7 Viewpoints and Visualisation

Establishing and maintaining a coherent enterprise architecture is clearly a complex task, because it involves many different people with differing backgrounds using various notations. In order to get to grips with this complexity, researchers have initially focused on the definition of architectural frameworks for classifying and positioning the various architecture descriptions with respect to each other. A problem with looking at enterprise architecture through the lens of an architectural framework is that it categorises and divides architecture descriptions rather than providing insight into their coherence.

To integrate the diverse architecture descriptions, we advocate an approach in which architects and other stakeholders can define their own views of the enterprise architecture. In this approach views are specified by viewpoints. *Viewpoints* define abstractions on the set of models representing the enterprise architecture, each aimed at a particular type of stakeholder and addressing a particular set of concerns. Viewpoints can be used both to view certain aspects in isolation, and for relating two or more aspects.

This chapter focuses on the use of views of enterprise architectures, both to create and manipulate architectural models and to give others insight into the architectures being describe. We describe the use of viewpoints in communication, and the distinction between an architecture model, a view of that model, and its visualisation and manipulation. We give guidelines for the selection and use of viewpoints, and we outline a number of viewpoints on the ArchiMate language that can be used by architects involved in the creation or change of enterprise architecture models.

7.1 Architecture Viewpoints

In this section we discuss the notion of views and viewpoints as basic tools in communicating about architectures. In the context of enterprise architectures, a viewpoint is typically used for activities like design, analysis, obtaining commitment, formal decision making, etc. As we argued in Chap. 4, we regard all of these activities to be communicative in nature.

As defined in Sect. 3.2.4, a *viewpoint* essentially prescribes the concepts, models, analysis techniques, and visualisations that are to be used in the construction of different views of an architecture description. A *view* is typically geared towards a set of stakeholders and their concerns. Simply put, a view is what you see, and a viewpoint describes from where you are looking.

In discussing the notion of viewpoint, we will first provide a brief overview of the origin of viewpoints. This is followed by a more precise definition of viewpoints, and the concept of viewpoint *frameworks*.

7.1.1 Origin of Viewpoints

The concept of viewpoint is not new. For example, in the mid 1980s, Multiview (Wood-Harper et al. 1985) already introduced the notion of views. In fact, Multiview identified five viewpoints for the development of (computerised) information systems: Human Activity System, Information Modelling, Socio-Technical System, Human–Computer Interface, and the Technical System. During the same period in which Multiview was developed, the so-called CRIS Task Group of IFIP Working Group 8.1 developed similar notions, where stakeholder views were reconciled via appropriate 'representations'. Special attention was paid to disagreement about which aspect (or *perspective*) was to dominate the system design (namely, 'process', 'data', or 'behaviour'). As a precursor to the notion of *concern*, the CRIS Task Group identified several *human roles* involved in information system development, such as executive responsible, development coordinator, business analyst, business designer (Olle et al. 1988).

The use of viewpoints is not limited to the information systems community, it was also introduced by the software engineering community. In the 1990s, a substantial number of software engineering researchers worked on what was phrased as 'the multiple perspectives problem' (Finkelstein et al. 1992, Kotonya and Sommerville 1992, Nuseibeh 1994, Reeves et al. 1995). By this term, the authors referred to the problem of how to organise and guide (software) development in a setting with many actors, using diverse representation schemes, having diverse domain knowledge, and using different development strategies. A general framework has been developed in order to address the diverse issues related to this problem (Finkelstein et al. 1992, Kotonya and Sommerville 1992, Nuseibeh 1994). In this framework, a viewpoint combines the notion of *actor*, *role*, or *agent* in the development process with the idea of a *perspective* or *view* which an actor maintains. A viewpoint is more than a partial specification; in addition, it contains partial knowledge of how further to develop that partial

specification. These early ideas on viewpoint-oriented software engineering have found their way into the IEEE 1471 standard for architecture description (IEEE Computer Society 2000) on which we have based our definitions below.

7.1.2 Architecture Viewpoints

In the context of architecture, viewpoints provide a means to focus on particular aspects of an architecture description. These aspects are determined by the concerns of the stakeholders with whom communication takes place. What should and should not be visible from a specific viewpoint is therefore entirely dependent on argumentation with respect to a stakeholder's concerns. Viewpoints are designed for the purpose of serving as a means of communication in a conversation about certain aspects of an architecture. Though viewpoints can be used in strictly uni-directional, informative conversations, they can in general also be used in bi-directional classes of conversations: the architect informs stakeholders, and stakeholders give their feedback (critique or consent) on the presented aspects. What is and what is not shown in a view depends on the scope of the viewpoint and on what is relevant to the concerns of the stakeholders. Ideally, these are the same, i.e., the viewpoint is designed with the specific concerns of a stakeholder in mind. Relevance to a stakeholder's concern, therefore, is the selection criterion that is used to determine which objects and relations are to appear in a view.

Below we list some examples of stakeholders and their concerns, which could typically serve as the basis for the definition/selection of viewpoints:

- Upper-level management: How can we ensure our policies are followed in the development and operation of processes and systems? What is the impact of decisions (on personnel, finance, ICT, etc.)? Which improvements can a new system bring to a pre-existing situation in relation to the costs of acquiring that system?
- Middle-level management: What is the current situation with regards to the computerised support of a business process?
- End user: What is the potential impact of a new system on the activities of a prospective user?
- Architect: What are the consequences for the maintainability of a system with respect to corrective, preventive, and adaptive maintenance?
- Operational manager: What new technologies do we need to prepare for? Is there a need to adapt maintenance processes? What is the impact of changes to existing applications? How secure are the systems?

- Project manager (of system development project): What are the relevant domains and their relations? What is the dependence of business processes on the applications to be built? What is their expected performance?
- System developer: What are the modifications with respect to the current situation that need to be performed?
- System administrators: What is the potential impact of a new system on the work of the system administrators that are to maintain the new system?

In line with the IEEE 1471 standard, and based on the detailed definition given in Proper (2004) we define a viewpoint as follows:

> **Viewpoint**: a specification of the conventions for constructing and using views.

This should also involve the various 'ways of …' that we outlined in Sect. 3.2.5, but in this chapter we will focus on the selection of the content of views, the visual representation of this content, and the typical use of these viewpoints, i.e., on the ways of modelling, communicating, and using. The 'way of supporting', i.e., tool support for views, will be addressed in Chap. 10, and the 'way of working' has already been addressed in Chap. 6.

7.1.3 Viewpoint Frameworks

In the context of architecture descriptions, a score of viewpoint frameworks exists, leaving designers and architects with the burden of selecting the viewpoints to be used in a specific situation. Some of these frameworks of viewpoints are: the Zachman framework (Zachman 1987), Kruchten's 4+1 view model (Kruchten 1995), RM-ODP (ITU 1996), and TOGAF (The Open Group 2002). These frameworks have usually been constructed by their authors in an attempt to cover all relevant aspects/concerns of the architecture of some class of systems. In practice, numerous large organisations have defined their own frameworks of viewpoints by which they describe their architectures. We shall discuss two of these framework in more detail below.

The '4+1' View Model

Kruchten (1995) introduced a framework of viewpoints (a view model) comprising five viewpoints. The use of multiple viewpoints is motivated by the observation that it 'allows to address separately the concerns of the

various stakeholders of the architecture: end-user, developers, systems engineers, project managers, etc., and to handle separately the functional and non-functional requirements'.

The goals, stakeholders, concerns, and meta-model of the 4+1 framework can be presented, in brief, as in Table 7.1. Note that in Kruchten (2000), the viewpoints have been renamed; physical viewpoint → deployment viewpoint, development viewpoint → implementation viewpoint, and scenario viewpoint → use-case viewpoint, better to match the terminology of UML.

The framework proposes modelling concepts (the meta-model) for each of the specific viewpoints. It does so, however, without explicitly discussing how these modelling concepts contribute to the goals of the specific viewpoints. One might, for example, wonder whether object classes, associations, etc., are the right concepts for communication with end users about the services they require from the system. The 4+1 framework is based on experiences in practical settings by its author.

Table 7.1. Kruchten's '4+1' view model.

Viewpoint	*Logical*	*Process*	*Development*	*Physical*	*Scenarios*
Goal	Capture the services which the system should provide	Capture concurrency and sychronisation aspects of the design	Describe static organisation of the software and its development	Describe mapping of software onto hardware, and its distribution	Provide a driver to discover key elements in design Validation and illustration
Stakeholders	Architect End users	Architect System designer Integrator	Architect Developer Manager	Architect System designer	Architect End users Developer
Concerns	Functionality	Performance Availability Fault tolerance ...	Organisation Reuse Portability ...	Scalability Performance Availability ...	Understandability
Meta-model	Object classes Associations Inheritance ...	Event Message Broadcast ...	Module Subsystem Layer ...	Processor Device Bandwidth ...	Objects Events Steps ...

RM-ODP

The Reference Model for Open Distributed Processing (RM-ODP) (ITU 1996) was produced in a joint effort by the international standard bodies

ISO and ITU in order to develop a coordinating framework for the standardisation of open distributed processing. The resulting framework defines five viewpoints: *enterprise*, *information*, *computation*, *engineering* and *technology*. The modelling concepts used in each of these views are based on the object-oriented paradigm.

The goals, concerns, and associated meta-models of the viewpoints identified by the RM-ODP can be presented, in brief, as in Table 7.2.

Table 7.2. The RM-ODP viewpoints.

Viewpoint	*Enterprise*	*Information*	*Computational*	*Engineering*	*Technology*
Goal	Capture purpose, scope, and policies of the system	Capture semantics of information and processing performed by the system	Express distribution of the system in interacting objects	Describe design of distribution-oriented aspects of the system	Describe choice of technology used in the system
Concerns	Organisational requirements and structure	Information and processing required	Distribution of system Functional decomposition	Distribution of the system, and mechanisms and functions needed	Hardware and software choices Compliancy to other views
Meta-model	Objects Communities Permissions Obligations Contract ...	Object classes Associations Process ...	Objects Interfaces Interaction Activities ...	Objects Channels Node Capsule Cluster ...	Not stated explicitly

RM-ODP provides a modelling language for each of the viewpoints identified. It furthermore states: 'Each language [for creating views/models conforming to a viewpoint] has sufficient expressive power to specify an ODP function, application or policy from the corresponding viewpoint.' RM-ODP does not explicitly associate viewpoints to a specific class of stakeholders. This is left implicit in the concerns which the viewpoints aim to address.

7.2 Models, Views, and Visualisations

An important principle in our approach is the separation of the content and the presentation or visualisation of a view. This separation is not explicitly made in the IEEE standard, but it has important advantages. It facilitates the use of different visualisation techniques on the same modelling con-

cepts, and vice versa. Operations on the visualisation of a view, e.g., changing its layout, need not change its content.

The view content, referred to as the 'view' in the remainder of this chapter, is a selection or derivation from a (symbolic) model of the architecture, and is expressed in terms of the same modelling concepts. The presentation or notation of this view, referred to as 'visualisation' in the remainder, can take many forms, from standard diagrams to tables, cartoons, or even dynamic visualisations like movies. Editing operations on this visualisation can lead to updates of the view and of the underlying model. The creation and update of both the view and the visualisation are governed by a viewpoint. This viewpoint is jointly defined and/or selected in an iterative process by architect and stakeholder together. This is illustrated in Fig. 7.1.

