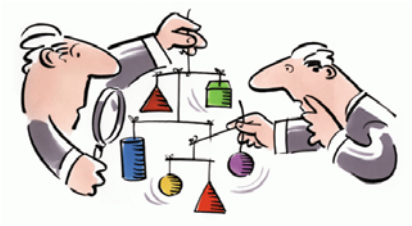
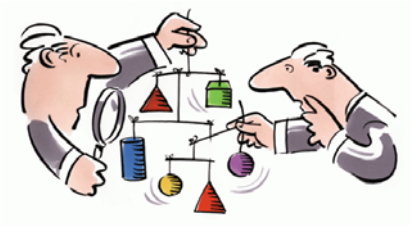


ArchiMate Language Primer

Introduction to the ArchiMate Modelling Language for Enterprise Architecture



Colophon

Title :	ArchiMate Language Primer
Date :	26-08-2004
Version :	1.0
Change :	
Project reference :	ArchiMate/D1.1.6a
TI reference :	TI/RS/2004/024
Company reference :	
URL :	https://doc.telin.nl/dscgi/ds.py/Get/File-43840/
Access permissions :	Project
Status :	Final
Editor :	Marc Lankhorst
Company :	Telematica Instituut
Author(s) :	Marc Lankhorst and the ArchiMate team

Synopsis:

This document describes the essentials of the ArchiMate language for enterprise architecture, and gives an extended example of its use.

ArchiMate

Organisations need to adapt increasingly fast and anticipate 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 fora 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, Centrum voor Wiskunde en Informatica (CWI), the 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

1 Introduction

In many modern ICT-intensive organisations, several types of architects and architectures can be found.The technical ICT-related disciplines already have a somewhat longer architectural tradition, although the distinction between architecture and design is not always sharp. Application architects, for example, describe the relations between the many software applications used within the enterprise, as well as the global internal structure of these applications. Presently, the Unified Modelling Language (UML) is usually the language of choice for this purpose, although there are still organisations using their own proprietary notation. The architecture of the technical infrastructure, describing, among others, the layout of the computer hardware and networks hardware in the company, is generally captured in informal drawings of ‘clouds’ and ‘boxes’, if at all.

In the more business-oriented disciplines, “working under architecture” is a more recent development. Since the advent of process orientation in the nineties, more and more organisations have started to document their business processes in a more or less formal way. However, these descriptions do not focus on the architectural aspects, i.e., they do not provide an overview of the global structure within processes and the relationships between them. Some organisations have a description of their product portfolio, which is generally text-based: visual modelling has not yet gained acceptance in this field.

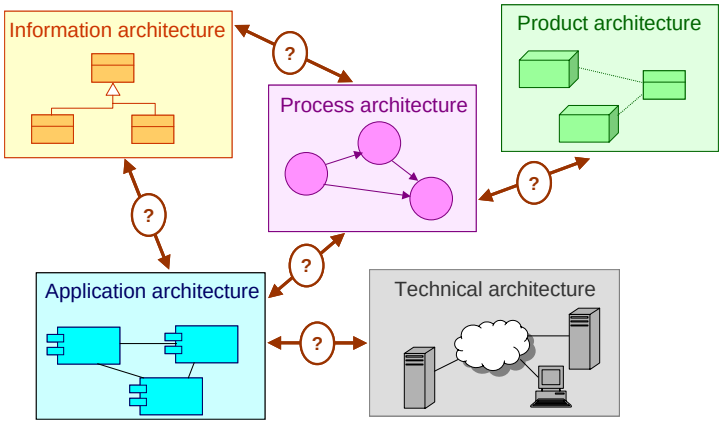


Figure 1. Heterogeneous architectural domains.

Thus, we can say that within many of the different domains of expertise that are present in an enterprise, some sort of architectural practice exists, with varying degrees of maturity. However, due to the heterogeneity of the methods and techniques used to document the architectures, it is very difficult to determine how the different domains are interrelated. Still, it is clear that there are strong dependencies between the domains. For example: the goal of the (primary) business processes of an organisation is to realise their products; software applications support business processes, while the technical infrastructure is needed to run the applications; information is used in the business processes and processed by the applications. For optimal communication between domain architects, needed to align designs in the different domains, a clear picture of the domain interdependencies is indispensable.

With these observations in mind, we conclude that a language for modelling *enterprise architectures* should focus on inter-domain relations. With such a language, we should be able to model:

- The global structure *within* each domain, showing the main elements and their dependencies, in a way that is easy to understand for non-experts of the domain.
- The relations *between* the domains.

Another important property of an enterprise modelling language – as for any modelling language – is a formal foundation, which ensures that models can be interpreted in an unambiguous way and that they are amenable to automated analysis. Also, it should be possible to visualise models in a different way, tailored towards specific stakeholders with specific information requirements.

None of the currently existing modelling languages completely meet these requirements. In this chapter, we propose the enterprise modelling language that we use throughout this book. Although, in principle, the concepts of this language are sufficiently generic and expressive to model many aspects within different domains, it is clearly *not* our intention to introduce a language that can replace all the domain-specific languages that exist. For specific (detailed) designs of, e.g., business processes or applications, the existing languages are likely to be more suitable. In the language that we propose, we conform as much as possible to existing standards.

2 Core Concepts of the Language

In the enterprise modelling language that we propose, the *service* concept plays a central role. A service is defined as a unit of functionality that some entity (e.g., a system, organisation or department) makes available to its environment, and which has some value for certain entities in the environment. Service orientation supports current trends such as the service-based network economy and ICT integration with Web services. These examples already show that services of a very different nature and granularity can be discerned: they can be provided by organisations to their customers, by applications to business processes, or by technological facilities (e.g., communication networks) to applications.

A layered view provides a natural way to look at service-oriented models. The higher layers make use of services that are provided by the lower layers. Although, at an abstract level, the concepts that are used within each layer are similar, we define more concrete concepts that are specific for a certain layer. In this context, we distinguish three main layers:

1. The *Business layer* offers products and services to external customers, which are realised in the organisation by business processes performed by business actors.
2. The *Application layer* supports the business layer with application services which are realised by (software) applications.
3. The *Technology layer* offers infrastructural services (e.g., processing, storage and communication services) needed to run applications, realised by computer and communication hardware and system software.

