MASTER'S THESIS INFORMATION SCIENCE

BY

ROGER USMANY

CONCEPTUAL MODELING FOR BUSINESS PROCESS SEMANTICS

RADBOUD UNIVERSITY NIJMEGEN
FACULTY OF SCIENCE
INSTITUTE OF COMPUTING AND INFORMATION SCIENCES

July 13th, 2012

SUPERVISOR:
PROF. DR. IR. T.P. (THEO) VAN DER WEIDE, RADBOUD UNIVERSITY
REFERENT:
DR. P. (PATRICK) VAN BOMMEL, RADBOUD UNIVERSITY

THESIS NUMBER: 125 IK

Radboud Universiteit Nijmegen

Abstract

Nowadays in an ever changing world, enterprises transform themselves in order to increase their flexibility, effectiveness and efficiency. Enterprise transformation is a model-intensive activity. Such transformation involves models in different roles with regard to the value chain of modeling activities. When taking the return on modeling effort (ROME) perspective, enterprises are able to achieve their expected returns. As there exist various different modeling languages, they focus on a specific architectural domain with their own characteristics concepts for expressing it. One such language that integrates these architectural domains is ArchiMate that emphasizes the enterprise architecture. In contrast to Business Process Modeling Notation (BPMN) that is specifically designed and used for process modeling. The master thesis examines the architecture underlying these two modeling languages in which the business process aspect is translated into Petri Nets for analyzing the semantic of processes.

Acknowledgements

I want to thank my supervisor Prof. Dr. Ir. T.P van der Weide for supporting me during my research. He provided me with feedback to govern and to assess the progress of my thesis research. At times when I was struggling to find my way in the research, he gave me back incentives which I appreciate enormously. The distant supervision with timely discussions gave me also the freedom to explore new and creative ideas, which made my period as a student very pleasant. These all were supported by intensive discussions about IT related subjects, which eventually results in the architecture of modeling languages and their semantics.

Roger Usmany
July 13th, 2012

Contents

Chapter 1

Introduction

Many companies are using various domain architectures such as organization, products, business process, application, information, and technical architectures. In each of these architectural domains, specific concepts are defined, which model and visualize their internal coherence. These specific models and visualizations simplify communication, discussion and analysis within the domain.

In transformation processes a lot of models are produced in several stages, while each individual model deals with the same domain. It takes a lot of time and money to produce models during one stage of the transformation processes. Increasing coherence between models would, for example, enable the re-use of investments made in models earlier on in a transformation processes.

It is often the case that models have to be re-drawn or even re-modelled from one stage of the transformation process, such as an ArchiMate model, to some other languages at a later stage of the transformation process e.g. BPMN. This leads to unfavourable situations such as enormous costs and delays. An alternative to this problem is to create a coherent modeling landscape, which underlines the integration of modeling concepts at different levels. This prevents unnecessary delays and costs during transformation processes.

Two aspects are considered valuable when mapping from an ArchiMate model to a BPMN model:

1. *Integration* of both the ArchiMate and the BPMN metamodel which make such transformations much easier. A BPMN model provides a more detailed view on business processes, whereas an ArchiMate model provides a global view on the enterprise's activities of the enterprise architecture. It might be useful to consider the BPMN metamodel as a specialisation of (relevant parts of) the ArchiMate model.

2. *Standardising* the needed transformations between, for example, an ArchiMate model towards/backwards a BPMN model. This can become a part of the body of standards and increases the *portability* of these transformations between different modeling tools.

Unfortunately, creating one integrated modeling language would not be effective at all, due to the fact that at different stages of the enterprise's transformation different sets of modeling concepts are needed. Therefore, it is much better to use the more specific modeling languages with their own characteristic features. ArchiMate can be used [Jonk 11] to elaborate the enterprise architecture towards IT support for the enterprise's activities, while BPMN can be used to refine things even further to the level of specific applications and business processes. In spite of this, it is possible to have coherence between these different models used by distinctive modeling languages.

As a consequence, it is unclear how concepts used in various modeling languages are interrelated. It is quite difficult to interrelate the different architectural domains (see **Fig. 1.1**), although there exists some interdependencies [Tuli 09]. Moreover, it is unclear whether the views are compatible with each other. This means that the relationship between the concepts in these different architectural domains is in many cases unclear [Land 09]. As a matter of fact, these domains often partially overlap, but use different notions to express the same ideas. In some cases people who are involved in this, do not even know resulting in ambiguity. This might have consequences for the flexibly and efficiently operating organizations.