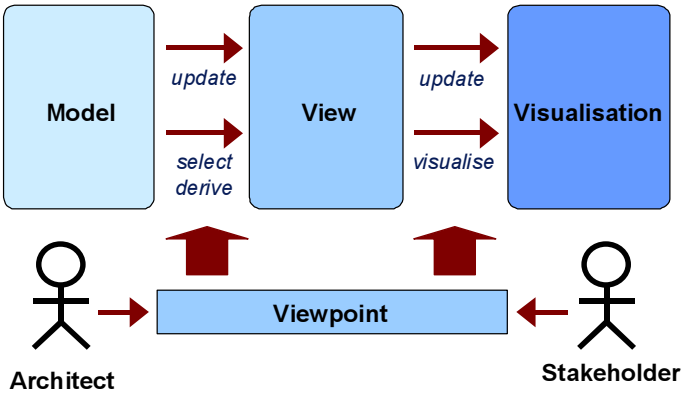

Fig. 7.1. Separation of concerns: model, view, visualisation, and viewpoint.

The separation between view and visualisation is based on the notion of 'meaning'. In Chap. 3 we introduced the concept of the signature of an architecture as its alphabet: that is, the set of symbols used to describe the concepts of the architecture and the relations among these concepts. This idea can also be used to clarify the distinction between view and its visualisation. A further discussion of these formal foundations can be found in Chap. 8.

A view stripped from its visual properties can be formalised just like any other model, e.g., by defining its signature, as outlined in Chap. 3. By formalising its relation with an underlying model, a view's quality and consistency can be greatly enhanced and new opportunities for its use may arise, e.g., in changing the underlying models by interacting with such a view.

7.2.1 Example: Process Illustrations

To illustrate the difference between a view and its visualisation, we introduce the *process illustration* viewpoint. This viewpoint illustrates a process model in an informal way for employees and managers. A process illustration is derived from a model of the architecture using a set of translation and abstraction rules. As process illustrations are meant for communicating the coherence between business processes, they typically abstract from details regarding the applications and technology involved. Moreover, process illustrations do not apply abstract concepts and notations, but rather use recognisable terms and intuitive notations.

A process illustration of the Car Tax Collection process is depicted in Fig. 7.2. The figure shows the various subprocesses involved and the information flows between them. The figure is derived from an ArchiMate model via a series of translation and abstraction rules, for instance to replace abstract shapes with meaningful symbols, abstract from complex relations, and visually group all objects and relations that belong to or happen within a certain actor.

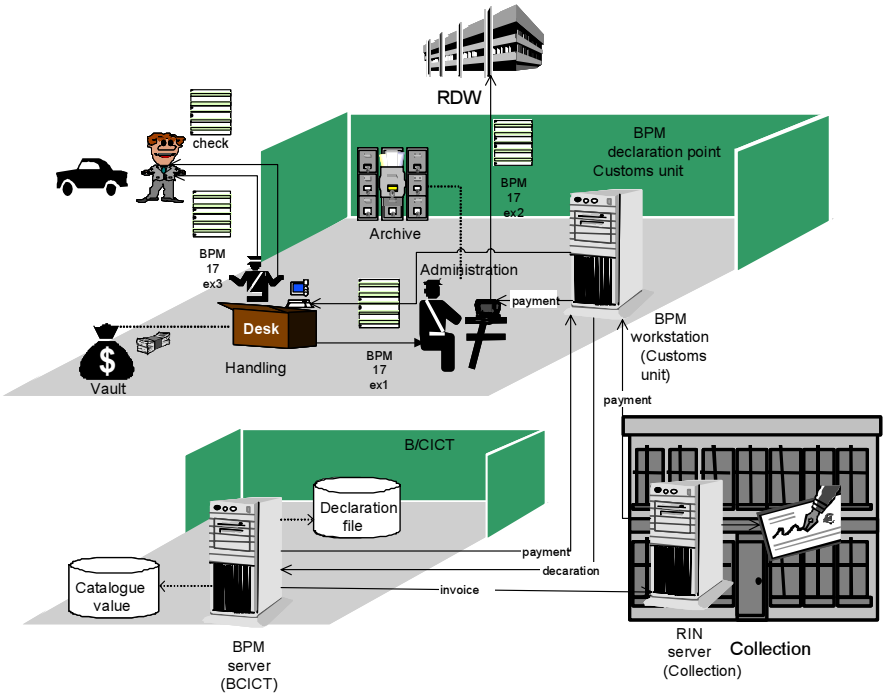

Fig. 7.2. Process illustration of the Car Tax Collection process.

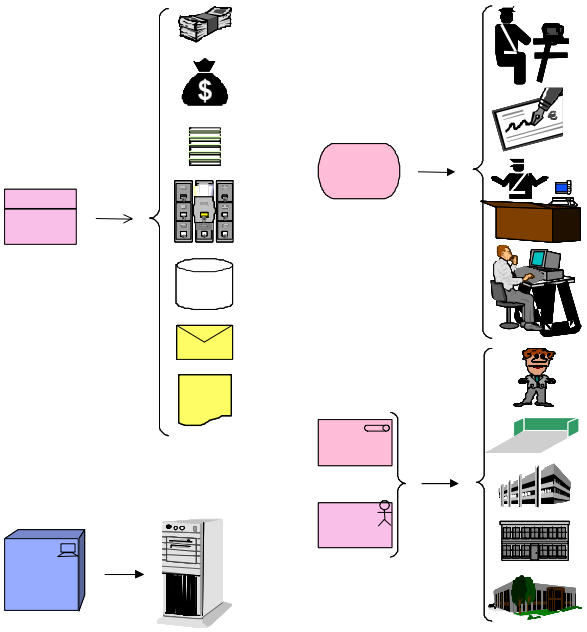

Fig. 7.3. Translation rules.

In Fig. 7.3 you can see a number of presentation rules that can be applied in the 'model-to-illustration' derivation. The basic idea behind these rules is to find suitable and intuitive graphic symbols that will replace ArchiMate shapes. These rules apply to ArchiMate concepts for which there is an immediate correspondent in the process illustration notation (i.e., actor, role, device, service, business object, etc.).

Of course, many other rules can be added here. For instance, rules referring to a specific layout of the final drawing or to the more extensive usage of 3D graphic symbols can increase the readability and usability of the final drawing.

7.2.2 Example: Landscape Maps

A more complex example to illustrate the differences between a model, a view, and its visualisation, is the *landscape map* viewpoint. Landscape maps, as defined in van der Sanden and Sturm (1997), are a technique for visualising enterprise architectures. They present architectural elements in the form of an easy-to-understand 2D 'map'. A landscape map view of architectures provides non-technical stakeholders, such as managers, with a high-level overview, without burdening them with the technicalities of architectural drawings.

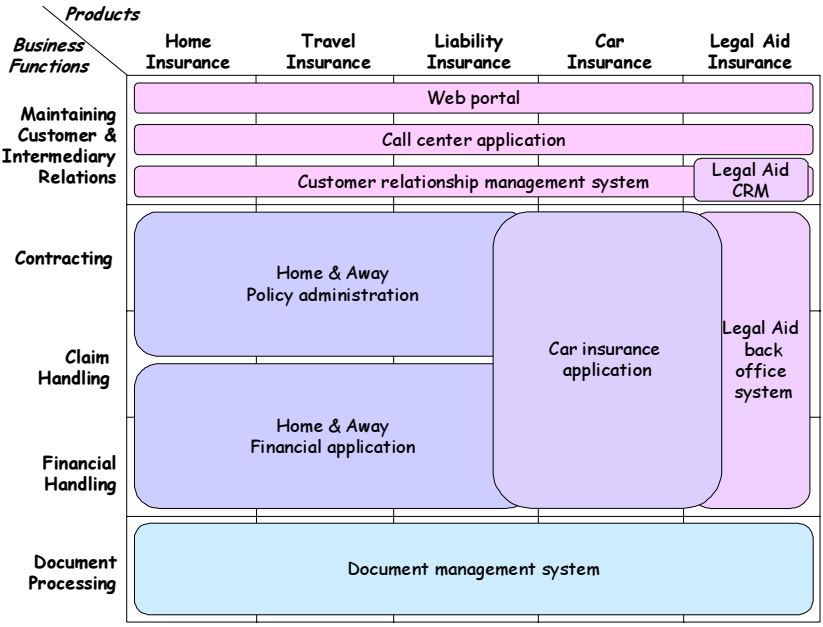

Fig. 7.4. Landscape map of ArchiSurance.

Many systems used by many processes realising various products and services comprise too much detail to display in a single figure. This is a typical example of where landscape maps can help. In Fig. 7.4, a landscape map is depicted that shows which information systems support the operations of our fictitious insurance company ArchiSurance. The vertical axis represents the company's business functions; the horizontal axis shows its insurance products. An application rectangle covering one or more cells means that this particular function/product pair is supported by the application, e.g., contracting of a legal aid insurance is supported by the legal aid back-office system. The visualisation chosen makes it immediately obvious to the viewer that there is (possibly unwanted) overlap between applications, as is the case in the Car insurance application and the Legal Aid CRM system. Clearly, landscape maps are a richer representation than cross-reference tables, which cover only two dimensions. In order to obtain the same expressive power of a landscape map two cross-reference tables would be necessary; but even then, you would get a presentation that is not as insightful and informative as a landscape map.

The dimensions of the landscape maps can be freely chosen from the architecture that is being modelled. In practice, dimensions are often chosen from different architectural domains, for instance business functions, products and applications, etc. In most cases, the vertical axis represents

behaviour such as business processes or functions; the horizontal axis represents 'cases' for which those functions or processes must be executed. These 'cases' can be different products, services, market segments, or scenarios. The third dimension represented by the cells of the matrix is used for assigning resources like information systems, infrastructure, or human resources.

The visualisation of architecture models as landscape maps is based on architecture relations. The dimensions that are used in the landscape maps determine which relations are used. For instance, the landscape map in Fig. 7.4 relates business functions (Contracting, Claim Handling, etc.) to products (Home insurance, Travel insurance, etc.) to applications (Web portal, Car insurance application, etc.). The relation between business functions, products, and applications is not directly supported by relations in the underlying model. Rather, this needs to be inferred indirectly: a product comprises a number of business services, which are realised by business processes and functions, which use (the application services of) application components. For this inference, the formalisation of the underlying symbolic models and the rules for the composition of relations described in Chaps. 3 and 4 are indispensable.

For landscape maps to be of practical use, the visualisation must be intuitive and easy to understand. To a large extent, the choice of the axes and the ordering of the rows and columns determine the layout of a landscape map. If adjacent cells in the plane have the same value assigned, they can be merged to form a single shape. If there are no other criteria for ordering the axes such as time or priority, changes to the ordering can be used to optimise the layout of shapes in the plane, and also to limit their number. Various layout optimisation algorithms can be employed, and user manipulation of, for example, the order of rows and columns may also help in creating a pleasing visualisation.

Summarising, in developing the landscape map viewpoint, it has been fruitful to distinguish the operation on the model from the visualisation of the view, because they are completely different concerns. The same holds for the other viewpoints we have defined. To separate these concerns, views have to be distinguished from their visualisation.

7.3 Visualisation and Interaction

The distinction we make between a model and its visualisation naturally leads to the concept of interactive visualisation; that is, visualisation which can change the model due to interaction with a stakeholder. Interaction has traditionally been considered as something completely outside the model

and the view. Interaction is at least partly a visualisation issue: for example, when a user draws an object on the canvas of some tool. However, it can also partly be defined as part of the model and view, since the object the user draws may be put in the underlying model or view as well.