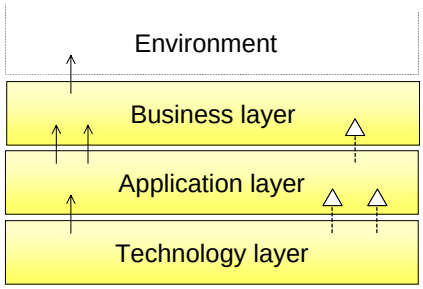


Figure 2. Layers.

Each of these main layers can be further divided in sub-layers. For example, in the Business layer, the primary business processes realising the products of a company may make use of a layer of secondary (supporting) business processes; in the Application layer, the end-user applications may make use of generic services offered by supporting applications. On top of the Business layer, a separate Environment layer may be added, modelling the external customers that make use of the services of the organisation (although these may also be considered part of the Business layer).

In line with service orientation, the most important relation between layers is formed by *use* relations, which show how the higher layers make use of the services of lower layers. However, a second type of link is formed by *realisation* relations: elements in lower layers may realise comparable elements in higher layers; e.g., a ‘data object’ (Application layer)

may realise a ‘business object’ (Business layer); or an ‘artifact’ (Technology layer) may realise either a ‘data object’ or an ‘application component’ (Application layer).

The general structure of models within the different layers is similar. The same types of concepts and relations are used, although their exact nature and granularity differ. Figure 3 shows the central structure that is found in each layer.

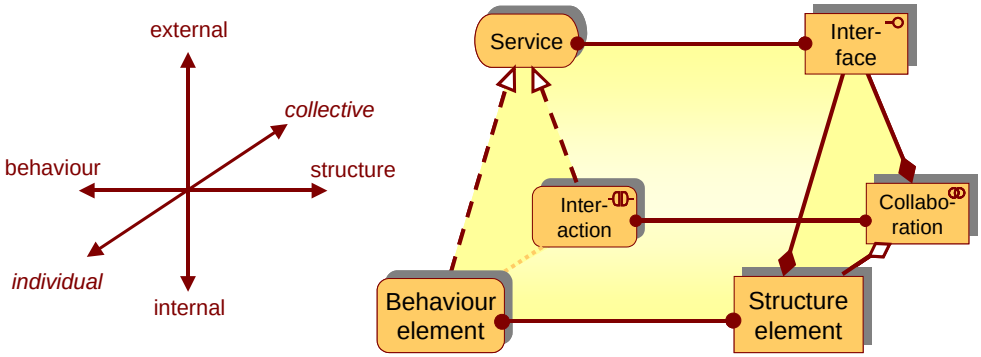


Figure 3. The core concepts in three dimensions.

First, we distinguish the *structural* or *static* aspect (right side of Figure 3) and the *behavioural* or *dynamic* aspect (left side of Figure 3). Behavioural concepts are *assigned* to structural concepts, to show who or what displays the behaviour. In the example, *role*, *interface* and *collaboration* are assigned to *business process*, *organisational service* and *business interaction*, respectively.

Second, we make a distinction between an *external view* and an *internal* view on systems. When looking at the behavioural aspect, these views reflect the principles of service orientation as introduced in the previous section. The *service* concept represents a unit of essential functionality that a system exposes to its environment. For the external users, only this external functionality, together with non-functional aspects such as the quality of service, costs etc., are relevant. If required, these can be specified in a contract or service level agreement. Services are accessible through *interfaces*, which constitute the external view on the structural aspect.

Although for the external users only the external view is relevant, the design of organisations or systems and their internal operations and management also requires knowledge about the *internal realisation* of the services and interfaces. For this realisation, we make a distinction between behaviour that is performed by an *individual* structural element (e.g., actor, role component, etc.), or collective behaviour (interaction) that is performed by a *collaboration* of multiple structural elements.

In addition to *active* structural elements (the business actors, application components and devices that display actual behaviour, i.e., the ‘subjects’ of activity), we also recognise *passive* structural elements, i.e., the *objects* on which behaviour is performed. In the domain of information-intensive organisations, which is the main focus of our language, these are usually *information objects* in the business layer and *data objects* in the application layer, but they may also be used to represent physical objects.

3 Business Layer Concepts

In this section we describe concepts for architectural descriptions that can be placed in the business layer of Figure 2. An example of a business layer model is shown in Figure 4.

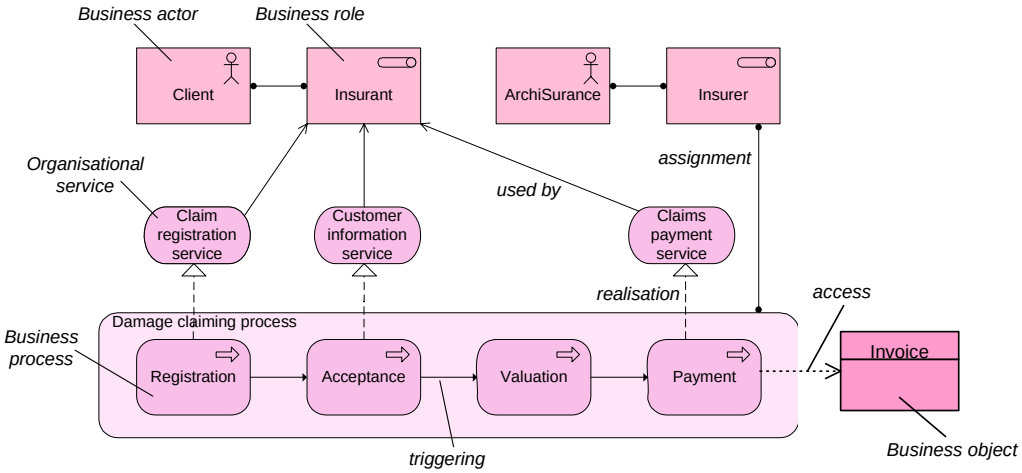


Figure 4. Example of a business-layer model.

In the example, Client and ArchiSurance are *business actors*, the active entities (the subjects) that perform behaviour such as business processes or functions. Business actors may be individual persons (e.g. customers or employees), but also groups of people and resources that have a permanent (or at least long-term) status within the organisations. To each actor a *business role* is assigned: Client has the role of Insurant and in this role makes use of two services offered by the insurance company. ArchiSurance plays the role of Insurer and this role it is responsible for the Damage claiming process; this is expressed by the assignment relation between the business process and the role. Note that the use of roles decouples (physical) actors from business activity and gives more flexibility in the allocation of activities to actors.