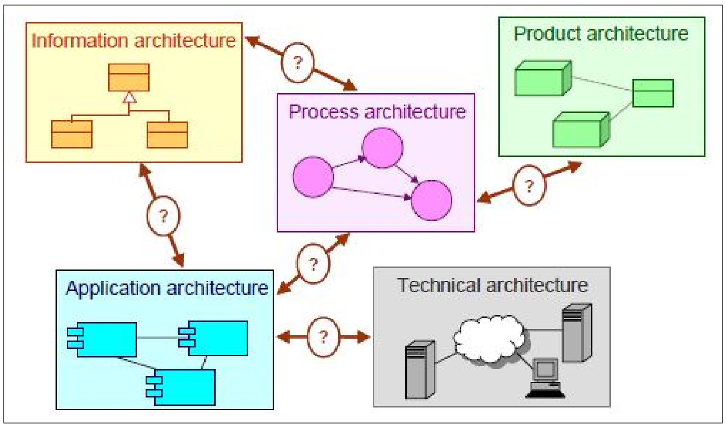


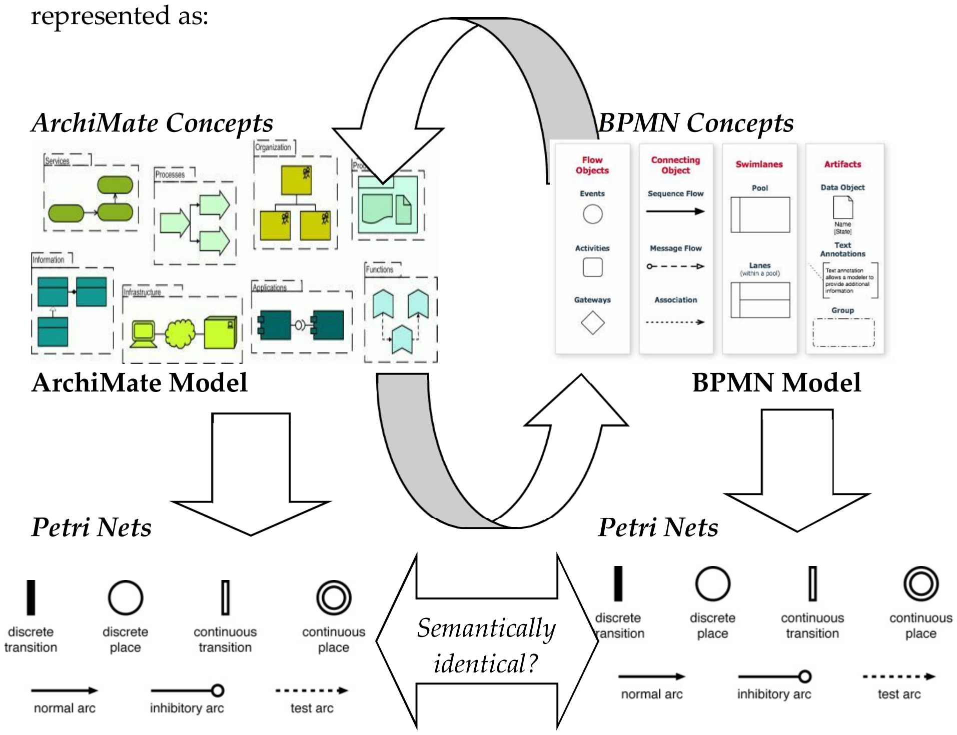
Fig. 1.1. Heterogeneity of architectural domains.

In order to see the coherence of metamodels of the distinctive modeling languages as well as the produced models, these models need to be interrelated [Lind 11]. This can be achieved by applying a disciplined naming convention for the concepts used in modeling languages. A way to realise this is to use persistent naming of concepts (e.g. actors, processes, functions) across the different models. This requires a relationship by matching the concepts according to the metamodels. In addition to this, the use of a domain model of different domain concepts, and consequent use of the concepts, could provide advantages to modellers to create more specific models which arise from the fact that they can start from a thorough understanding of the domain.

1.1 Research Questions

To cope with the previous mentioned issues, this master thesis examines the underlying architecture of the language ArchiMate and Business Process Modeling Notation (BPMN). These modeling languages have their own characteristic concepts for describing their architectural domain(s) [Hopp 05].

The following concept modeling approach can be simply graphically represented as:



When models need to be transformed or translated into similar models in a different modeling language at some stage, the following research question arises, with respect to *business* domains, and is formulated as:

> ***'To what extent do the semantics of business process models, arising from the ArchiMate language, correspond to similar business process models originating from Business Process Modeling Notation (BPMN), using their own typical framework underlying its architecture?'***

The research question can be divided into two subquestions, leading to the following questions:

- ✓ 'To what extent does the architecture of the ArchiMate modeling language and the Business Process Modeling Notation (BPMN) language relate to each other with respect to its internal structure and underlying principles?'

- ✓ 'To what extent does it seem to be possible to perform a mapping of architectural concepts among ArchiMate and BPMN mutually, with respect to semantics of business concepts?'

1.2 Relevance

It is often the case that model transformations, i.e. mapping architectural domain concepts to a similar model with preservation of the semantic model, cannot be completed, as relating architectures [Odeh 03] have to deal with implicitness.

In line with this, it means that each of the architectural domains is developed by distinct stakeholders with their own concerns. Therefore it is preferable to have views which are in some sense *consistent* with each other [Land 09]. An architectural language is not only needed for the description of integrated architectures, but also as a prerequisite for linking the different tools used in the various architectural domains [Lank 09b].

On the other hand, the more concepts are used in a modeling language, the more ways a situation can be expressed [Prop 05]. As ArchiMate is designed to be simple in learning and use, it has been limited to the concepts that suffice in the most modeling practical cases.

1.3 Research Approach

In order to gain more insight and to provide more in depth-knowledge, this master thesis will concentrate on how to deal with the complexity of architectures with respect to concept mapping of enterprise architecture concepts (ArchiMate) and project level (i.e. detailed specific) concepts (BPMN).

The hierarchy started from a set of relatively generic concepts (higher up in the pyramid). These were then specialized towards application at different architectural layers, as explained below.

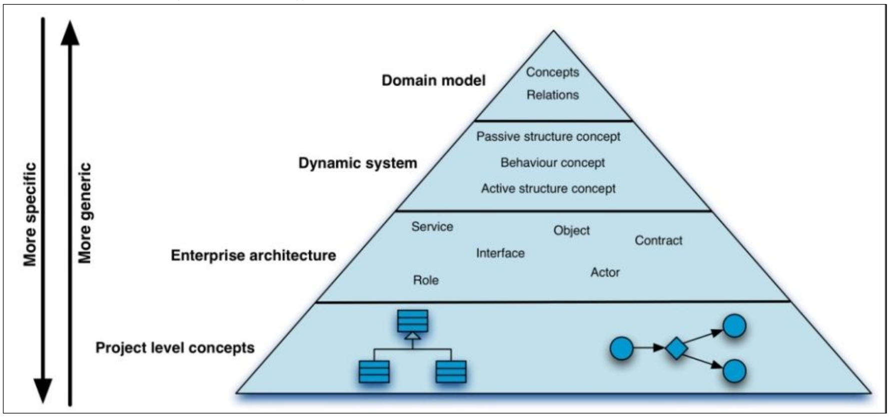


Fig. 1.3. Hierarchy of concepts at different levels of specialisation.

The concept hierarchy (see **Fig. 1.3**) describes at each level the core concepts, from generic concepts at the top to specific concepts at the bottom of the triangle. Concepts at the top of the triangle comprise 'concepts' and 'relations' in the domain modeling. Then the more specific concepts can be found at the level of the dynamic systems.

At the second layer, dynamic systems [Tuli 09] encompasses the 'passive', 'behaviour' and 'active' concepts, which inherit the characteristics of the layer above. Further down below, enterprise architecture concepts are expressed in terms of 'services', 'roles', 'interfaces', 'objects', 'actors' and 'contracts'.