These two considerations have led to a new visualisation and interaction model for enterprise architectures in ArchiMate. Its goal is that interaction is separated from updating the model, or from its visualisation.

7.3.1 Actions in Views

The effect of a user interacting with the visualisation can be an update of the view. But where will this be defined? Clearly, the visualisation itself is 'dumb' and does not know about the semantics of the view. Hence, rules for changing the view cannot be tied to the visualisation and must be defined in the view itself. This is why we introduce the notion of *actions in views*. Consider for example a landscape map view, and a user who interacts with this view by moving an application to another business function. Does the relation between the interaction with the landscape map and the update of the model *mean* something? Obviously the relation between the move in the landscape map leads to an update of the underlying model or view, and thus means something.

In Sect. 6.2.3 we have identified a number of basic modelling actions, such as introducing, refining, abandoning, abstracting, and translating a concept in a model. These actions operate on the architecture model or view, not on its visualisation. However, most changes to a model will be conducted by a user who changes a visualisation of that model. Hence, we need to define the ways in which a user can manipulate these visualisations and the effects on the underlying model in terms of these basic modelling actions. We can then relate these actions to the manipulations of the visualisation by making the actions part of the view being visualised.

Thus, a clear separation of model and visualisation leads to a separation of concerns in tool building. An extremely generic visualisation engine can be constructed that does not need to know about the semantics of the models it displays. If we define the possible actions together with the views, a generic editor can be configured by this set of actions.

The actions in views should be defined in terms of the effects they have on elements of the underlying model. For example, consider a view of a business process model, and an action that merges two processes into a single process. Issues that are relevant for the action of merging processes are the effects of the merger: for example, the removal of processes, addi-

tion of a new process, transferring some relations from an old, removed process to a new process.

For each viewpoint, we define a set of actions. For example, for the landscape map viewpoint we define the move of an application to another cell, we define changing the columns and rows of the matrix, and we define the addition and deletion of applications. Moreover, we must determine for each action which parameters it needs as input, and define the consequences of executing the action.

When actions for each view have been defined, we can go one step further and define the relation between actions. One important relation is that one action may consist of a set of simpler actions. For example, consider an architect or stakeholder that wishes to change an existing landscape map. First the effects of this change on the underlying model need to be assessed. Some changes may be purely 'cosmetic' in nature, e.g., changing the colour of an object. Other changes need to be propagated to the underlying model by invoking one of the basic modelling actions of Sect. 6.2.3, e.g., if an object is added or deleted.

Mapping a seemingly simple change to the map onto the necessary modifications of the model may become quite complicated. Since a landscape map abstracts from many aspects of the underlying model, such a mapping might be ambiguous: many different modifications to the model might correspond to the same change of the landscape map. Human intervention is required to solve this, but a landscape map tool might suggest where the impact of the change is located.

In the example of Fig. 7.4, you may for instance want to remove the seemingly redundant Legal aid CRM system by invoking a 'remove overlap' operation on this object. This operation influences both the visualisation and the architectural model. The effects of the operation on the underlying model are shown in Fig. 7.5. First, you select the object to be removed, in this case the Legal Aid CRM system. The envisaged tool colours this object and maps it back onto the underlying object in the architecture. Next, the relations connecting this object to its environment are computed, possibly using the impact-of-change analysis techniques described in Chap. 8 (the second part of Fig. 7.5). Here, this concerns the relations of Legal Aid CRM to the Web portal and the Legal Aid back-office system. These relations will have to be connected to one or more objects that replace the objects that are to be removed. Since we have chosen a 'remove overlap' operation, the landscape tool computes with which other objects Legal Aid CRM overlaps, in this case the CRM system. The relations formerly connecting Legal Aid CRM are then moved to the other CRM system, unless these already exist (e.g., the relation with the Web portal).

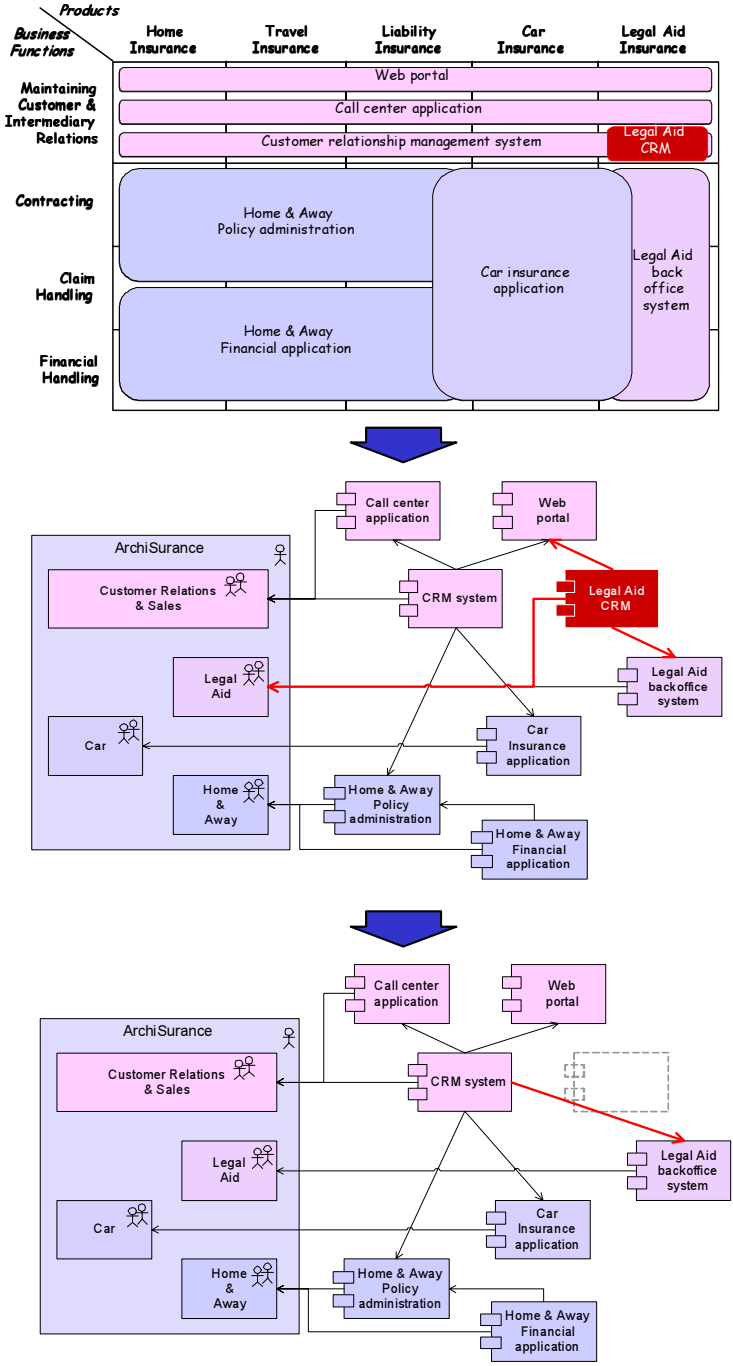

Fig. 7.5. Editing a landscape map.

Naturally, this scenario presents an ideal situation with minimal user intervention. In reality, a tool cannot always decide how a proposed change is to be mapped back onto the model, and may only present the user with a number of options. For example, if the functionality of the Legal Aid CRM system overlaps with more than one other system, remapping its relations requires knowledge about the correspondence between these relations and the functions realised by these other systems.

Implementing a tool that realises this 'actions in views' concept is not a trivial task. In Chap. 10, we will describe the design of a prototype tool that provides a proof of concept of these ideas.

7.4 Creating, Selecting, and Using Viewpoints

It is interesting to note that both of the discussed frameworks of viewpoints (Sect. 7.1.3) do not provide an explicit motivation for their choice regarding the modelling concepts used in specific viewpoints. When using one of the two frameworks, architects will not find it difficult to select a viewpoint for the modelling task at hand. However, this 'ease of choice' is more a result of the limitation of the selections of options available (one is limited to the number of viewpoints provided by the framework) than the result of a well-motivated choice about the viewpoint's utility towards the tasks at hand.

One should realise that a well-integrated set of viewpoints (such as the ArchiMate viewpoints) brings more (utility!) to a development project than the sum of its parts! Among other things, it allows views to be more easily related and integrated into a consistent whole. However, defining such an integrated viewpoint framework is an expensive undertaking. This means that even though a pre-existing (off-the-shelf) viewpoint framework may not be the ideal answer to an architect's specific communication needs, the alternative strategy of defining a tailor-made viewpoint framework for each development project is likely to be too costly. Hence our attention to defining 'ad hoc' viewpoints relative to a predefined modelling language (i.e., meta-model) as a compromise between fixed viewpoints and free viewpoints.

7.4.1 Classification of Viewpoints

As we can see from the list of stakeholders in Sect. 7.1.2, an architect is confronted with many different types of stakeholders and concerns. To help the architect in selecting the right viewpoints for the task at hand, we

introduce a framework for the definition and classification of viewpoints and views. The framework is based on two dimensions, *purpose* and *content*. The following three types of architecture support define the purpose dimension of architecture views (Steen et al. 2004):

- **Designing**: Design viewpoints support architects and designers in the design process from initial sketch to detailed design. Typically, design viewpoints consist of diagrams, like those used in UML.
- **Deciding**: Decision support views assist managers in the process of decision making by offering an insight into cross-domain architecture relations, typically through projections and intersections of underlying models, but also by means of analytical techniques. Typical examples are cross-reference tables, landscape maps, lists, and reports.
- **Informing**: These viewpoints help to inform any stakeholder about the enterprise architecture, in order to achieve understanding, obtain commitment, and convince adversaries. Typical examples are illustrations, animations, cartoons, flyers, etc.

The goal of this classification is to assist architects and others to find suitable viewpoints given their task at hand, i.e., the purpose that a view must serve and the content it should display. With the help of this framework, it is easier to find typical viewpoints that might be useful in a given situation. This implies that we do not provide an orthogonal categorisation of each viewpoint into one of three classes; these categories are not exclusive in the sense that a viewpoint in one category cannot be applied to achieve another type of support. For instance, some decision support viewpoints may be used to communicate to any other stakeholders as well.

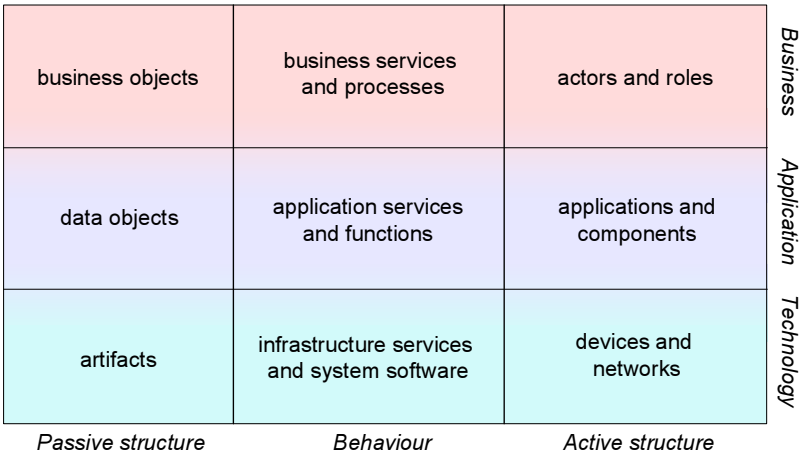

Fig. 7.6. Elements of an enterprise architecture.