In the example a distinction has been made between “external” and “internal” behaviour of ArchiSurance. The externally visible behaviour is modelled by the concept *organisational service*, which represents a unit of functionality that is meaningful from the point of view of the environment; ArchiSurance has three such organisational services. Within ArchiSurance, these services are realised by one *business process*: the Damage claiming process, which consists of four subprocesses. Other concepts that can be used for modelling behaviour are *business functions* and *business interactions*. Business processes, functions and interactions, in turn, may use other services (internal to the organisation, but external to a smaller entity within the organisation).

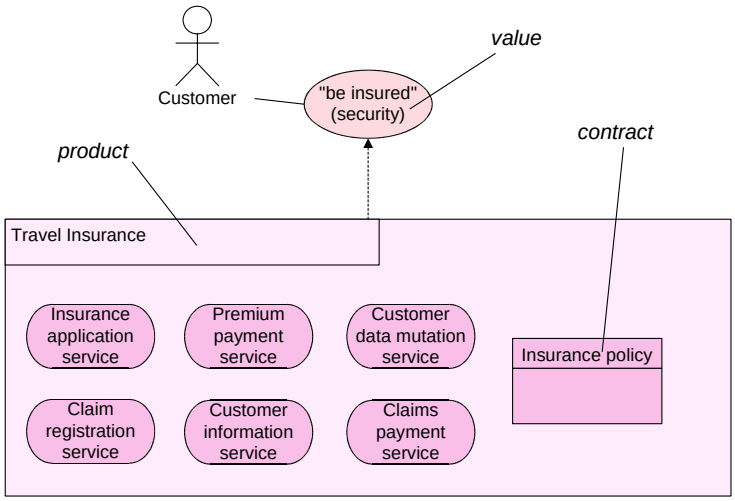


Figure 5. Services grouped into a product.

Services are grouped to form (financial or information) *products*, together with a *contract* that specifies the characteristics, rights and requirements associated with the product. Figure 5, for example, shows the Travel insurance product. These services are often organisational services, but application services may also be part of a product. This 'package' is offered as a whole to (internal or external) customers. 'Buying' a product gives the customer the right to use the associated services. The *value* of a product or service is what makes some party appreciate it. Value is often expressed in terms of money, but non-monetary value is also essential to business, for example, practical or functional value (including the right to use a service), and the value of information or knowledge. In our example, the value associated by the client of the travel insurance would typically be something like 'to be insured' or 'security'.

4 Application Layer Concepts

The main structural concept for the application layer is the *application component*. This concept is used to model any structural entity in the application layer: not just (reusable) software components that can be part of one or more applications, but also complete software applications, subapplications or information systems, such as the CRM system, the Policy administration, and the Financial application in the example of Figure 6. This concept is very similar to the UML *component* concept (Object Management Group, 2003b). *Data objects* are used in the same way as data objects (or object types) in well-known data modelling approaches, most notably the 'class' concept in UML class diagrams.

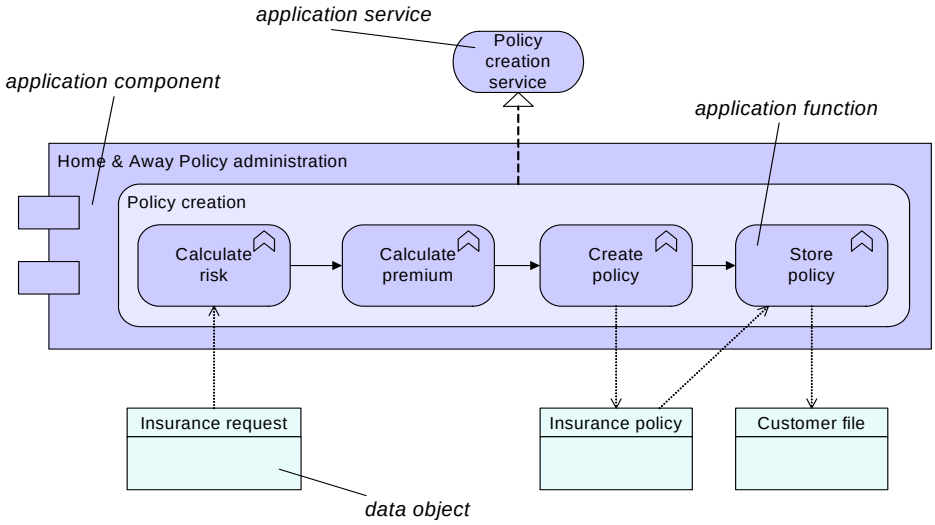


Figure 6. Example of an application-layer model.

In the purely structural sense, an *application interface* is the (logical) location where the services of a component can be accessed. In a broader sense (as used in, among others, the UML definition), an application interface also has some behavioural characteristics: it defines the set of operations and events that are provided by the component, or those that are required from the environment.

Behaviour in the application layer can be described in a way that is very similar to business layer behaviour. We make a distinction between the externally visible behaviour of application components in terms of *application services*, and the internal behaviour of these components to realise these services. This concept fits well within the current developments in the area of, e.g., web services.

An *application function* describes the internal behaviour of a component needed to realise one or more application services. An *application interaction* is the collaborative behaviour of two or more application components.

5 Technology Layer Concepts

The main structural concept for the technology layer is the *node*. This concept is used to model structural entities in the technology layer. Nodes come in two flavours: *device* and *system software*, both inspired by UML 2.0 (the latter is called *execution environment* in UML). A device models a physical computational resource, on which artifacts may be deployed for execution. An example is the zSeries mainframe Figure 7. System software represents the software environment for specific types of components and data objects, like the DB2 database in the figure. Typically, a node will consist of a number of subnodes, for example a device such as a server and an execution environment to model the operating system.

An *infrastructure interface* is the (logical) location where the infrastructural services offered by a node can be accessed by other nodes or by application components from the application layer. An *artifact* is a physical piece of information that is used or produced in a software development process, or by deployment and operation of a system. It is the representation, in the form of e.g. a file, of a data object or an application component, and can be assigned to (i.e., deployed on) a node.