At the base of the triangle the metamodels of the project level modeling concepts are used by specific organizations such as BPMN. A variety of existing modeling languages and standards can be found here as well.
The ArchiMate concept is found between the two extremes, the 'project level' layer and the 'domain model', namely the 'dynamic system' and 'enterprise architecture'. ArchiMate is designed in such a way that it is easy to use and to learn.

Arising from the relationship between enterprise architecture and project level concepts, the master thesis focusses on the semantics of concepts with respect to business processes that cover both modeling languages.

Underlying architecture of ArchiMate, BPMN and Petri Net

In order to provide a thorough understanding of the architectures of the ArchiMate, BPMN and Petri Net language, it is required to look at their metamodel and their internal structure. This runs parallel with the relevant concepts in Business Process Modeling Notation (BPMN). There are several different modeling languages ranging from architecture models to specific designs. To start, BPMN can serve as an example.
As ORM [Halp 96], [Halp 98],[Over 07],[Tuli 09] can be used for modeling, as it provides a comprehensive view of the domain. It is also well suited for the representation of metamodel due to precise modeling and elaborated verbalisations.

Mapping business concepts directly and indirectly to Petri Net

The mapping from an ArchiMate/BPMN concept to Petri Nets requires a well-formed definition that describes the semantics of concepts. Therefore, in such situations transformation [Soar 08] might be suitable and needed to establish a connection among ArchiMate and BPMN business concepts. The mapping of concepts can be done at both sides, namely from ArchiMate to BPMN and conversely from a BPMN concept towards an ArchiMate concept, in order to map these concepts properly and mutually to Petri Nets. The connectedness will not always be strictly done in terms of transformations, as sometimes the bridge is a bit loose. This bridging can be expressed in terms of textual/graphical expressions. Supplementary a glossary is listed, which describes the semantics of the core concepts used within the architecture of the modeling languages.

1.4 Related Work

In the past years there have been several researches in model transformations and providing formal semantics [Dijk 08] for specific architectural domains. Enterprise architecture provides concepts and techniques to support enterprise architects in the visualization, communication and analysis of integrated architectures [John 07]. Several researches have already shown the importance of models which have a strong connection to enterprise transformations.

Information System Engineering

As many enterprises want to aim for new challenges, they are enforced due to future changes to develop enterprise systems which are flexible and integratable i.e. system integration, in such a way to create a coherent landscape of enterprise systems. Therefore a method with a Meta Model integration technology has been introduced [Wang 05] to integrate several different enterprise systems such as Electronic commerce (EC) and enterprise resource planning (ERP).

Enterprise Engineering

Enterprises are constantly changing due to the dynamic nature of the environment in which they are operating. Enterprise (re-)engineering is brought to understand and optimize the enterprise operations. [Kosa 07] discusses enterprise engineering as an enterprise life-cycle oriented discipline for identification, design, and implementation of enterprises and their continuous evolution. Enterprise modeling will play an important role in creating the knowledge base and in using it for enterprise integration and operational decision support.

Model transformations

Transformations of models [Wier 04],[Soar 08],are in some cases essential to provide insight by deriving views from models. It is often the case that a model has to be redrawn or transformed to a similar model conforming to a given metamodel. Models mostly are not kept up-to-date which is an enormous waste of engineering effort. Therefore all relevant stakeholders (technical, business, operational) need to be involved in modeling tasks to take advantage of the modeling efforts.

Model Driven Architecture (MDA)

Model driven architecture is a new way to develop applications and writing applications. Due to the increasing complexity of enterprise computing systems, a model driven Web service development framework is presented to combat challenges in system development, integration, and maintenance [Yu 07].

Modeling linguistically

There exists a lot of variety in modeling techniques such as ORM, UML, BPMN, DEMO E3Value etc. Each language has their characteristic concepts, ontologies, terminologies that are related to a typical metamodel for describing the architecture of different domains e.g. business processes, applications or technical infrastructures. The paper [Hopp 04] stated that some specific languages, for instance ORM, are well suited for modeling complex business domains. Domain modeling is intended to support consensus and to achieve conceptual clarity among stakeholders involved in software development projects.

Chapter 2

Literature study

With literature study one can obtain the knowledge to deal with the comparison of two various modeling techniques. This chapter describes the relationship between the architecture of the ArchiMate and the BPMN language, including the key design principals and properties of these languages. Furthermore existing modeling methods are discussed to clarify the comparison of these two modeling languages.

2.1 Relating ArchiMate and BPMN

To understand the relationship between ArchiMate and BPMN, we first clarify the key designs of ArchiMate and BPMN separately.

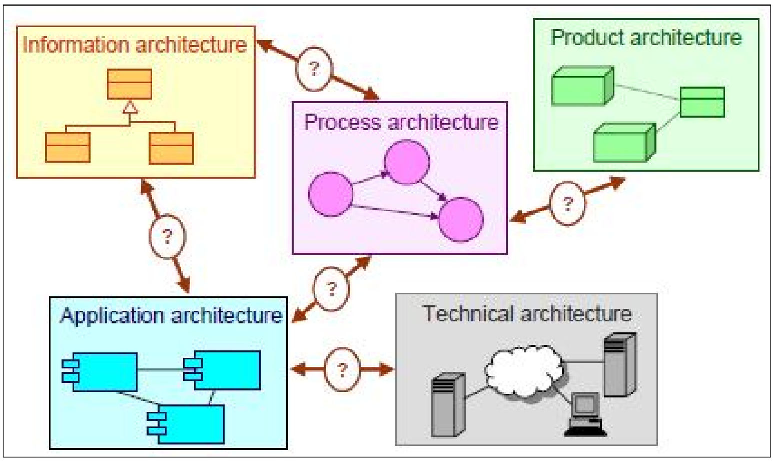


Fig. 2.1. Relationship between architectural domains.

ArchiMate provides a means for integration, by allowing the creation of models that show high-level structures *within* domains and the relationship *between* domains. Based on the work of [Jonk 03],[Buur 04] it is clear that there is a strong need for an integrated architecture language which focuses on concepts of modeling the relationships between architectural domains (see **Fig. 2.1**). The design of the ArchiMate language encompasses concepts that make it possible to inter-relate models used in other languages.

2.1.1 Enterprise Architecture

ArchiMate is a standard for modeling enterprise architectures. Modeling tools based on enterprise architecture merely focus on interdomain relations. Thereby domain interdependencies exist, which have to be drawn and which are needed to align designs in the different domains. When taking this perspective into account using an enterprise architecture language, it makes it possible to:

1) model any 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 and consequently,

2) model the relevant relations *between* the domains. Another important property of an enterprise modeling language is a formal foundation.