For characterising the content of a view we define the following abstraction levels:

- **Details**: Views of the detailed level typically consider one layer and one aspect from the framework that was introduced in Chap. 5 (Fig. 7.6). Typical stakeholders are a software engineer responsible for the design and implementation of a software component or a process owner responsible for effective and efficient process execution. Examples of views are a BPMN process diagram and a UML class diagram.
- **Coherence**: At the coherence abstraction level, multiple layers or multiple aspects are spanned. Extending the view to more than one layer or aspect enables the stakeholder to focus on architecture relations like process–use–system (multiple layer) or application–uses–object (multiple aspect). Typical stakeholders are operational managers responsible for a collection of IT services or business processes.
- **Overview**: The overview abstraction level addresses both multiple layers and multiple aspects. Typically such overviews are addressed to enterprise architects and decision makers such as CEOs and CIOs.

In Fig. 7.7, the dimensions of purpose and abstraction level are visualised in a single picture, together with examples of stakeholders. Table 7.3 and Table 7.4 summarise the different purposes and abstraction levels.

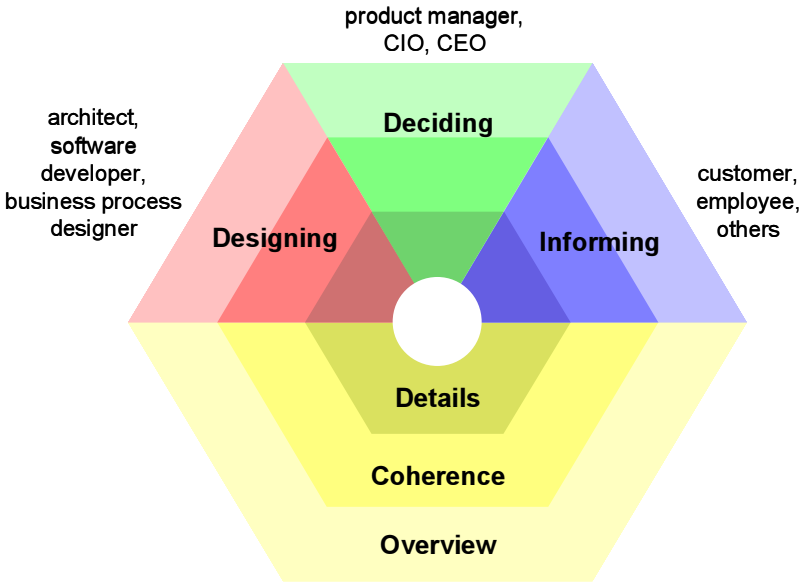

Fig. 7.7. Classification of enterprise architecture viewpoints.

Table 7.3. Viewpoint purpose.

	Typical stakeholders	*Purpose*	*Examples*
Designing	Architect, software developer, business process designer	Navigate, design, support design decisions, compare alternatives	UML diagram, BPMN diagram, flowchart, ER diagram
Deciding	Manager, CIO, CEO	Decision making	Cross-reference table, landscape map, list, report
Informing	Employee, customer, others	Explain, convince, obtain commitment	Animation, cartoon, process illustration, chart

Table 7.4. Viewpoint abstraction levels.

	Typical stakeholders	*Purpose*	*Examples*
Details	Software engineer, process owner	Design, manage	UML class diagram, Testbed process diagram
Coherence	Operational managers	Analyse dependencies, impact of-change	Views expressing relations like 'use', 'realise', and 'assign'
Overview	Enterprise architect, CIO, CEO	Change management	Landscape map

The landscape map viewpoint described in Sect. 7.2.1 is a typical example of a decision support view, which give a high-level overview and can, for example, be used to identify redundancies or gaps in the application landscape of an enterprise.

The process illustration viewpoint described in Sect. 7.2.1 is an example of a viewpoint intended for 'informing' others. It depicts workflows in a cartoon-like fashion, easily readable for employees and managers. Process illustrations can be on the detailed, coherence, or overview abstraction level.

To assist the architect in designing an enterprise architecture, we present a set of basic design viewpoints in the next sections. These viewpoints are all diagrams for designing architectures. Some viewpoints are multiple-aspect and multiple-layer overviews at the 'coherence' level of abstraction, while others are at the 'details' level.

7.4.2 Guidelines for Using Viewpoints

To help you in selecting and using viewpoints for tasks at hand, we present a number of guidelines, based on our own experience and interviews with architects from practice.

In general, the use of an architectural viewpoint will pass through a number of phases. These phases roughly are:

1. **Scoping**: Select one or more appropriate viewpoints, select the (sub-)domain that needs to be represented or modelled, and determine the constraints that apply to the domain being modelled.
2. **Creation of views**: Create or select the actual content of the viewpoint, i.e., create or select a view conforming to the viewpoint used. This can pertain to the selection of a part of the larger (pre-existing) architecture model, or the creation or refinement of a part of the architecture model (in terms of a view).
3. **Validation**: Validate the resulting view. Do the stakeholders agree that the view is a correct representation of the actual or intended situation?
4. **Obtaining commitment**: If *agreement* has been reached among the key stakeholders involved, the next step will be to create *commitment* for the results. In other words, do the stakeholders commit themselves to the (potential) impact of what is described by the view?
5. **Informing**: Inform other stakeholders of the results. These stakeholders will be those members of the development community, whose explicit commitment has, in a *conscious* decision, been considered not to be crucial.

Note that these phases will not necessarily be executed in a linear order. Practical circumstances usually dictate a more evolutionary approach. The viewpoints to be used for architectural communication will have to support the activities of each of the phases. The guidelines resulting from the interviews are divided over them. They are discussed in the next sections.

7.4.3 Scoping

The importance of focusing on the concerns of stakeholders, and the extent to which a specific view(point) addresses these concerns, was confirmed by the outcomes of the interviews. When you communicate with business managers, you only need those views or models that enable a discussion of factors deserving special attention. Typically these are factors that have a high impact if they fail and also have a high risk of indeed failing. For communication with the actual software developers, on the other hand, more detailed models are crucial.

The selection of viewpoints should be done consciously and based on rational considerations. Furthermore, architects state that this decision, and its rationalisation, must be readily available. It is quite possible that a stakeholder (usually a technology-oriented one) will ask for more detail in a model than you can give, or want to give, in that particular phase of the project. An architect should be prepared to clarify better the goals of the particular model and phase, and why the requested details are not yet relevant (or even harmful).

Determining the constraints that should guide the ensuing creation phase is also considered to be important. Numerous IT projects suffer from the problem that designers have too much 'design freedom' when producing a model of a desired future system. This increases the risk of ending up with lengthy design processes. Limiting design freedom by means of architecture principles, a higher-level architecture, or any other means, reduces this risk considerably.

7.4.4 Creation of Views

During the creation of a view, in particular when it involves actual modelling, you should try to put a limit on the number of participants in a conversation. Graphical models may or may not be used in communication with stakeholders, but most actual modelling is done by individuals (or two people at most). Genuine group modelling sessions are very rare.

During the early stages of system design, it is often considered bad to 'think' in terms of 'solutions'. However, when detailed modelling takes place in a cooperative setting, give informants some room to think in terms of 'solutions' even if pure requirements thinking (what, not how) does not officially allow for this. Most people just think better in terms of concrete solutions; it is a vital part of their creativity. Just be sure that requirements thinking is returned to in due course. In general, when you discuss models with stakeholders and informants, in particular when you try to establish a common understanding, you should discuss different scenarios and alternatives to the model being considered. Doing so leads to an exploration of the meaning and impact of the model taking shape, and also leads to improved mutual understanding.

The graphical notation that is part of a viewpoint should be approached flexibly when it comes to communicating with non-technical stakeholders. If people are not used to or prepared to deal with abstract graphical models, do not use them. Use other forms of visualisation, text, or tables. Iconised diagrams work particularly well. However, be prepared to point out

the relation between the alternative visualisation and your abstract models if asked to.

Even if graphical models play a big role in architecture, text is the chief form in which (written) communication takes place. Two main ways in which this occurs are:

- Graphical (partial!) models that are used to support textual descriptions ('illustration by diagram').
- Text explaining and elaborating on a graphical model ('textual modelling').

In fact, text is often better than a graphical model for conveying large amounts of detail.

Language studies have indeed shown how the specific form of a language does have an impact on what is expressed by means of the language (Cruse 2000). In the case of modelling languages, the modelling concepts offered by the language will, in general, influence the level of detail or abstraction that the resulting models will exhibit.

Finally, during a modelling session, several things may come to the fore that will influence the further process. External events may occur that are a threat to the process as a whole. Be prepared to stop modelling if executive commitment is withdrawn. It may be frustrating, but from a business perspective it may also be crucial. It is simply part of a flexible project setup. If the informants turn out to be less informed than expected, it is better to stop than to try and 'make the best of it' and produce an ill-conceived model.

In the field of agile development (Martin 2002, Rueping 2003, Ambler 2002), a refreshing perspective can be found on such considerations.

7.4.5 Validation

In validation of an architecture with stakeholders, a clear difference should be made between validation of content (qualitative validation, by modellers and experts) and validation in terms of commitment (by executives). Both are crucial, but very different. Obtaining (and validating) commitment is discussed in the next subsection.

Whether good mutual communication and understanding about a model is being reached is often a matter of intuition. If the people involved have a mutual feeling that 'their thoughts are well in sync', then dare to trust that feeling. However, if the opposite is the case, be prepared to invest in substantial discussion of concrete examples, or face the dire consequences of poor validation. If the required 'level of agreement' between participants is

high, an atmosphere of mutual trust and cooperation between these participants is crucial.

Validation is an activity that should be conducted in limited groups. 'Feedback rounds' involving a larger number of people, by e-mail or printed documentation, do not really work. If you want feedback that is worth something, find key people and discuss the models/views, preferably face to face. Make sure the 'opinion leaders' in an organisation agree to the model.

Also, you should take care that the languages used to express a view do not have a wrong connotation that may result in incorrect impressions about the scope and status of models. A language like UML cannot be used in a discussion with business people. Even though the language is suitable to express the models, the notation has an implementation-oriented connotation to this audience.

Furthermore, do not show a concrete view of the desired system too early on in the development process. The concreteness of the diagram may give the stakeholders a feeling that important decisions have already been made.

With regards to the last observation, an interesting statement on this issue can also be found in Weinberg (1988). He argues that when the design of a system, or a model in general, is still in its early stages, and different aspects are not yet clear and definite, the graphical notation used should also reflect this. He suggests using squiggly lines rather than firm lines, so as to communicate to the reader of a view that specific parts of the view are still open to debate. We use this principle in the Introductory viewpoint discussed in Sect. 7.5.1.

7.4.6 Obtaining Commitment

Obtaining commitment for a specific architectural design involves obtaining commitment for the *impact* of this design on the future system and its evolution, as well as the *costs/resources* needed to arrive at this future system. This means that the message that one needs to get across to the stakeholders involves:

- What are the major problems in the current situation?
- How bad are these problems (to the concerns and objectives of the stakeholders)?
- How will this improve in the new situation? (Benefits!)
- At what costs will these improvements come?

When discussing costs and benefits with stakeholders, make these costs and benefits as SMART (Specific, Measurable, Attainable, Realisable, and Time-bound) as possible. Make sure that the stakeholders agree, up front, with the criteria that are used to express/determine costs and benefits. It is their commitment that is needed. They will be the judge. Let them also decide what they want to base their judgement on! Create shared responsibility towards the outcomes.