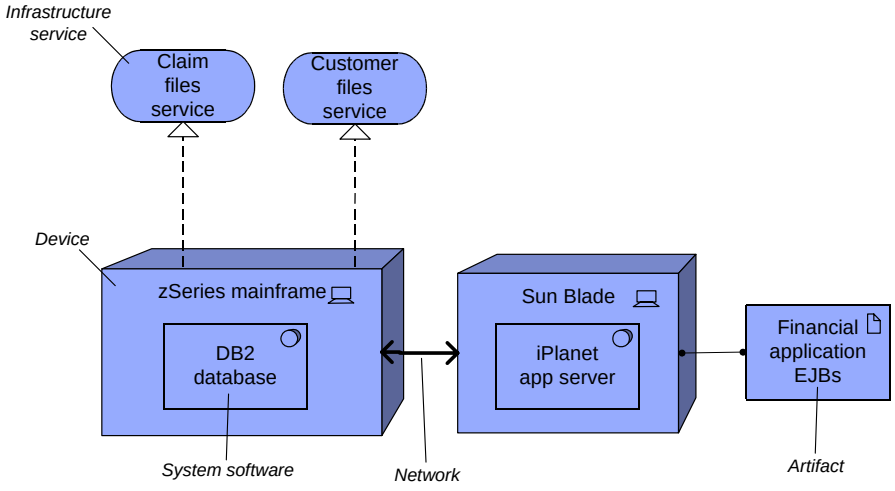


Figure 7. Example of a technology-layer model.

The interrelationships of components in the technology layer are mainly formed by communication infrastructure. The *communication path* models the relation between two or more nodes, through which these nodes can exchange information. The physical realisation of a communication path is a modelled with a *network*, i.e., a physical communication medium between two or more devices.

In the technology layer, the central behavioural concept is the *infrastructure service*. We do not model the internal behaviour of infrastructure components such as routers or database servers; that would add a level of detail that is not useful at the enterprise level of abstraction.

6 Relations

In the previous sections we have presented the concepts to model the business, application, and technology layers of an enterprise. In each of the layers presented thus far, different relations between concepts have been used:

- The *access* relation models the access of passive elements, e.g. business or data objects, by processes, functions or interactions.
- The *use* relation models the use of active or behavioural elements, e.g. the use of services by processes, functions or interactions, or the use of interfaces by roles, components or collaborations.
- The *composition* relation indicates that an object consists of a number of other objects, i.e., the lifecycles of the contained objects are tied to that of their container.
- The *aggregation* relation indicates that an object groups a number of other objects, but the grouped objects continue to have an independent lifecycle.
- The *assignment* relation links units of behaviour with active elements (e.g. roles, components) that perform them, roles with actors that fulfil them, or artifacts that are deployed on nodes.
- *Association* models a relation between objects that is not covered by another, more specific relation.
- The *realisation* relation links a logical entity with a more concrete entity that realises it.
- The *specialisation* relation indicates that an object is a specialisation of another object.
- The *triggering* relation describes the temporal or causal relations between processes, function, interactions and events.

As we did for the concepts used to describe the different conceptual domains, as much as possible we adopt corresponding relation concepts from existing standards. For instance, relation concepts such as composition, association, specialisation are taken from UML, while triggering is used in most business process modelling languages.

As we observed before, the architectural layers (business, application and technology) constitute some sort of hierarchy within an enterprise. A common way of looking at an enterprise is to start from the business processes and activities performed. These are carried out by some actor or role in the organisation, possibly supported by one or more business applications, or even fully automated. These activities, however, can also be viewed as *services* to this business process.

If we connect the separate models shown in the previous sections by means of services, we arrive at Figure 8, which shows a small example of an integrated and service-oriented enterprise architecture model.