This means that models can be interpreted in an unambiguous way, and that they are suitable to automated analysis [Boer 05]. The concepts of this language are sufficiently generic and expressive to model many of the aspects within specific domains. Although, it is clearly not the intention to introduce a language that can replace all the domain-specific languages that nowadays exist. For specific (detailed) designs of, for example, business processes or applications of the existing languages, these are likely to be more suitable. However, it is remarkable that ArchiMate fits itself as much possible into the modeling standards that exist in the different domains.

2.1.2 Business Processes Modeling

The Business Process Modeling Notation (BPMN) is a standard part of the Object Management Group (OMG). The main purpose of BPMN is to provide a uniform notation for modeling business processes in terms of their activities and relationships. BPMN itself only defines a concrete syntax, i.e., a uniform (graphical) notation for business process modeling concepts. However, there is a formal mapping to the XML-based business process execution language WSBPEL. It serves as a common basis for a variety of business process modeling and execution languages. BPMN is restricted to the process modeling, which means that when modeling applications or infrastructure, the language is not suitable since these domains are not covered by the language. Chapter 3, in Sect. 3.2, describes BPMN in more detail for a wide comprehensive understanding of its design features.

2.2 Design Principals of Modeling Languages

When we compare two or more related modeling methods or techniques, it is convenient to look at earlier comparisons and discuss them. A better understanding can be done in a more appropriate way, when selecting relevant parts of the modeling languages. A number of principles need to be taken that are of important value [Paig 00], when *designing* modeling languages and are applicable to new and for improving existing modeling languages. Recent work shows an approach and accompanying process for the development and use of architecture principles [Gree 11]. Underlying these principles, modeling languages need to be practical, usable, accepted and of lasting value. The key design principals are elaborated in this section.

2.2.1 Simplicity

Starting with the first principle, *simplicity* keeps the language simple and more suitable in use. When designing a modeling language this should be taken into account. If a modeling language is simple, then it will be small, much easier to remember the operations to accomplish a task (memorable), and it can be learned in its entirety by its users (simple). It is easier to aim for *simplicity* of a modeling language than it is to aim for a language that satisfies the goal of *modularity (i.e.* applying languages by understanding only a subset of it), because it is more difficult to achieve modularity. One of the key design principles of ArchiMate is the fact that the language should be as compact as possible, but still suitable for the most modeling tasks [Lank 09a].

2.2.2 Uniqueness

The principle of uniqueness, also called as *orthogonality*, can be expressed easily. A language that holds the principle of uniqueness provides one good way to express every concept of interest, and it avoids providing more than one. Modeling languages need to kept concepts as small as possible due to the fact of avoiding duplication of features. Consequently, the language will be more explainable. A feature should be included in a modeling language if it is necessary for modeling a required concept and if there is no way of modeling it using current features. The intent with uniqueness is to have languages defined by a small number of *powerful* features that may be useful in more than one context. By keeping the number of features small, it is easier to understand the consequences of using the features together. The formal modeling language Petri Net, which is described in more detail in Sect. 3.3, fulfills the property of uniqueness.

2.2.3 Consistency

[Meye 92] clarifies the consistency principle. Consistency means that there is a purpose to the design of the language. All features that are included or are to be added to the language must contribute to this purpose. Any feature that does not support the purpose must be discarded. ArchiMate is an obvious example that shows the principle of consistency. Its purpose is to support seamless and reversible development. Any other feature which is not in line with seamless and reversible development should be discarded. Unfortunately there are some languages in which the consistency principle is not clear (yet). This lies in the fact that there are no precise design goals. Consistency of language should not be confused with consistency of the *models* produced using the language. Implementation of these models needs to be checked for consistency and to automate this process is questionable. For that reason, it is difficult to check on consistency of models due to ambiguity of the constructed models with many different relationships and abstractions.

2.2.4 Seamlessness

The seamlessness principle contributes to being able to generate codes from models, and also is a significant contribution towards producing maintainable software. Seamlessness allows the mapping of abstractions in the problem space to be implemented in the solution space without changing notation, thus avoiding the impedance mismatches that often arise throughout the development process. In all stages of the software lifecycle, developers work with the same kind of abstraction, e.g. classes, processes, etc. At the end of development, a tool - typically a compiler - will have to render some executable code from the design. Modeling languages for OO development are well-suited to satisfy this principle. As described earlier, BPMN supports the principle of seamlessness, mentioned as seamless development. Different views of the models may automatically be generated. This model contains an implementation of other pieces of information. So, seamlessness is guaranteed. A contrasting language and method that supports seamlessness is the formal language Petri Net, where abstract machines are used throughout development until codes are generated automatically by a specialized tool.

2.2.5 Reversibility

The principle of reversibility contributes to the production of a maintainable software, and to producing better documentation for software systems. The principle of reversibility requires that changes made during one stage of the development lifecycle can be automatically reflected back to earlier stages. To clearly explain this design characteristic, reversibility means that a modification made to an implementation class written, e.g. JAVA can be reflected in diagrammatic models written in e.g. UML. So, this captures the notion of feeding back to the design level pragmatic constraints from the implementation. Reversibility, combined with seamlessness, allows

programs and models to be kept in sync, and thus helps to create and maintain system documentation. Changes made to models can be reflected in code; and the other way around, changes made to the code can automatically be reflected in changed models. This is exactly what is required in the maintenance process, as well as to further future maintenance. The rationale for this is that models will be kept up to date with the code; otherwise the code will be maintained and the documentation provided by the models will be of less use without reversibility. A requirement for reversibility is that it can be supported by tools. BPMN (WS-BPEL) based tools support this principle that also generates code for particular programming languages (XML). The primary focus of these tools and languages is to support production of architectural descriptions from programs. With formal modeling languages generally, it is difficult to support reversibility, as this would require automatic production of formal specifications from programs.
Several reasons can be given to the above-mentioned complexity: a program may deal with many different formal specifications or elements of a specific type from a program cannot be mapped or transformed to concepts available in a specification language, for example.