Selecting the stakeholders that should be involved when obtaining commitment is also of key importance. Involving the wrong stakeholders, or leaving out important ones, will have obvious repercussions. At the same time, selecting a too large a group of stakeholders may bog down the process. Too much communication may be a bad thing: it may create unnoticed and uncontrolled discussion outside the main discussion, leading to twisted conceptualisations and expectations.

Though ideally 'everyone' should be heard, this is generally a practical impossibility. Therefore, choose your experts carefully. Aim for the opinion leaders, and also accept that you cannot please everyone. Realise that some people will not be perfectly satisfied, prepare for it, and deal with it.

People who actually make the decisions are usually those who are just outside the group of people who really know what is going on. Make sure that the former people are *also* involved and aware of what is happening.

Getting executive commitment may actually be dictated technologically. If their business is highly technological, business people do not see technology as secondary, and will only commit to something if they are assured that 'their organisation will be able to run it'.

Sharing design decisions and their underlying considerations at a late stage has a negative impact on the commitment of stakeholders. Start building commitment early on in the process. This implies that the linear ordering of the 'viewpoint use phases' as provided at the start of this section should not be applied strictly.

Once agreement has been reached, you should document this explicitly. Models are never accepted as sufficient statements to base agreements and commitment on. Commitments and agreements also need to be spelled out separately, in text.

7.4.7 Informing Stakeholders

Once commitment from the opinion leaders has been obtained, other stakeholders may be informed about the future plans and their impact. In doing so, it still makes sense to concentrate on cost/benefit considerations when trying to 'sell' the new system. Below, we have gathered some ob-

servations that apply to the informing phase. However, due to their general communicative nature, some of these observations are also applicable to the creation, validation, and commitment phases.

Do not impose presumed architectural terminology on true business people. Use *their* terminology. Even a concept like 'service' is suspect because it is relatively technology oriented and often unknown by stakeholders that are strictly on the business side.

Models are particularly important in giving stakeholders a feeling that they are 'part of the larger whole'. Often, just knowing where in the model 'they can be found' is important to stakeholders, even if they do not understand the fine points of the model.

Communication is the crucial factor in enterprise architecture. It will even pay off actually to employ some communication experts (think marketing, PR, even entertainment!) in larger projects. As a result, you will end up with stakeholders that are genuinely prepared to change the way they and their business work, not just with some interesting looking plans and models. Crucially, communication can be quite different for various stages of system development. Therefore, it is important to have a good communication strategy and a framework guiding you in this.

Even if people are willing to and able to read models thoroughly, text (spoken or written) needs to be added. Models alone never suffice.

7.5 Basic Design Viewpoints

The most basic type of viewpoint is the selection of a relevant subset of the ArchiMate concepts and the representation of that part of an architecture that is expressed in the concepts in this selection. This is sometimes called a 'diagram', akin to, for instance, the UML diagrams.

In Sect. 6.3.2, we introduced the following four metaphorical directions from which we can identify relevant model elements:

1. 'inwards', towards the internal composition of the element;
2. 'upwards', towards the elements that are supported by it;
3. 'downwards', towards its realisation by other elements;
4. 'sideways', towards peer elements with which it cooperates.

We also use these directions to identify possibly useful viewpoints.

For the 'composition' viewpoints, we start from the basic structure of our modelling language. In its elementary form, the generic meta-model that is behind the language consists of active structural elements such as actors, behavioural elements such as functions and processes, and passive informational elements such as business and data objects, which are pro-

essed by the active elements in the course of their behaviour (see also Fig. 7.6).

From this basic structure, we can deduce a first set of viewpoint types, containing three viewpoints that are centred around one specific type of concept:

1. active elements, e.g., the composition of a business actor from sub-actors, i.e., an organisation structure;
2. behaviour elements, e.g., the structure of a business process in terms of subprocesses;
3. passive elements, e.g., the information structure in terms of data objects.

Although these viewpoints take a specific type of concept and its structure as their focus, they are not limited to these concepts, and closely related concepts are also included.

For the 'upwards' support of elements in their environment, the active elements offer interfaces through which their services can be used. 'Downwards' services are realised by processes and functions, and application components are deployed on infrastructure elements. 'Sideways' cooperation is achieved through collaborations between active elements and their behaviour in the form of interactions, and flows of information and value that relate the elements. Passive elements often play a role in these relations, e.g., by being passed from one element to another, but are not the focus. Hence we concentrate on the relations between the active and behaviour elements.

Next to the design viewpoints resulting from these metaphorical directions, which focus on a limited part of an enterprise architecture, we also need to represent the whole architecture, but in a simplified form. Especially early in the design process, when we do not yet know all the details that are added later on, we want to express an architecture using a subset of the ArchiMate language denoted in an informal, simplified form. This helps to avoid the impression that the design is already fixed and immutable, which may easily arise from a more formal diagram. Furthermore, such a high-level overview is very useful in obtaining commitment from stakeholders at an early stage of the design (see also Sect. 7.4.6). To this end, we introduce the Simplified viewpoint.

In each of the viewpoint types, concepts from the three layers of business, application, and technology may be used. However, not every combination of these would give meaningful results; in some cases, for example, separate viewpoints for the different layers are advisable. Based on common architectural practice, our experiences with the use of ArchiMate models in practical cases, and on the diagrams used in other languages like UML, we have selected the most useful combinations in the form of a

'standard' set of basic viewpoints to be used with the ArchiMate concepts (Table 7.5).

Table 7.5. Design viewpoints.

Early design	*Cooperation*
Introductory, p. 173	Actor Cooperation, p. 175 Business Process Cooperation, p. 180 Application Cooperation, p. 182
Composition	*Realisation*
Organisation, p. 175 Business Function, p. 177 Business Process, p. 181 Information Structure, p. 182 Application Behaviour, p. 185 Application Structure, p. 186 Infrastructure, p. 186	Service Realisation, p. 179 Implementation & Deployment, p. 187
	Support
	Product, p. 178 Application Usage, p. 184 Infrastructure Usage, p. 187

Some of these viewpoints have a scope that is limited to a single layer or aspect: the Business Function and Business Process viewpoints show the two main perspectives on the business behaviour; the Organisation viewpoint depicts the structure of the enterprise in terms of its departments, roles, etc.; the Information Structure viewpoint describes the information and data used; the Application Structure, Behaviour, and Cooperation viewpoints contain the applications and components and their mutual relations; and the Infrastructure viewpoint shows the infrastructure and platforms underlying the enterprise's information systems in terms of networks, devices, and system software. Other viewpoints link multiple layers and/or aspects: the Actor Cooperation and Product viewpoints relate the enterprise to its environment; the Application Usage viewpoint relates applications to their use in, for example, business processes; and the Deployment viewpoint shows how applications are mapped onto the underlying infrastructure.

In the next subsections, we will explain these design viewpoints in more detail and provide examples of each one. In these examples, we have made extensive use of the abstraction rule that can be applied on chains of structural relations in ArchiMate, which was explained in Sect. 5.7. Note that it is explicitly *not* the intention to limit the user of the ArchiMate language to these viewpoints; neither do we expect an architect to draw all these diagrams in a given situation! They are meant to assist the modeller in choosing the contents of a view, but combinations or subsets of these viewpoints could well be useful in specific situations.

It is important in the examples that these views exhibit considerable overlap., e.g., in Fig. 7.13, which shows the high-level business functions of our ArchiSurance example, there is a business function Customer Relations. This reappears in Fig. 7.18, which shows how the Handle Claim business process is related to a number of business functions. Different aspects of this business process are shown, for example, in Fig. 7.19 (its use of information), Fig. 7.16 (realisation of services by business processes), and Fig. 7.17 (its relations with other business processes), and there are many more of these overlaps between views. This shows that underlying these different views there is a single model, and each view is a projection of the relevant elements in that model. We will use two examples throughout the description of the basic design viewpoints to illustrate this coherence:

- The handling of insurance claims;
- The policy administration systems and infrastructure.

7.5.1 Introductory Viewpoint

The Introductory viewpoint forms a subset of the full ArchiMate language using a simplified notation. It is typically used at the start of a design trajectory, when not everything needs to be detailed, or to explain the essence of an architecture model to non-architects who require a simpler notation. Another use of this basic, less formal viewpoint is that it tries to avoid the impression that the architectural design is already fixed, an impression that may easily arise when using a more formal, highly structured, or detailed visualisation.

We use a simplified notation for the concepts (Fig. 7.8), and for the relations. All relations except 'triggering' and 'realisation' are denoted by simple lines; 'realisation' has an arrow in the direction of the realised service; 'triggering' is also represented by an arrow. The concepts are denoted with slightly thicker lines and rounded corners, which give a less formal impression. The example in Fig. 7.9 illustrates this notation.

On purpose, the layout of this example is not as 'straight' as an ordinary architecture diagram; this serves to avoid the idea that the design is already fixed and immutable. This conforms to the suggestion made in Weinberg (1988) to use squiggly lines rather than firm lines, to show to the reader of a view that specific parts of the view are still open to debate.

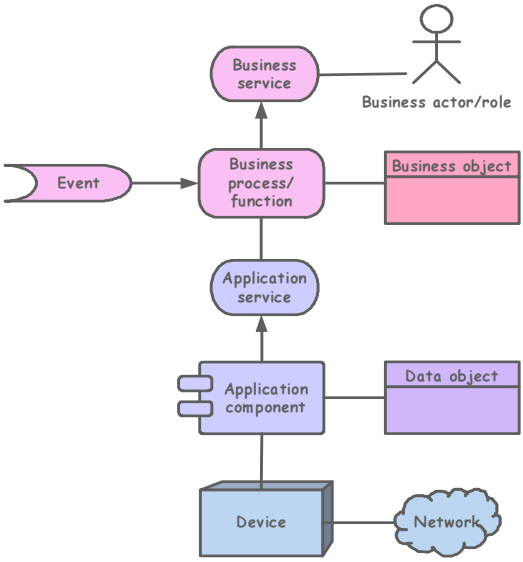

Fig. 7.8. Concepts and notation for the Introductory viewpoint.

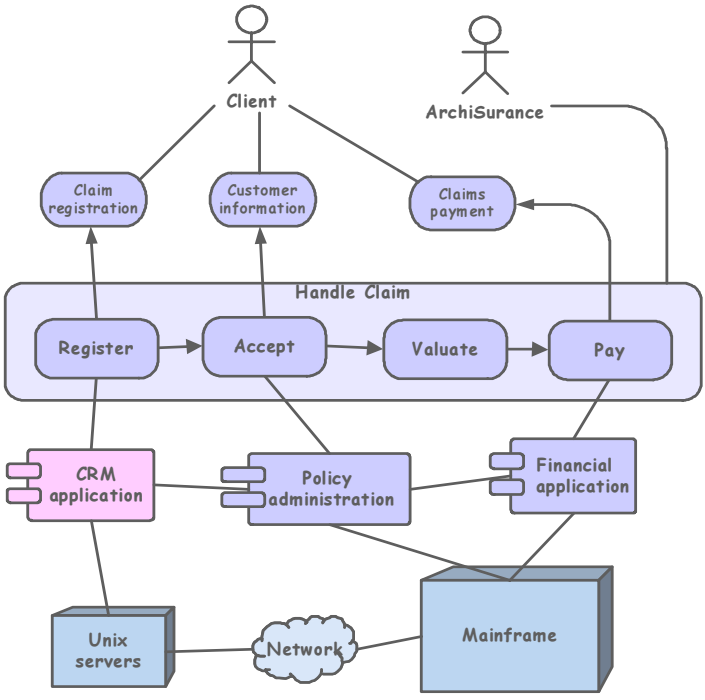

Fig. 7.9. Example of an Introductory view which conforms to the viewpoint of Fig. 7.8.