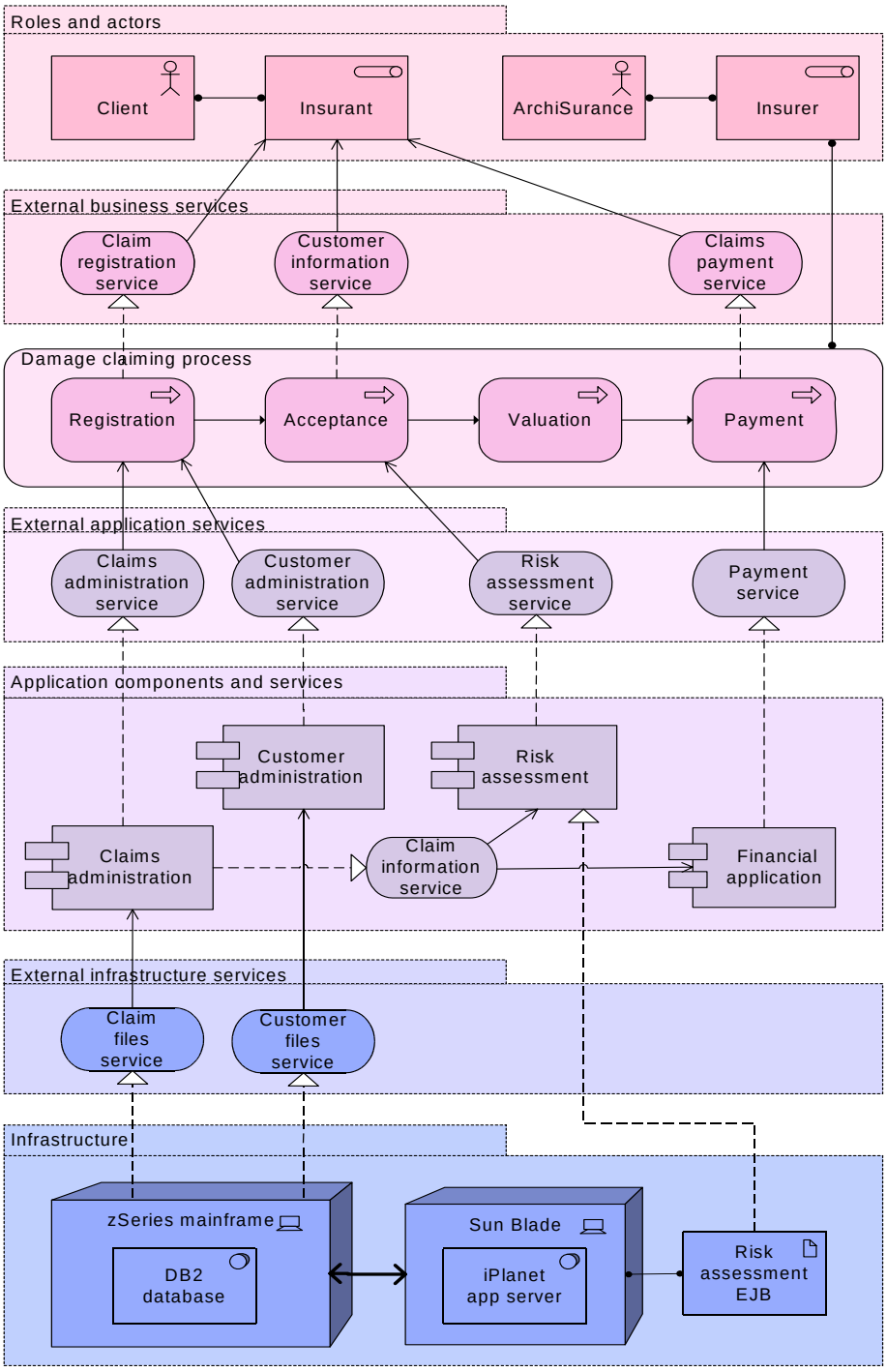


Figure 8. Example of an integrated enterprise architecture.

7 Example Case: ArchiSurance

ArchiSurance, a (fictious) company that originally provided home and travel insurance, has merged recently with two other insurance companies, PRO-FIT (car insurance) and LegallyYours (legal aid). To create insight in ArchiSurance's primary operations, the company is described in terms of its main business functions:

- Maintaining Customer Relations and Intermediary Relations: these business functions are responsible for the contacts of ArchiSurance with its customers and the intermediaries that sell its products. It handles customer questions and incoming claims, and performs marketing and sales.
- Contracting: this function does the 'back-office' processing of contracts. It performs risk analysis and ensures legally and financially correct contracts.
- Claims Handling: this function is responsible for handling insurance claims.
- Financial Handling: this function performs the regular premium collection, according to the insurance policies with customers as produced by Contracting, and handles the payment of insurance claims.
- Asset Management: this function manages the financial assets of ArchiSurance, e.g. by investing in stocks and bonds.

These business functions are very similar for most insurance companies and represent what is most stable about an enterprise.

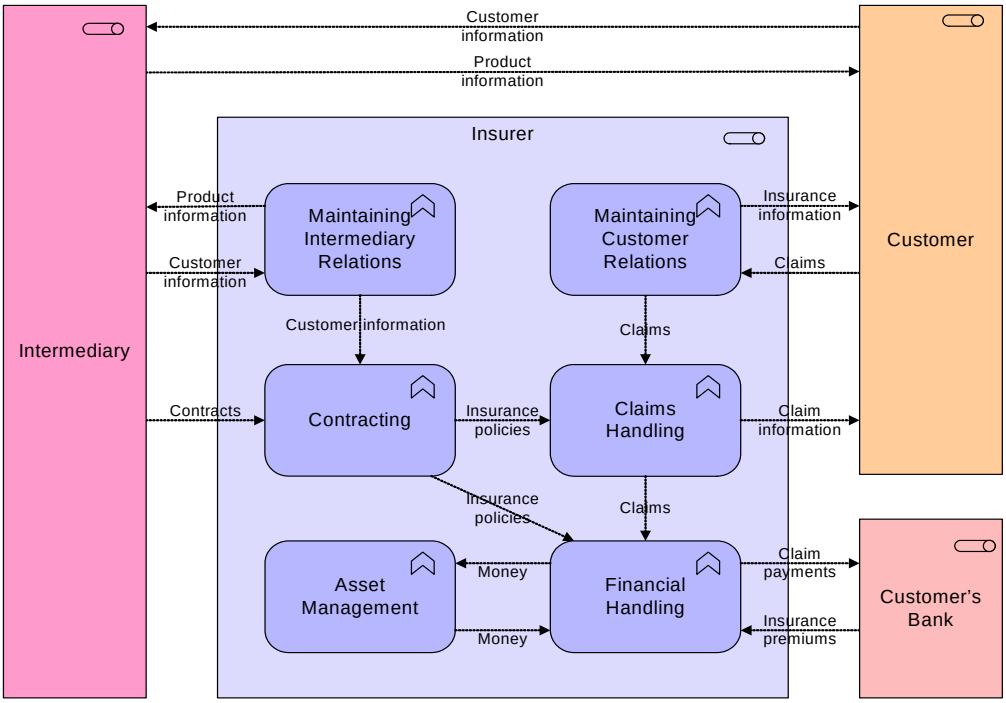


Figure 9. ArchiSurance business functions.

Post-merger integration is in full swing. The first step in the integration process has been the creation of a unified front office, comprising departments for managing relations with

customers on the one hand, and intermediaries on the other hand. However, behind this front office are still three separate back offices:

- Home & Away: this department was the original pre-merger ArchiSurance, responsible for home and travel insurance.
- Legal Aid: this is the old LegallyYours, responsible for legal aid and liability insurance.
- Car: this department is the core of the old PRO-FIT and handles car insurance, including some legal aid.

Furthermore, ArchiSurance is in the process of setting up a Shared Service Center for document processing, which will handle all document streams and performs scanning, printing, and archiving jobs. The company's structure is shown in Figure 10.

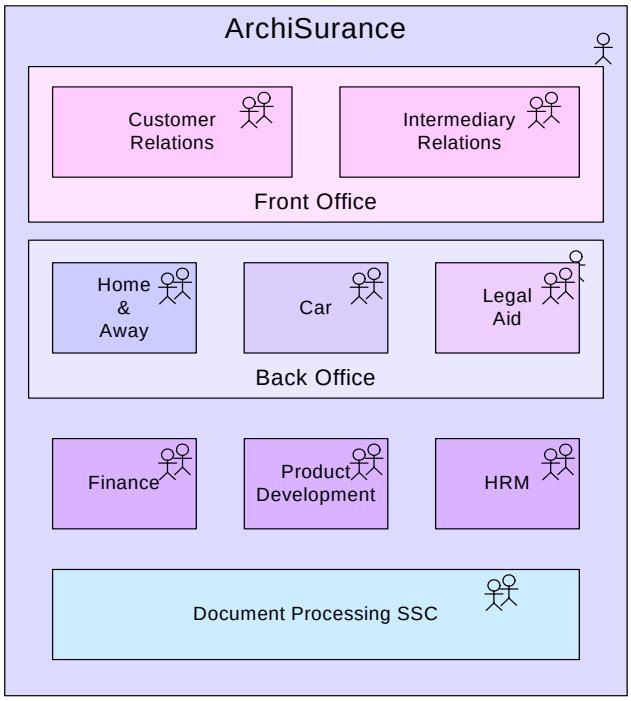


Figure 10. ArchiSurance departments.