2.2.6 Scalability

Scalability focuses on the extent to which the modeling language can be used for a wide range of systems. Ideally a modeling language should be used for both simple, small systems as well as advanced, large systems. This means, that the modeling language can be used for relatively simple modeling systems with a few components and interrelations, but also for modeling systems with large numbers of components and interrelations. Therefore, modeling languages must meet certain requirements to deal with scalability. First of all, they must provide a concise mechanism for describing the fundamental abstractions for their problem domain. Besides, the modeling language must also cope with several levels of abstractions to hide the details. Eventually the language must also provide a *grouping mechanism* that allows the modeler to collect abstractions, name them, and hide their details. Formal methods of formal modeling languages like Petri Net can cope with large problems due to scalability. BPMN provides a structuring mechanism by having the ability to hide details related to abstractions.

2.2.7 Supportability

The principle of supportability states that a modeling language should be designed to be implementable by humans and supportable by software tools. Modeling languages are designed to provide the modeler a set of modeling concepts in order to produce models (for a specific domain) graphically. A quick way to draw models is done by the modeler himself using a pencil and paper or a classical whiteboard. Sometimes it can be useful to use software tools to support or help the modeler in his work in producing correct models, in generating programs from models and in producing models from code i.e. for reverse engineering purposes. It is inevitable that large software systems need software tool support, because they can provide help in drawing, managing, and maintaining models during the several stages in the modeling process. This places restrictions on the notation syntax (i.e. it should also be easy to draw and display on a computer screen, it should be concise) and the semantics (i.e. it should be defined in such a way that it can be (semi-)automatically translated into code, and possibly the other way around in terms of reverse engineering, although to do this appropriately it requires in certain cases human adjustment. With formal modeling languages like Petri Net, it has been designed with tool support in mind. Arising from this, the modeling language ArchiMate has aimed to provide a foundation for visualization and analysis techniques.

2.2.8 Reliability

The goal of software development is to produce quality software. There are many definitions to what is meant by quality, but a common factor is that quality software is *reliable*. This firstly means that reliable software meets their specifications e.g. via formal analysis or traceability combined with testing; and secondly reliable software reacts appropriately in case the user is given unexpected or erroneous input e.g. via design-by-contract mechanisms, or by use of error and exception handling. Reliable software is therewith *robust*. Modeling languages should provide support for ensuring that the models being produced are consistent as discussed in section *2.2.3* to eliminate the ambiguity. Methods for producing software must emphasize quality. Therefore it is important that modeling languages support the production of reliable programs. In the past, few improvements have been made on developing correct software in part by adding formal semantics to ensure reliability.

2.2.9 Space economy

The final principle, space economy, is quite simple. Space economy states that models should be as concise as possible to limit the space on the printed page. This has to do with the understandability of the language; smaller models have less to understand. In addition to these, the maintainability of the models performed by modelers and tools require less work. This principle has no delineation as long as space economy preserves its simplicity and understandability of the language as well.

2.2.10 Underlying modeling principals of ArchiMate and BPMN

From the modeling perspective, some principals reflect both the architecture of ArchiMate and BPMN language that were a primary source for the design of the language.

Concept Coverage** - **Scalability

Several domains for grouping concepts have been identified, such as product, process, organization, information, application, and technology (infrastructure, system development, and maintenance). The concepts in the ArchiMate language must cover the concepts in these domains [Lank 05], while the BPMN language covers only the *process* aspect.

Enterprise level and project level concepts – Simplicity

At an enterprise level, it is important to be able to represent the core elements from the different domains such as product, process, et cetera, as well as the coherence between these aspects. In enterprise architecture models, *coherence* and *overview* are more important than specificity and detail. This also implies the need for more coarse grained modeling concepts. At a project level, it is important to represents the core elements from a specific domain (in this case business processes). Thus, specificity and detail are more of importance in business architecture models.

Concept mapping – Supportability

ArchiMate is intended to connect heterogeneous architectural domains such as processes (i.e. the fine-grained business concepts in BPMN) and applications (i.e. the fine-grained application concepts in UML), rather than replacing them. Organizations or individual architects must be able to keep using their own concepts and descriptions in development projects. This requires a mapping from the coarse grained concepts in ArchiMate to the fine-grained concepts used in languages at project level.

Unambiguous definitions of concepts - Uniqueness, Consistency

The meaning and definition of the modeling concepts offered by the language is unambiguous. Each concept of both ArchiMate and BPMN visualization techniques is required to be unambiguous with respect to informal description, specialization, notation, properties, structuring, rules and restrictions and guidelines for use.

Structuring mechanisms** - **Scalability, Space economy

ArchiMate supports the use of composition / decomposition, generalization / specialization, and aggregation of concepts. BPMN supports a structuring mechanism in terms of grouping concepts based on common properties by means of grouping relation, pools and lanes.

Abstraction - Consistency

ArchiMate models the relations at different abstraction levels, which is one of the key designs by means of formulating relations between concepts, groups of concepts or different architectural domains. BPMN abstract only the process domain.

Analysis of architectural properties - Seamlessness

ArchiMate is designed with the principle that it offers the possibility to perform qualitative and quantitative analysis [Lank 09b] of properties of architectures.

Impact of change analysis - Supportability, Reliability

Impact of change analysis must be supported. In general, such an analysis describes or identifies effects that a certain change has on the architecture or on characteristics of the architecture.

Executable environment - Seamlessness

The underlying executable mechanism of BPMN, which one part is WS-BPEL, generates code that is applicable for execution of business processes.

The principles can be summarized in table 2.1 thus:

Table 2.1. Modeling design principles

Principles	*Description*
Simplicity	No unnecessary complexity is included in the language.
Uniqueness	There are no redundant or overlapping features.
Consistency	Language features cooperate to meet language design goals.
Seamlessness	The same abstractions can be used throughout development.
Reversibility	Implementation changes can be propagated into the model.
Scalability	Large and small systems can be modeled.
Supportability	The language is usable by humans, and supportable by tools.
Reliability	The language encourages the production of reliable software.
Space economy	Concise models are produced.

Chapter 3

Modelling languages: architecture of ArchiMate, BPMN & Petri Nets

This chapter describes the underlying architecture of the ArchiMate modeling language. The core concepts of the languages ArchiMate, Business Process Modeling Notation and Petri Nets are described in further detail. In order to understand its architecture framework the metamodel has been explicitly formulated using visualizing technique (ORM). Moreover, modeling concepts at different levels, i.e. the enterprise and project level are introduced and relations between concepts are discussed. Also examples are provided, which cover (parts of) the modeling language in order to understand the underlying architecture.