7.5.2 Organisation Viewpoint

The Organisation viewpoint shows the structure of an internal organisation of the enterprise, department, or other organisational entity. It can be represented in the form of a nested block diagram, but also in more traditional ways like the organigram. An Organisation view is typically used to identify authority, competencies, and responsibilities within an organisation.

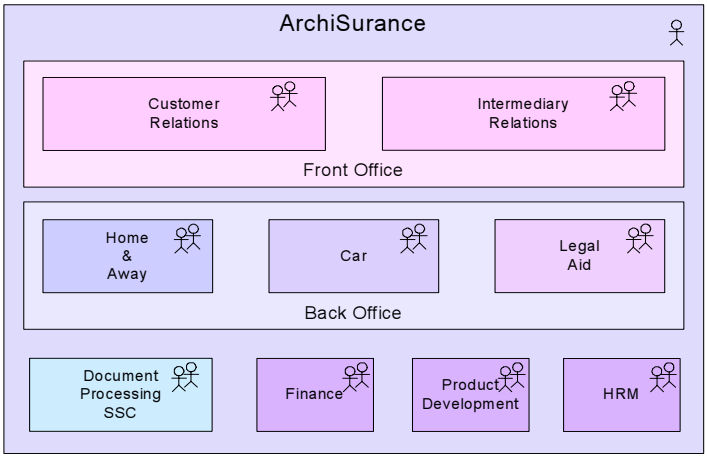

Fig. 7.10. ArchiSurance organisation structure.

In Fig. 7.10, we can see the high-level subdivision of ArchiSurance into a front and back office, and a finance department. Within the back office, there are three departments responsible for specific products, e.g., car, travel, or legal aid insurance, and the shared service centre for document processing. The front office comprises two departments that handle the relations with customers and intermediaries, respectively.

7.5.3 Actor Cooperation Viewpoint

The Actor Cooperation viewpoint focuses on the relations of actors with each other and their environment. A common example of this is what is sometimes called a 'context diagram', which puts an organisation into its environment, consisting of external parties such as customers, suppliers, and other business partners. It is useful in determining external dependencies and collaborations and shows the value chain or network in which the organisation operates. Another important use of this viewpoint is in showing how a number of cooperating (business and/or application) actors together realise a business process, by showing the flows between them.

The main roles involved in the insurance business are the customer, the insurer, the intermediary, and the customer's bank. These cooperate in different settings. For example, closing an insurance contract involves the customer, insurer, and intermediary, whereas premium collection involves the insurer, the customer and the customer's bank. The main collaborations of ArchiSurance, which fulfils the role of the insurer, are shown in Fig. 7.11.

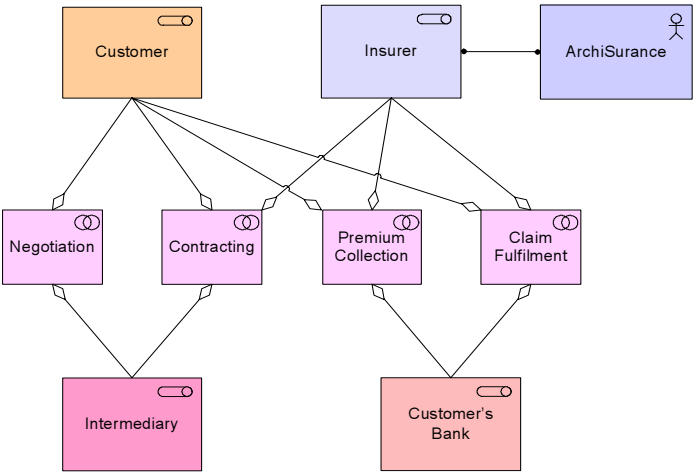

Fig. 7.11. Collaborations of ArchiSurance and its partners.

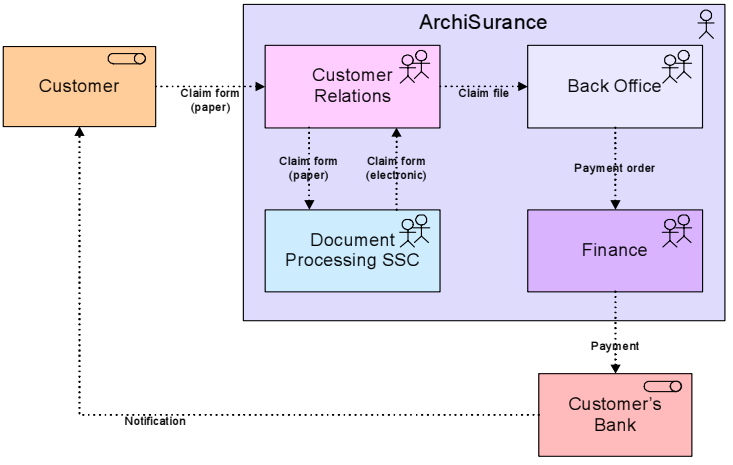

Fig. 7.12. Information flows between ArchiSurance's departments and partners in handling insurance claims.

If we look more closely at the relations between actors and roles, it is useful to focus on the information flows between them to identify, for example, important dependencies. In Fig. 7.12, we see the information flows that are associated with the Handle Claim business process that is used as an example throughout the description of these viewpoints. The types of business objects passed between the actors are put as annotations to the flow arrows; these correspond to the business objects used by the Handle Claim process shown in Fig. 7.19. If needed, we could also include the interfaces used in exchanging this information, e.g., telephone or e-mail.

7.5.4 Business Function Viewpoint

The Business Function viewpoint shows the main business functions of an organisation and their relations in terms of the flows of information, value, or goods between them. Business functions are used to represent what is most stable about a company in terms of the primary activities it performs, regardless of organisational changes or technological developments. Business function architectures of companies that operate in the same market therefore often exhibit many similarities. The Business Function viewpoint thus provides high-level insight into the general operations of the company, and can be used to identify necessary competencies, or to structure an organisation according to its main activities.

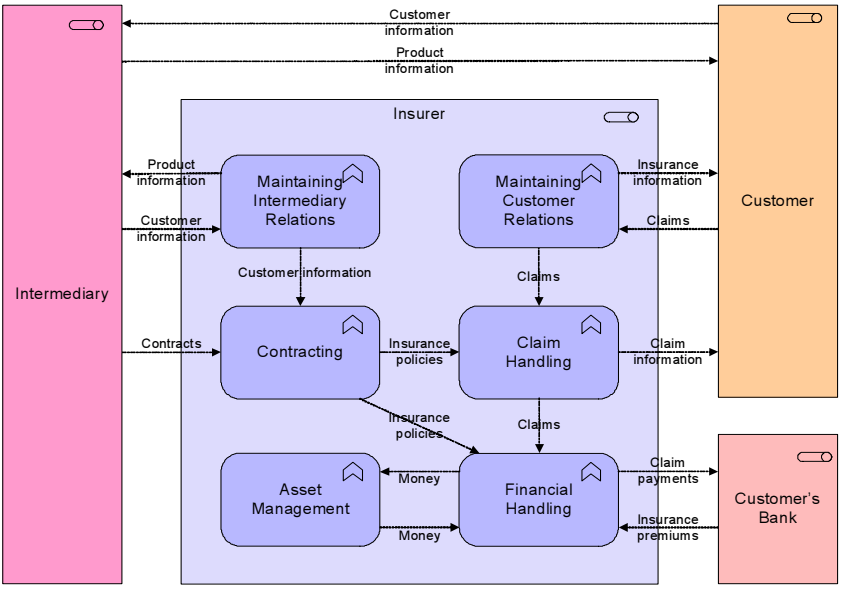

Fig. 7.13. Business functions and flows of information and money.

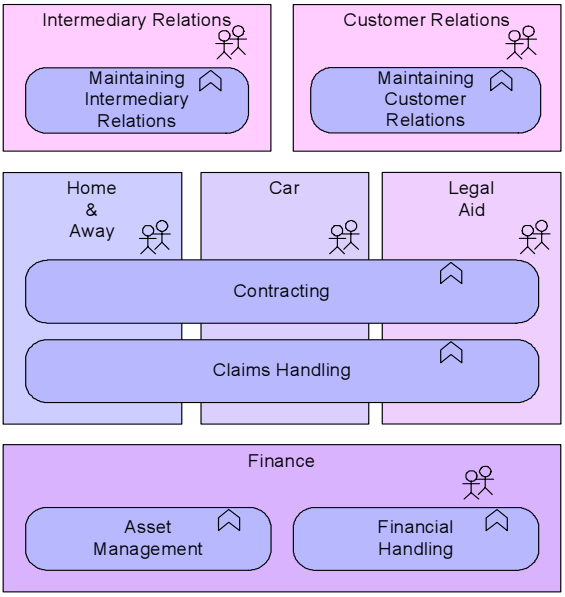

Fig. 7.14. Business functions and organisation structure.

In the example of Fig. 7.13, we can see the information flow associated with the handling of insurance claims. Claims are submitted to the Maintaining Customer Relations business function, processed by Claim Handling, and paid by Financial Handling. In the Business Process viewpoint (Sect. 7.5.5), we will see a more detailed depiction of this process. In Fig. 7.14, these business functions are mapped onto the responsible organisational units that were shown in Fig. 7.10.

7.5.5 Product Viewpoint

The Product viewpoint depicts the value this product offers to the customers or other external parties involved and shows the composition of one or more products in terms of the constituting (business or application) services, and the associated contract(s) or other agreements. It may also be used to show the interfaces (channels) through which this product is offered, and the events associated with the product.

A Product view is typically used in designing a product by composing existing services or by identifying which new services have to be created for this product, given the value a customer expects from it. It may then serve as input for business process architects and others that need to design the processes and IT systems that realise this product.

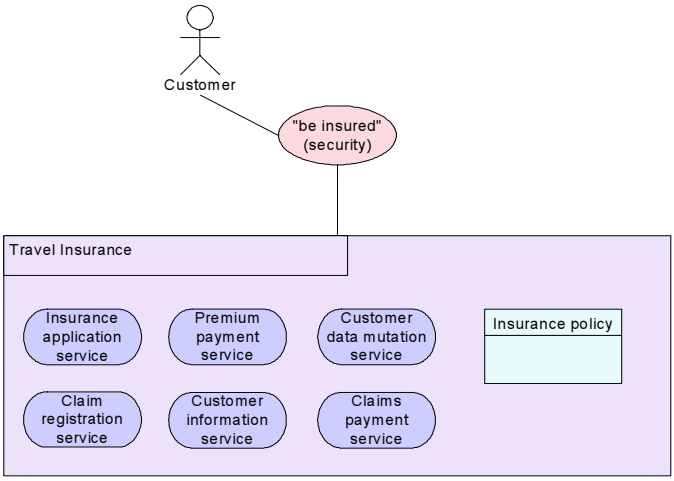

Fig. 7.15. The travel insurance product.

A typical insurance product of ArchiSurance is depicted in Fig. 7.15. The value to the customer of an insurance is typically the added security it provides. The services mentioned here are realised by various business processes, an example of which is given in Sect. 7.5.6.