As in many recently merged companies, IT integration is a problem. ArchiSurance want to move to a single CRM system, separate back-office systems for policy administration and finance, and a single document management system. However, Home & Away still have separate systems for the policy administration and the financial handling of premium collection and claims payment, and use the central CRM system and call center. The Car department have their own monolithic back-office system, but use the central CRM system and call center. The Legal Aid department have their own back- and front office systems (Figure 11).

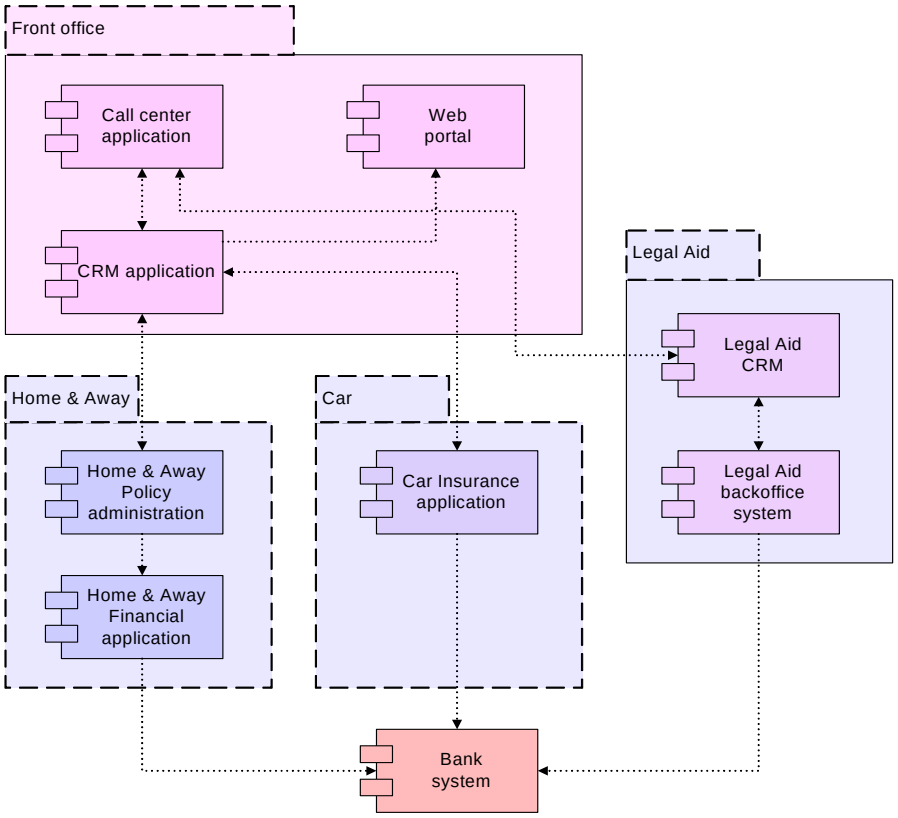


Figure 11. Applications grouped according to departments.

An important prerequisite for the changes in ArchiSurance's IT is that the IT integration should be "invisible" to ArchiSurance's clients: products & services remain the same. However, this is not a straightforward requirement. To illustrate the complexity of the relationships between products, business processes and IT support, Figure 12 shows the services provided by the Handle Claim business process, Figure 13 shows the relations between this business process and its supporting IT applications, and Figure 14 shows how a single service of these applications is realised. Note that this only shows these relations for a *single* business process! In general, many different business processes within the back office link the external products and services with the internal systems. This web of relations creates a major problem if we want to create insight in the IT support of ArchiSurance.

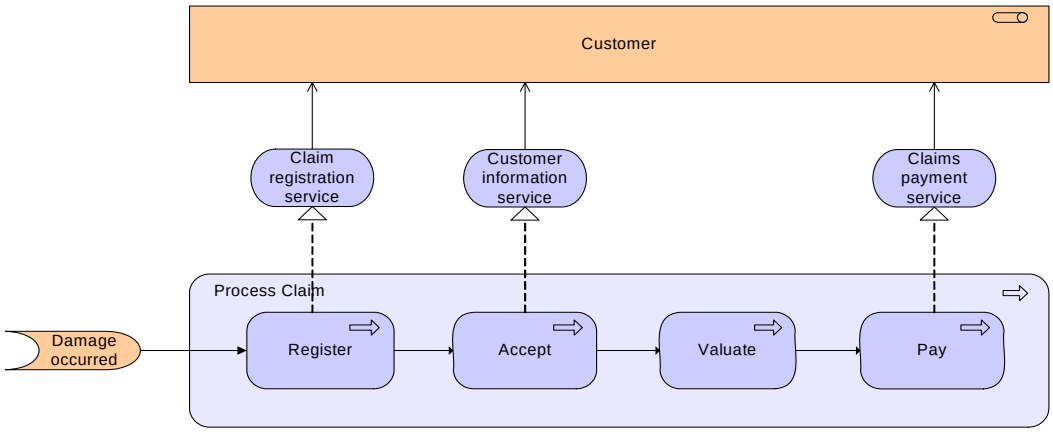


Figure 12. Business services provided by the Handle Claim business process.