ArchiMate

ArchiMate is a modeling language for describing the architecture of the enterprise by providing visualization techniques. It is a design tool which supports IT architects as a basis for visualizing and analyzing techniques to describe the enterprise's architecture. This language technique has been constructed in such a way that it offers a set of generic concepts within domains and its relationship between these different domains that allows coherent modeling of enterprise architecture descriptions. As a result, describing and relating architectural domains has been made possible by providing a fundamental uniform structure. Such an integrated architectural approach, allow enterprises in assessing the impact of design choices and changes.

ArchiMate uses the perspective that enterprises are considered as a set of layered systems. Resulting that the ArchiMate metamodel distinguishes three layers. The first layer is the *business layer*, following the *application layer* and the *technology layer*. Thereby the framework consists of the extended concepts *active structure*, *behaviour* and the *passive structure*. The set of concepts are further extended including the *internal/external* view and the *individual/collective* view which forms the three dimensions of architectural concepts.

Enterprises require an architecture modeling language that fulfills consistency in alignment and to simplify coherent modeling of enterprise architectures. There is a strong need for integrating models and to describe the coherence between them. ArchiMate plays in model integration thereby a central role.

Coherent Modeling

Many existing architectural approaches are used in practice to model the enterprise's architecture with respect to different domains of expertise. ArchiMate focus on heterogeneity of architectural domains that are used to describe the architecture, which makes it much easier to inter-relate these different domains (see **Fig. 1.1***)*. ArchiMate has clearly *not* the intention to replace existing domain specific modeling techniques but merely wants to model the global structure *within* each domain and to address the relevant relations *between* the domains. However, ArchiMate encompasses sufficient generic expressive concepts to model many of the aspects within specific domains. The role of the ArchiMate modeling language aims to provide high-level modeling *within* a domain and modeling relations *between* domains (**Sect**. **3.1.2**). Furthermore it acts as an instrument for visualization and analysis techniques.

3.1 The ArchiMate Modeling Language

ArchiMate Modeling the Enterprise Architecture

ArchiMate is a standard for enterprise modeling for describing the enterprise architecture. The core of this language lies in the coherence / relations between concepts. In particular how the relations between different layers or aspects of an architecture can help to gain insight into the alignment between for example the business processes and their supporting application or the applications and the technical infrastructure [Lank 09a]. ArchiMate has his own typical architecture framework where all concepts can be defined and consists of three layers that are connected to each other through the so-called services as described earlier. Five core modeling concepts (object, service, behaviour, interface and structure elements) can be discerned at each level. The boundaries between these layers are not strict and are reflecting the enterprise's division. Layers are only explicitly related to layers directly above or below them.

3.1.1 A Language for Modeling the Enterprise Architecture

To handle the complexity of modern information-intensive enterprises, architects need ways to express architectures as clearly as possible for their own understanding and for communication with other stakeholders. But often the case is that architects coming from different domains use their own description techniques and conventions. Sometimes their descriptions are too detailed such as UML that it is difficult to understand for non-experts or contains informal pictures in which the meaning is not well defined. This leads to misunderstandings that interrupt the collaboration of architects and other stakeholders. Besides, it makes it very hard to provide tools for visualization and analysis of these architectures. ArchiMate needs to bring some added value to these similar existing model integration problems. In these section concepts will be introduced of the ArchiMate modeling language and some examples will be given to illustrate how they can be used. As mentioned before, special attention is paid to the relations between concepts.

3.1.2 Describing Coherence

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 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 business processes of an organization is to realize their products and software applications support business processes, 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 modeling enterprise architectures should focus on interdomain relations (see **Fig. 3.1**).

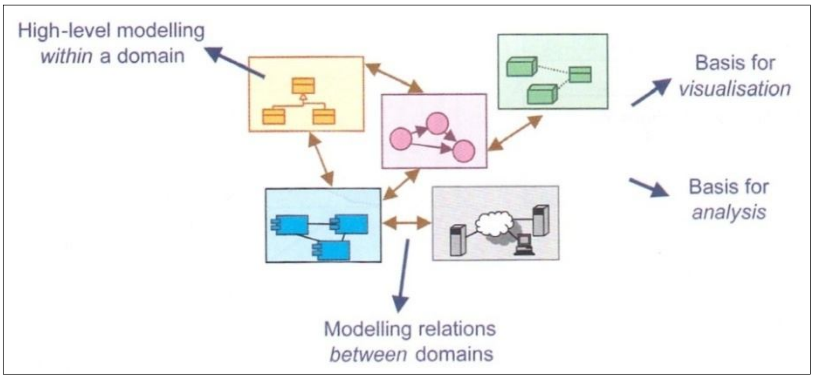


Fig. 3.1. The role of ArchiMate language.

3.1.3 Service Orientation and Layering

Services play a central role in ArchiMate as these services are the *'connectors'* in the ArchiMate framework between the different layers. The ArchiMate framework is based on service-oriented models as this means that services are used by the higher layers which are provided by the lower layers. In line with this, services can be provided by organizations to their customers, by applications to business processes or by technological facilities to applications. Service layers with services made available to higher layers are interleaved with implementation layers that realize the services. Within a layer, there may also be internal services, e.g., services of supporting applications that are used by the end-user applications. This leads to a stack of service layers and implementation layers (see **Fig. 3.2**).

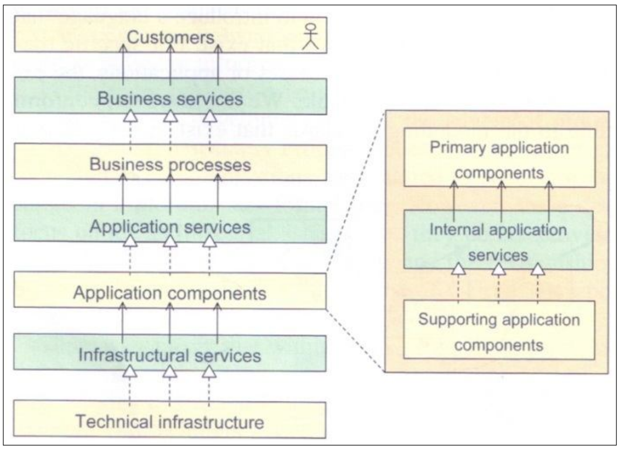


Fig. 3.2. Service orientation as a layered view.