7.5.6 Service Realisation Viewpoint

The Service Realisation viewpoint is used to show how one or more business services are realised by the underlying processes (and sometimes by application components). Thus, it forms the bridge between the Product viewpoint and the Business Process viewpoint. It provides a 'view from the outside' of one or more business processes.

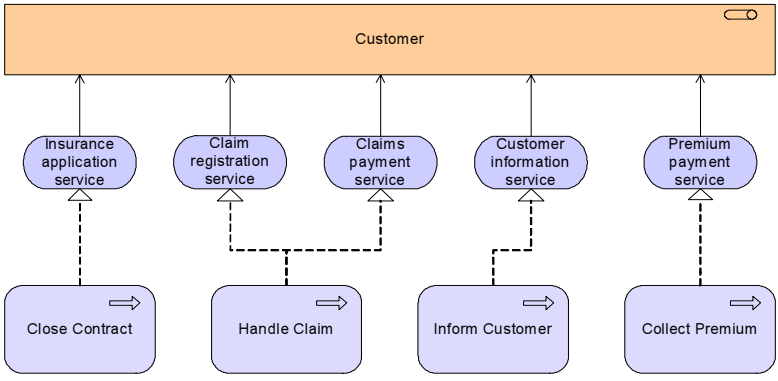

Fig. 7.16. Realisation of business services by ArchiSurance business processes.

Business services are realised by business processes. In Fig. 7.15, we saw the services that constitute the travel insurance product. The business processes that realise these services are shown in Fig. 7.16. For example, the Claim registration service is realised by the Handle Claim business process that we use as an example throughout this chapter.

7.5.7 Business Process Cooperation Viewpoint

The Business Process Cooperation viewpoint is used to show the relations of one or more business processes with each other and/or their surroundings. It can be used both to create a high-level design of business processes within their context and to provide an operational manager responsible for one or more such processes with insight into their dependencies. Important aspects of coordination are:

- causal relations between the main business processes of the enterprise;
- the mapping of business processes onto business functions;
- realisation of services by business processes;
- the use of shared data;
- the execution of a business process by the same roles or actors.

Each of these can be regarded as a 'sub-viewpoint' of the Business Process Cooperation viewpoint. Below, we give examples of some of the resulting views.

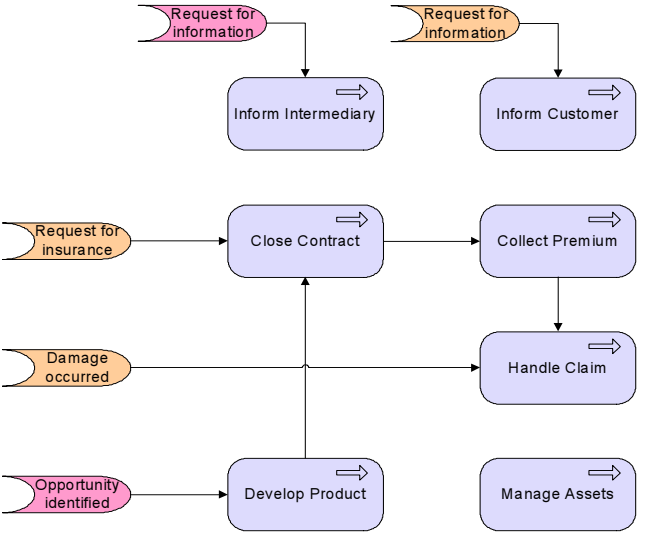

Fig. 7.17. Some of the main business processes, triggers, and relations of ArchiSurance.

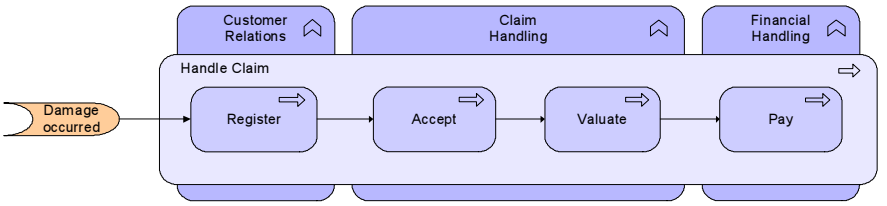

Fig. 7.18. The Handle Claim business process mapped onto the business functions.

In Fig. 7.17, the most important business processes of ArchiSurance are depicted. It also shows their relations, e.g., the Collect Premium process needs to be preceded by the Close Contract process, since of course no premium can be collected before the insurance policy has been issued. This figure also shows the Handle Claim process that occurs in many of the other viewpoints. In Fig. 7.18, the Handle Claim business process of Fig. 7.19 is mapped onto the business functions of Fig. 7.13.

7.5.8 Business Process Viewpoint

The Business Process viewpoint is used to show the high-level structure and composition of one or more business processes. Next to the processes themselves, this viewpoint contains other directly related concepts such as:

- the services a business process offers to the outside world, showing how a process contributes to the realisation of the company's products;
- the assignment of business processes to roles, which gives insight into the responsibilities of the associated actors;
- the information used by the business process.

Each of these can be regarded as a 'sub-viewpoint' of the Business Process viewpoint.

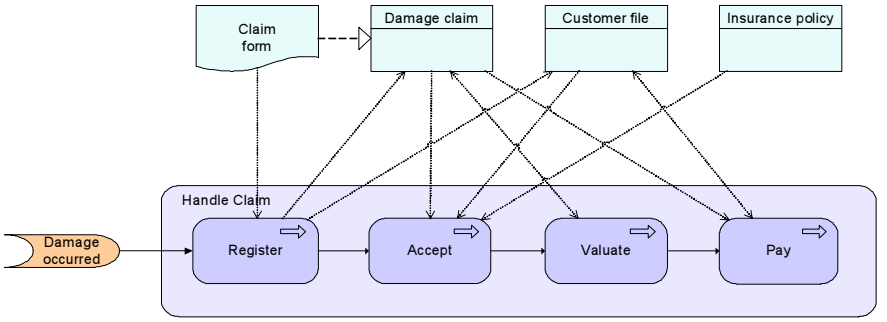

Fig. 7.19. The Handle Claim business process and its use of information.

In Fig. 7.19, the Handle Claim business process is shown, together with the information it uses. This shows in more detail which subprocesses are carried out in handling insurance claims.

7.5.9 Information Structure Viewpoint

The Information Structure viewpoint is basically identical to the traditional information models created in the development of almost any information system. It shows the structure of the information used in the enterprise or in a specific business process or application, in terms of data types or (object-oriented) class structures. Furthermore, it may show how the information at the business level is represented at the application level in the form of the data structures used there, and how these are then mapped onto the underlying infrastructure, e.g., by means of a database schema.

In Fig. 7.20, the most important business objects of ArchiSurance are shown. Some of these are used in the Handle Claim business process, as depicted in Fig. 7.19.

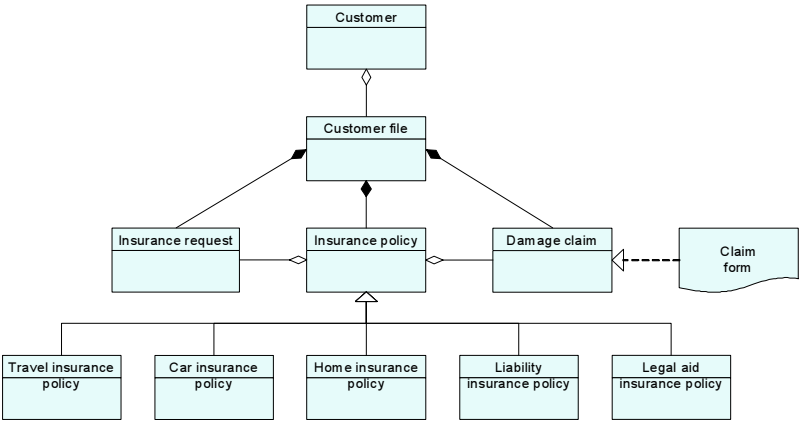

Fig. 7.20. Information model of ArchiSurance.

7.5.10 Application Cooperation Viewpoint

The Application Cooperation viewpoint shows the relations of a number of applications or components. It describes the dependencies in terms of the information flows between them, or the services they offer and use. This viewpoint is typically used to create an overview of the application landscape of an organisation.

This viewpoint is also used to express the coordination or orchestration (i.e., internal coordination) of services that together support the execution of a business process. By modelling the interdependencies between services, the coordination of the underlying applications is established in a more independent way. If this coordination is centralised and internal to the enterprise, we speak of 'orchestration'; in the case of coordination between independent entities, the term 'choreography' is often used.

The front- and back-office applications of ArchiSurance are shown in Fig. 7.21. It is clear that the back office is structured according to the different types of products, whereas the front office is already more integrated. One of the applications shown is the Home & Away policy administration used in several other viewpoints as well.

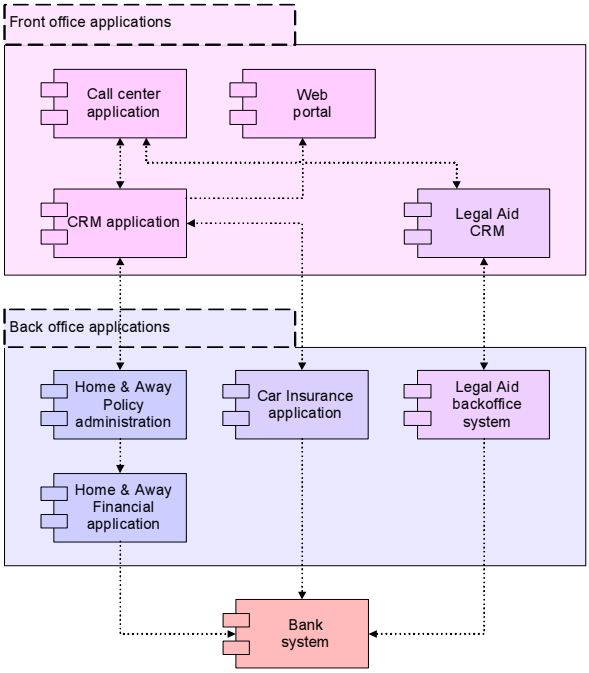

Fig. 7.21. Applications and information flow of ArchiSurance.

Some of the connections between the ArchiSurance applications are shown in Fig. 7.22, which shows that ArchiSurance uses the Enterprise Service Bus concept to link its applications. In Fig. 7.23, we see in more detail how the Claim information service from the Home & Away Policy administration is used by the department's Financial application, through an interface in which the message queuing service from the lower-level infrastructure is used (see also Fig. 7.28).

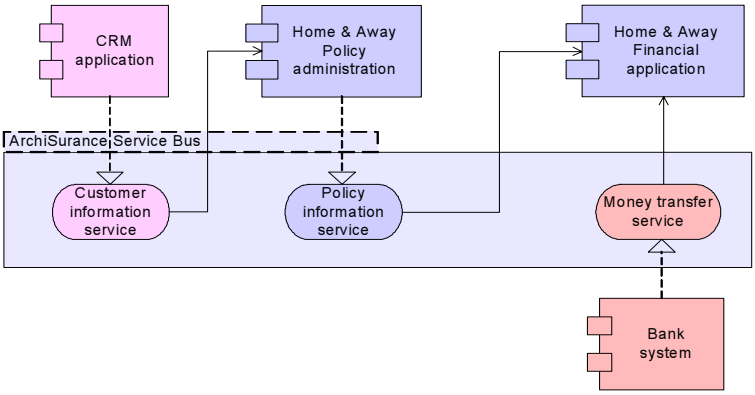

Fig. 7.22. Applications connected through the ArchiSurance Service Bus.