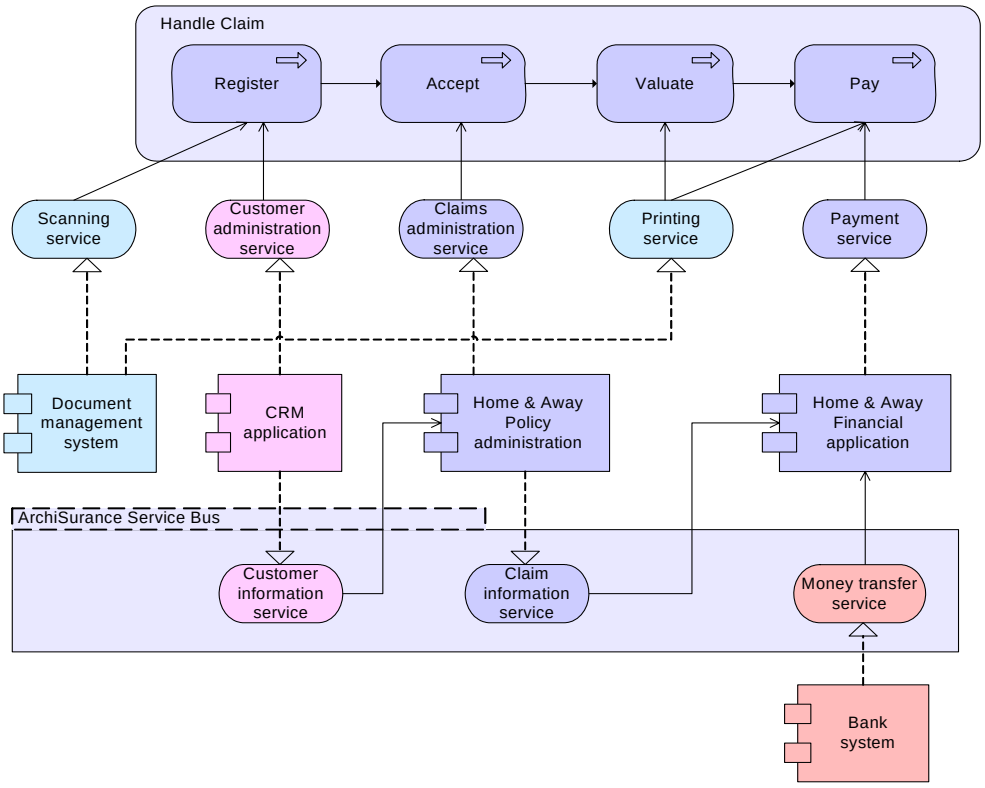


Figure 13. Relations between the Handle Claim business process and its IT support.

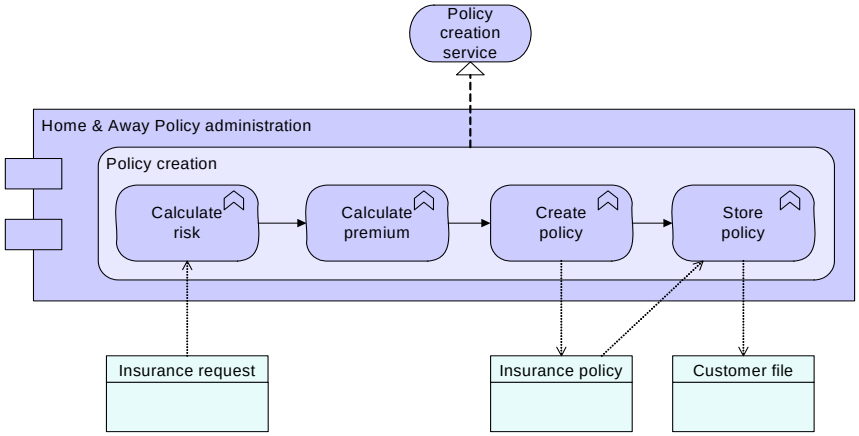


Figure 14. Realisation of the Policy creation application service by the Home & Away Policy administration.

The infrastructure on which the applications are deployed is a hybrid of traditional mainframe processing and more recent additions in the form of a server farm of Unix blade servers (Figure 15). A Network Attached Storage (NAS) server is used by the Unix machines, whereas the mainframe runs the usual system software such as a DBMS, message queuing middleware, and a CICS environment. The ArchiSurance network is connected to the intermediaries via the Internet, of course with the appropriate firewalling. Onto this infrastructure, the various logical applications are deployed in the form of physical artifacts such as EJBs; an example of this is given in Figure 16.

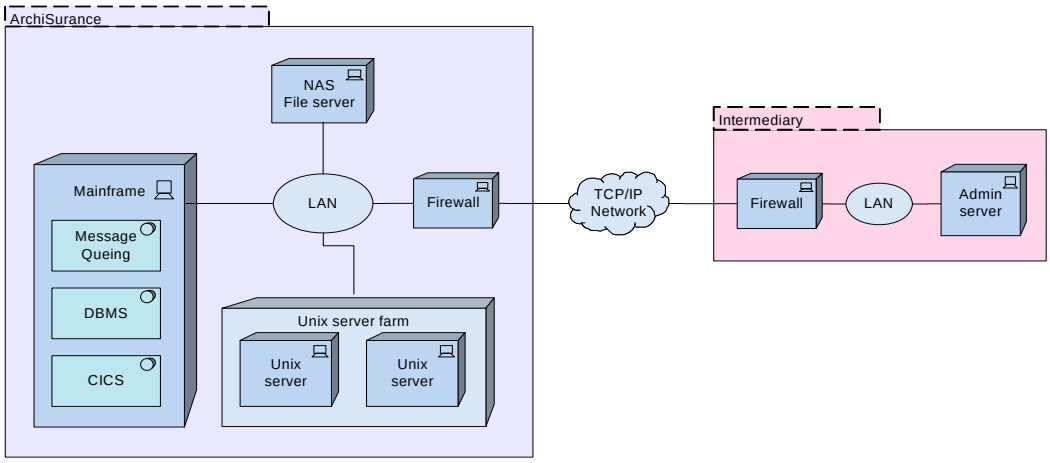


Figure 15. ArchiSurance infrastructure.