These layers are linked by *used by* relations and *realization* relations, showing how services are used and realized. Before concretizing more concepts, that are specific for a certain layer, we can now distinguish three layers which are from top down the business layer, the application layer and the technology layer as described earlier (**Sect. 1.1**).

3.1.4 Dimensions of modeling

The general structure of the models within the different layers is similar. Same types of concepts and relations are used, but their nature and granularity differs from each other. Due to this uniformity, models that are created, derived from different layers, can easily be aligned with each other.

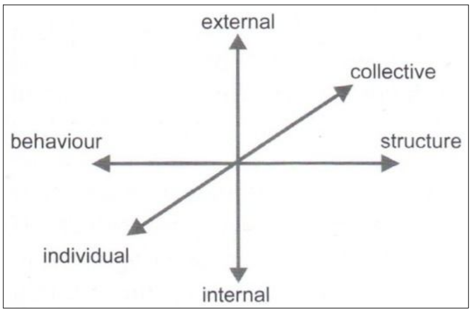


Fig. 3.3. Three dimensions of architectural concepts.

The picture as depicted in **Fig. 3.3** illustrates how architectural concepts can be identified that makes use of the same general structure.

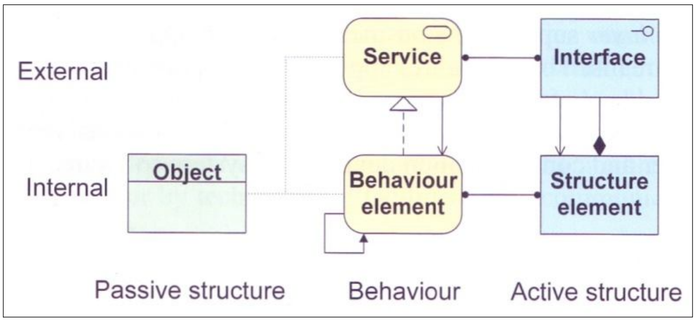


Fig. 3.4. Core concepts of the AM language.

The example in **Fig. 3.4** illustrates the core concepts that are found in each layer of the language. On the right side we see the *structure* aspect. In the center state the *behavioral* aspect. There is a close relationship between these two aspects: behavioral concepts are assigned to structural concepts, to depict who or what performs the behaviour. The *active* structure elements - which can be found on the right side - show the actual behaviour. On the left side stated the *passive* structural elements, which means *objects* on which behaviour is performed in terms of information objects or physical objects as the focus lies in the domain of information intensive organizations. Further distinctions are made between an *external* view (depicted as the top side) and an *internal* view (depicted as the bottom side) of systems. These views reflect the service orientation principles as described earlier (**Sect. 3.1.2)**. The *service* concept represents a unit of essential functionality that some entity e.g. system, organization or department makes available to its environment. The exposed service has some value for certain entities in the environment which can be denoted as the 'service users'. For the functional aspects such as the quality of service or costs are relevant and can be specified in a contract or service level agreement (SLA). Services are accessible through *interfaces* which are depicted in **Fig. 3.4** as the external view of the structural aspect.

Looking at the internal realization of services and interfaces, a distinction can be made between behaviour that is performed by an *individual* structural element and *collective* behaviour (i.e. interaction) that is performed by a collaboration of multiple structural elements. Interaction can trigger other behaviour elements or interactions but can also be triggered by them as well. In line with this, an interaction can be treated as a specialization of a behaviour element likewise collaboration can be treated as a specialization of a structure element which enables recurrence. An addition hereby is that this collaboration mechanism besides individual structure elements, may also aggregate other more fine-grained collaborations. The structure of the ArchiMate language and the relevant layer-specific concepts are summarized in **Fig. 3.5**.

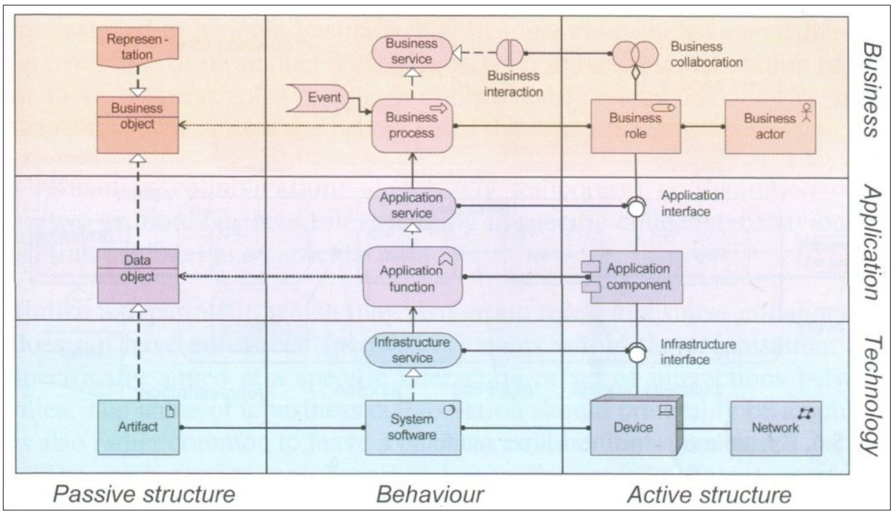


Fig. 3.5. Main concepts of the ArchiMate language.

Some additional concepts located in the three layers will be further explained. Concise example models are used to illustrate the use of these concepts that holds the general structure that make use of the three dimensions of architectural concepts, but differs from each other with respect to their exact nature and granularity.

