems and networks, language support is minimal. In Boar (1998), Boar introduces his own technique, Enterprise IT Architecture Blueprinting (EAB), to fill this gap.

As described in Section 4.1, UML uses the deployment diagram to model nodes in a technical infrastructure and their connections. IDEF14 (see Section 3.2) can be used to model a network design, but as it is not part of the “core” of IDEF, its use is not widespread. Moreover, it is very difficult to find information about the exact contents of the IDEF14 standard.

## 4.5 Conclusions

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, and its use is expanding into other areas. 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 accessible and understandable for managers and organisational specialists; therefore, special visualisations and views of UML models should be provided. Given the importance of UML, other modelling languages will likely provide an interface or mapping to it. This is also advisable to ArchiMate; it should be possible to describe the ArchiMate concepts in UML, or to map them to UML by using e.g. stereotypes and profiles.

The concepts used in ADLs are generally well-defined and – although originally intended for the description of software architectures – broadly applicable. However, because of the generic concepts which may be difficult to understand for non-expert users, ADLs are more suitable as an underlying foundation for the ArchiMate language than as a language that is used for the actual descriptions. ACME, which can serve as a standard for the interchange of architectural descriptions, is of particular interest. However, UML 2.0 will most likely comprise more concepts on the architectural level as well, drawing inspiration from languages like ACME. This may obviate the need for a separate ADL for modelling software systems.

At the technical infrastructure level, well-defined modelling languages are nearly non-existent.

# References


Architecture Board ORMSC (2001), Model Driven Architecture (MDA), J. Miller and J. Mukerji (eds.). Document nr. ormsc/2001-07-01, July 2001.

BiZZdesign (2000), Handboek Testbed, versie 6.1, juni 2000, Enschede (in Dutch).

Boar, B.H., (1998), Constructing Blueprints for Enterprise I.T. Architecture, Wiley, 1998.

Boar, B.H., (1999), A blueprint for solving problems in your IT architecture, IT Professional, Nov./Dec. 1999, pp. 23-29.

C4ISR Architecture Working Group (1997), C4ISR Architecture Framework Version 2.0., US Department of Defense, Dec. 18, 1997. http://www.c3i.osd.mil/org/cio/i3/AWG_Digital_Library/pdfdocs/fw.pdf

Fielt, E., Eijkel, G. van den, Janssen, W., Steen, M., and Oude Luttighuis, P., (2000), Rapid Service Development Methodology, TI/RS/2000/030, Telematica Instituut, May 2000.

Ferreira Pires, L., Jonkers, H., Lankhorst, M., Quartel, D.A.C., Sinderen, M.J. van, Teeuw, W., (1998) AMBER - Architecturale ModelleerBouwdoos voor bEdRijfsprocessen, Testbed project, Testbed/WP3/D3.2/V3.4, Telematica Instituut, Enschede, March 1998.

Fowler, M., Scott, K., (1999), UML Distilled: A Brief Guide to the Standard Object Modeling Language, 2nd edition, Addison-Wesley, 1999.

Holt, A.W., Ramsey, H.R., Grimes, J.D., (1983), Coordination system technology as the basis for a programming environment, Electrical Communication, vol. 57, no. 4, 1983.

Groenewegen, L., and E. de Vink, (2002), Operational Semantics for Coordination in Paradigm, in F. Arbab and C. Talcott (eds.), Coordination 2002, LNCS 2315, pp. 191-206.

IDEF (1993), Integration Definition for Function Modeling (IDEF0) Draft, Federal Information Processing Standards Publication FIPSPUB 183, U.S. Department of Commerce, Springfield, VA 22161, Dec. 1993.

IEEE Computer Society, (2000), IEEE Std 1472-2000: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, Oct. 9, 2000.

IFIP-IFAC (1999), IFIP-IFAC Task force on Architectures for Enterprise Integration, GERAM: Generalised Enterprise Reference Architecture and Methodology, Version 1.6.3, March 1999 (Published also as Annex to ISO WD15704). http://www.fe.up.pt/~jjpf/isf2000/v1_6_3.html

Iacob, M.-E., W. Huijsen, D. van Leeuwen, H. ter Doest, H. Bosma, J. Guillen Scholten, (2002), State of the Art in Architecture Support, M.-E. Iacob (ed.), ArchiMate deliverable D3.1, Telematica Instituut, Enschede, Nov. 2002.