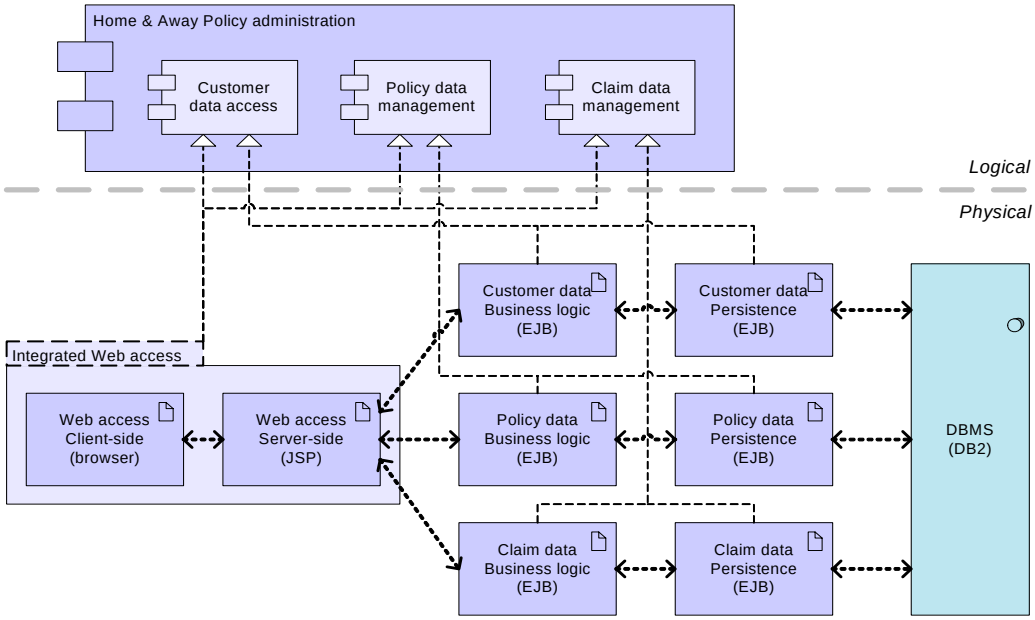


Figure 16. Mapping of logical application components of the ArchiSurance policy administration onto physical artifacts.

Appendix A - ArchiMate Language Metamodel

Figure 17 gives a summary of the ArchiMate concepts and their relationships.

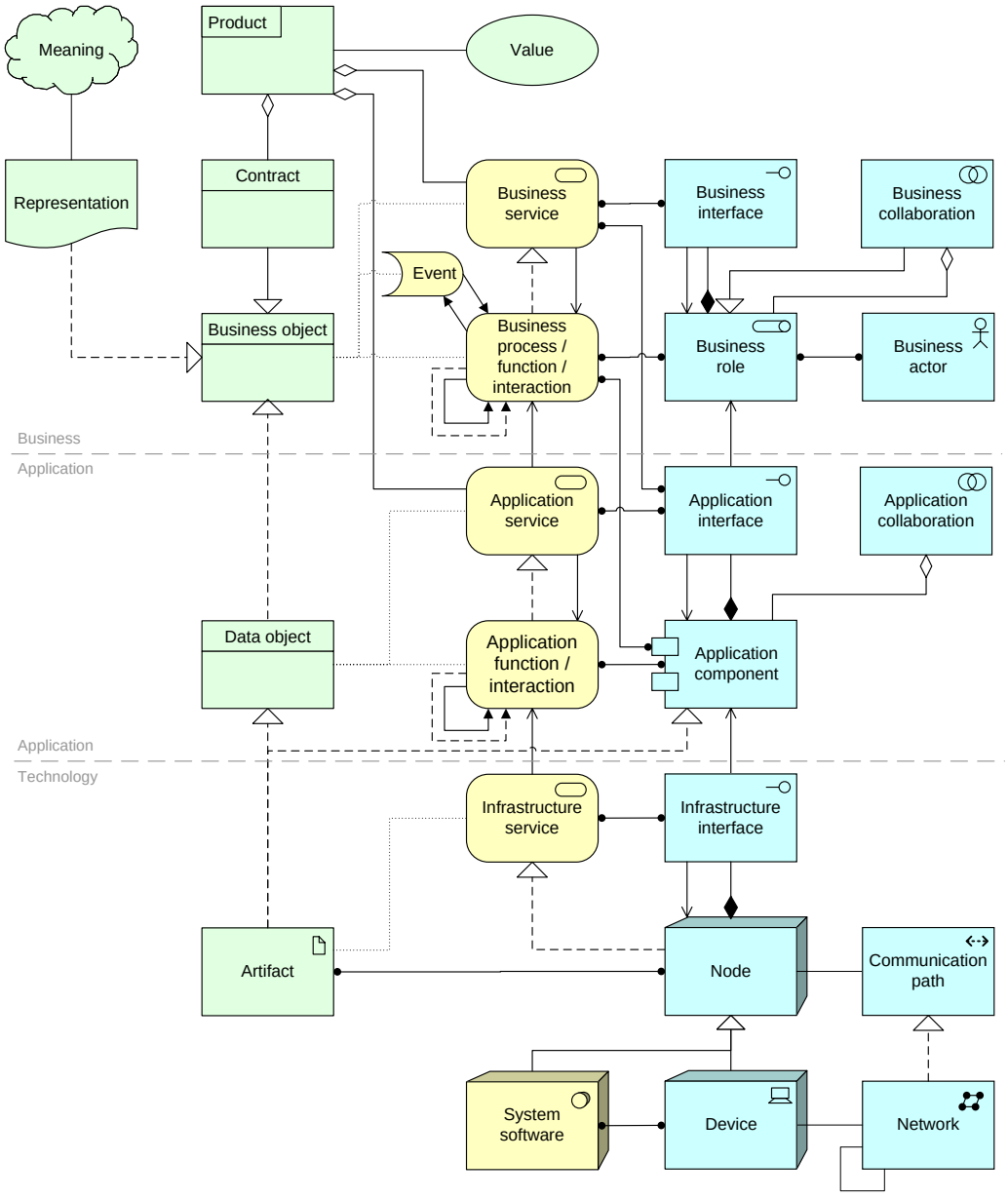


Figure 17. Overview of the main concepts and relationships.

A distinction is made between the “external” and “internal” behaviour of an organisation or system. For the external behaviour, the designation “service” is used. The service is the externally visible effect of processes, function or interactions, that can be used by other units of behaviour that require this service. The internal behaviour, on the other hand, represents what is required to realise this service. For the ‘consumers’ of a service the

internal behaviour of a system or organisation is usually irrelevant: they are only interested in the functional and non-functional results of the behaviour that are advertised by the service. In this way different layers can be related to each other (behaviour aspect) by means of services. Each layer makes its services available to the next higher layer.

As for the structure aspect, we could say that an (application) interface is the location where components (applications) in the application layer interact with (business) actors. Therefore, ‘interface’ can be considered a linking concept comparable to the service concept for the behaviour aspect.

Appendix B - Graphical Notation

In the picture below we summarise the graphical notation for the ArchiMate concepts and relations as proposed in the previous chapters. In most cases the graphical notation has been taken over from standards such as UML or other architecture tools. For the behaviour and structure concepts we propose a choice between two notations. Take for instance the behaviour concepts, which are represented by a rectangle with rounded angles or oval shape. In order to distinguish the different concepts an icon is placed within these graphical representations. We propose to use these icons as a possible notation as well.