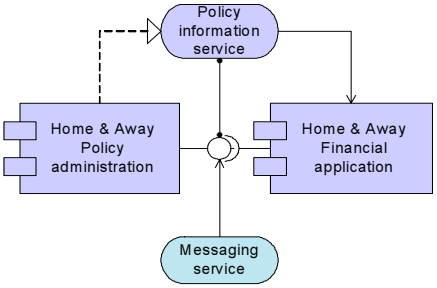

Fig. 7.23. Details of the connection between the Home & Away Policy administration and Financial application.

7.5.11 Application Usage Viewpoint

The Application Usage viewpoint describes how applications are used to support one or more business processes, and how they are used by other applications. It can be used in designing an application by identifying the services needed by business processes and other applications, or in designing business processes by describing the services that are available. Furthermore, since it identifies the dependencies of business processes upon applications, it may be useful to operational managers responsible for these processes.

In Fig. 7.24 it is shown how the Handle Claim business process uses the application services offered by several applications. Each of these services is realised by the behaviour of an application, an example of which is given in Fig. 7.25.

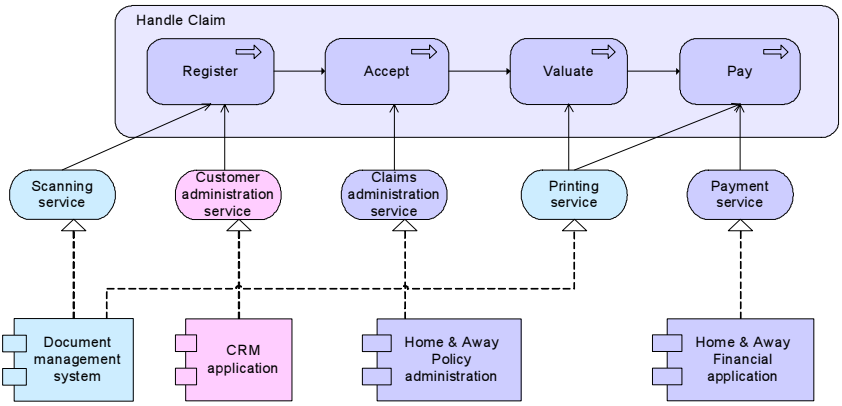

Fig. 7.24. Application usage by the Handle Claim business process.

7.5.12 Application Behaviour Viewpoint

The Application Behaviour viewpoint describes the internal behaviour of an application or component, for example, as it realises one or more application services. This viewpoint is useful in designing the main behaviour of applications or components, or in identifying functional overlap between different applications.

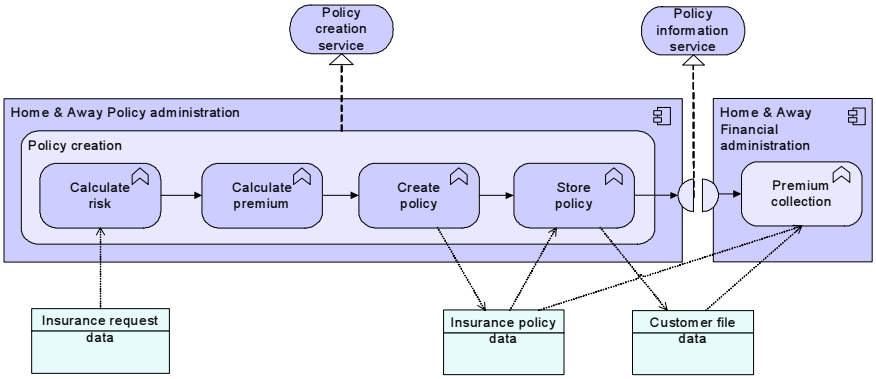

Fig. 7.25. Behaviour of the Home & Away Policy administration in realising the Policy creation service.

Part of the behaviour of the Home & Away Policy administration is shown in Fig. 7.25. The individual application functions are chained together and collectively realise the Policy creation application service. The

communication with the Financial administration takes place in an interaction that realises the Policy information service.

7.5.13 Application Structure Viewpoint

The Application Structure viewpoint shows the structure of one or more applications or components. This viewpoint is useful in designing or understanding the main structure of applications or components and the associated data, e.g., to create a first-step work breakdown structure for building a system, or in identifying legacy parts suitable for migration.

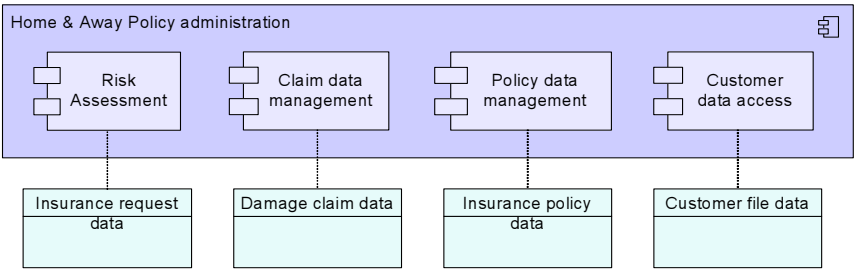

Fig. 7.26. Main structure of the Home & Away Policy administration.

Fig. 7.26 shows the main components that constitute the policy administration of ArchiSurance's Home & Away department. It also depicts some of the important data objects used by these components. These data objects are realisations of the business objects of Fig. 7.20.

7.5.14 Infrastructure Viewpoint

The Infrastructure viewpoint comprises the hardware and software infrastructure upon which the application layer depends. It contains physical devices and networks, and supporting system software such as operating systems, databases, and middleware.

The physical infrastructure of ArchiSurance and its intermediaries is shown in Fig. 7.27.

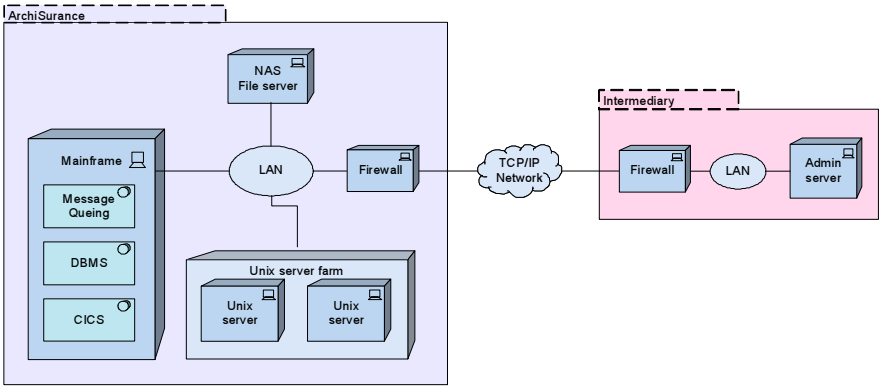

Fig. 7.27. Infrastructure of ArchiSurance.

7.5.15 Infrastructure Usage Viewpoint

The Infrastructure Usage viewpoint shows how applications are supported by the software and hardware infrastructure: infrastructure services delivered by the devices, system software, and networks are provided to the applications.

An example of this viewpoint is given in Fig. 7.28, which shows the use, by a number of back-office applications, of the messaging and data access services offered by ArchiSurance's infrastructure.

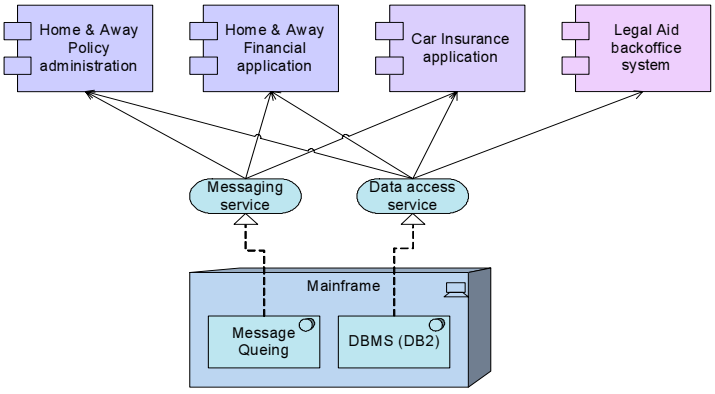

Fig. 7.28. Use of infrastructure services by ArchiSurance's back-office applications.

This viewpoint plays an important role in the analysis of performance and scalability, since it relate the physical infrastructure to the logical

world of applications. It is very useful in determining the performance and quality requirements of the infrastructure based on the demands of the various applications that use it. In Chap. 8, we will describe a quantitative analysis technique that can be used to determine, for example, the load on the infrastructure, based on its use by applications (and their use by business processes).

7.5.16 Implementation & Deployment Viewpoint

The Implementation & Deployment viewpoint shows how one or more applications are deployed on the infrastructure. This comprises the mapping of (logical) applications and components onto (physical) artifacts like, for instance, Enterprise Java Beans, and the mapping of the information used by these applications and components onto the underlying storage infrastructure, e.g., database tables or other files. In security and risk analysis, Deployment views are used to identify critical dependencies and risks.
Fig. 7.29 shows the mapping of logical application components of the Home & Away Policy administration (see Fig. 7.26) used in several of the other examples onto physical artifacts such as Enterprise Java Beans.

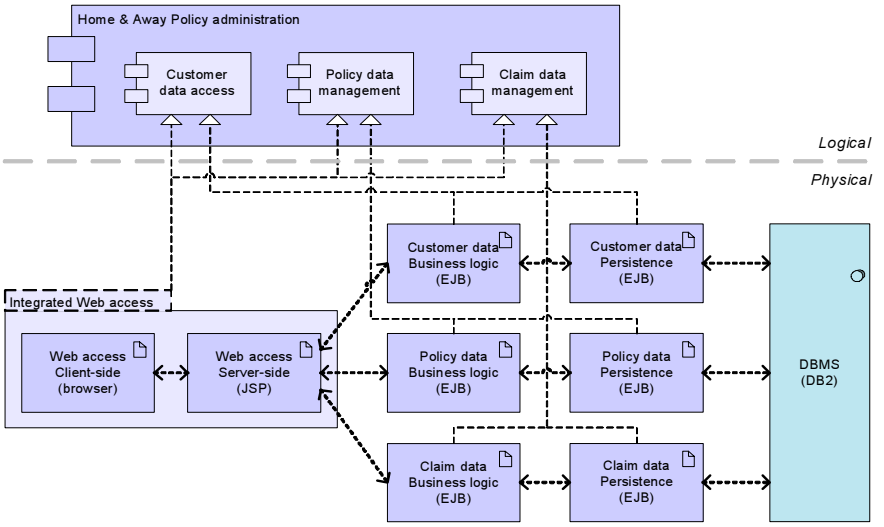

Fig. 7.29. Implementation of the Home & Away Policy administration.

7.6 Summary

In the previous sections, we have advocated a viewpoint-oriented approach to enterprise architecture modelling, in which architects and other stakeholders can define their own views of the architecture. In this approach views are specified by viewpoints, which define abstractions on the set of models representing the enterprise architecture, each aimed at a particular type of stakeholder and addressing a particular set of concerns.

We have described the use of viewpoints in communication, and the distinction between an architecture model, a view of that model, and its visualisation and manipulation. We have presented guidelines for the selection and use of viewpoints, and outlined a number of viewpoints in the ArchiMate language that can be used by architects involved in the creation or change of enterprise architecture models.