ITU, (1996) ITU Recommendation X.901 | ISO/IEC 10746-1: Open Distributed Processing - Reference Model - Part 1: Overview, 1996.

ITU, (1995a), ITU Recommendation X.902 | ISO/IEC 10746-2, Open Distributed Processing - Reference Model - Part 2: Foundations, 1995.

ITU, (1995b), ITU Recommendation X.903 | ISO/IEC 10746-3, Open Distributed Processing - Reference Model - Part 3: Architecture, 1995.

ITU, (1997) ITU Recommendation X.904 | ISO/IEC 10746-4, Open Distributed Processing - Reference Model - Part 4: Architectural Semantics, 1997.

Janssen, W.M.P. and M.W.A. Steen, (2000), Rapid Service Development: An integral approach to e-business engineering, in S. Murugesan and Y. Deshpande (eds.), Proc. 3rd Workshop on Web Engineering at the 9th International World Wide Web Conference, Amsterdam, the Netherlands, May 2000.

Kruchten, P., (1995), Architectural Blueprints - The “4+1” view model of software architecture, IEEE Software, vol. 12, no. 6, Nov. 1995, pp. 42-50.

Menzel, C. and Mayer, R.J., (1998), The IDEF family of languages, in P. Bernus, K. Mertins and G. Schmidt (eds.), Handbook on Architectures of Information Systems, vol. 1 of International Handbooks on Information Systems, chapter 10, pp. 209-241. Springer Verlag, 1998.

Medvidovic, N. and R.N. Taylor, (2000), A classification and comparison framework for software architecture description languages, IEEE Transactions on Software Engineering, 26 (1), Jan. 2000, pp. 70-93.

Mayer, R.J., Menzel, C.P., Painter, M.K., deWitte, P.S., Blinn, T., Perakath, B., (1995), Information Integration for Concurrent Engineering (IICE) IDEF3 Process Descrition Capture Method Report, Interim Technical Report April 1992-September 1995, Knowledge Based Systems Inc.

Murdoch, J. and J.A. McDermid, (2000), Modeling engineering design processes with Role Activity Diagrams, Transactions of the Society for Design and Process Science, vol. 4, no. 2, June 2000.

Ould, M.A., (1995), Business processes: Modelling and analysis for re-engineering and improvement. J. Wiley, Chichester, 1995.

Presley, A. and Liles, D. (1995). The Use of IDEF0 for the Design and Specification of Methodologies. 4th Industrial Engineering Research Conference, Nashville.

Putman, J.R., (1991), Architecting with RM-ODP, Prentice-Hall, 1991.

Rittgen, P., (2000), A modelling method for developing web-based applications, in Proc. of International Conference IRMA 2000, Anchorage, Alaska, pp. 135-140.

Scheer, A.-W., (1992), Architektur Integrierter Informationssysteme, 2nd edition. Springer-Verlag, 1992.

Scheer, A.-W., (1994), Business Process Engineering: Reference Models for Industrial Enterprises, Springer, Berlin, 2nd ed. 1994.

Steen, M.W.A., M.M. Lankhorst and R.G. van de Wetering, (2002), Modelling Networked Enterprises, in Proc. Sixth International Enterprise Distributed Object Computing Conference (EDOC'02), Lausanne, Switzerland, Sept., pp. 109-119.

Sowa, J.F. and J.A. Zachman, (1992), Extending and formalizing the framework for information systems architecture, IBM Systems Journal, 31 (3), 1992, pp. 590-616.

U2 Partners, (2002a), Unified Modeling Language: Infrastructure, version 2 beta R2 (draft), 3 June 2002. OMG document ad/02-06-01, http://cgi.omg.org/cgi-bin/doc?ad/02-06-01.pdf.

U2 Partners, (2002b), Unified Modeling Language 2.0 Proposal, version 0.69 (draft), 15 March 2002. OMG document ad/02-04-05, http://cgi.omg.org/docs/ad/02-04-05.pdf.

Zachman, J.A., (1987), A framework for information systems architecture, IBM Systems Journal, 26 (3), 1987, pp. 276-292.

Zee, H. van der, Laagland, P. and Hafkenscheid, B. (eds.), (2000), Architectuur als Management Instrument - Beheersing en Besturing van Complexiteit in het Netwerktijdperk, 2000. ISBN 90-440-0087-x.

# Index