3.1.5 The Business Layer

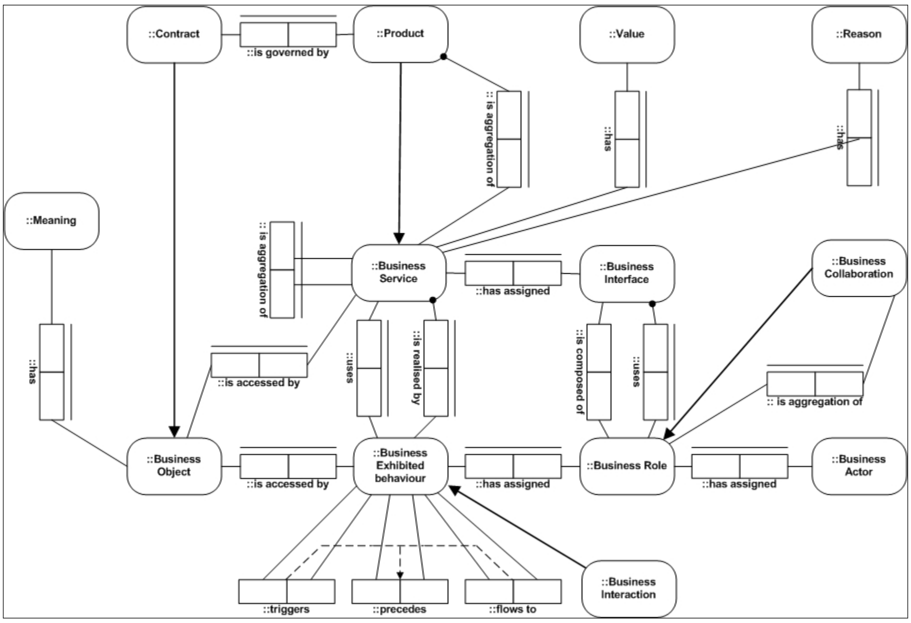


Fig. 3.6. Business layer metamodel.

Business Behaviour Concepts

We start from a business layer model where the use of the business concepts and their relations are illustrated through the example models concerning a fictitious insurance company called 'ArchiSurance'. The example models concerns about how to handle the claims in an insurance when customers or assurers report a damage that has occurred to determine if the claim will be accept for receiving their compensation. The metamodel illustrated in **Fig 3.6** gives a metamodel of the language at the business layer. The business layer concepts and its relations conform to the core concepts that make up the general structure discussed in the previous section. Taking this approach, at this layer-specific concept we distinguish business structure concepts. See Chapter 9 '**Glossary**' for the definitions of the layer specific concepts, which are underlined with a superscript star.

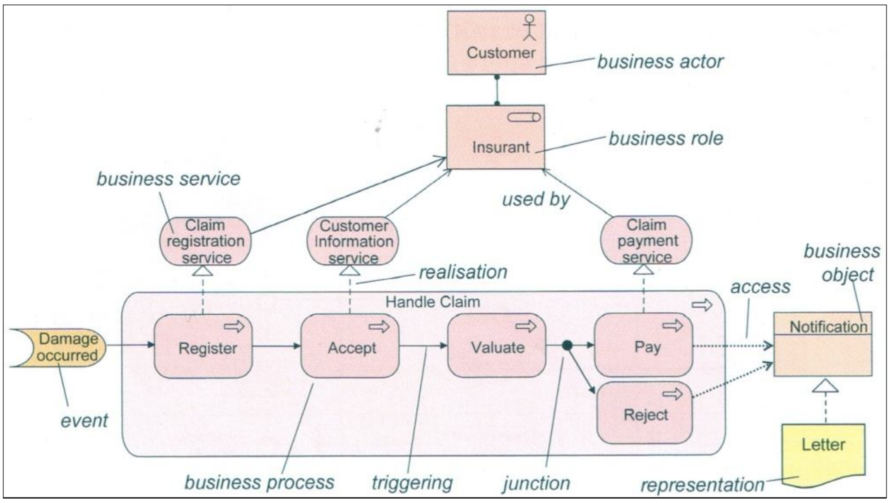


Fig. 3.7. Example of a business layer model.

An example of a business layer model is depicted in **Fig. 3.7**. The structure aspect at this layer refers to the organization structure. A business actor* (e.g. 'Customer') makes up the organization and their relationships and it can be fulfilled by a single person (e.g. a customer or an employee) but also a group of people and resources that have a permanent status in the organization (e.g. department or a business unit).

This type of actor assumes a certain role in the organization which is closely related to the work the actor fulfills: a business role* (e.g. 'Insurant'). It is preferably to use a noun* for the name of a business concept. Mostly all names of the business concepts should preferably use by a noun. Often the name of business collaboration is left open. The possibility exists that multiple actors fulfill the same role and the other way around that a single actor can fulfill multiple roles. A business process or function (in the example business layer model: 'Register', 'Accept', 'Valuate', 'Pay', 'Reject') can be seen as the internal behaviour that is assigned to a (single or multiple) business role(s).
When more than one business roles are involved in this situation, typically a collaboration occurred, than this will leads to a collective behaviour, which is exactly the aggregation of the single roles separately: a business collaboration*.

In comparison with a business actor such as a business unit, which may contain also multiple roles, collaboration has not a permanent status within the organization. An interaction is aimed at a specific interaction or set of interactions between roles. As we know that services are accessible possibly through a number of interfaces, like mail, telephone, or internet, these are typically business interfaces*. The picture depicted in **Fig. 3.7** on the right side we see the business object that represent the information in which the business consider as relevant in their point of view.

Commonly a business object* is used to model an object type, where several instances may exist in the organization. In this case a 'Letter' exists which is an instance of the object type 'Notification'. Business object are passive as they undergo some behaviour which are performed by business actors; they cannot trigger or perform processes. A business object can be accessed in terms of (e.g. created, read, or written) by a business process, function, interaction, event or service. In the example model the business object are accessed by only two business processes ('Pay' and 'Reject'). Different specializations exist at the business layer (see **Sect. 3.1.9**).

Representations* can take different forms in terms of medium (e.g., electronic, paper, audio, or video) or format (e.g., HTML, PDF, or charts). In the used example model a paper form is perceptible. A single business object can contain multiple representations. A remarkable point is that a representation always belongs to one specific business object.

Business Behaviour Concepts

Business services* are used to expose business functionality to its environment which is realized by business behaviour including a number of business concepts: business process, business function or business interaction. The name of a business service should contain a verb ending with '-ing' or explicitly contain the word service e.g., 'Claim registration service', 'Customer information service' and 'Claim payment service'). A distinction can be made between 'external' business services and 'internal' business services. The external business services are aimed to external customers outside the 'business', whereas internal business services aimed to supporting functionality to processes or functions within the organization. To exclude confusion of a function and a process, - because some organizations use the term business service to refer to application services used by the 'business' and (business) function to indicate an external unit of behaviour that is implementation-independent - we now distinguish a process view from a function view of behaviour. The example of Fig. 3.8 illustrate that botch concepts can group activities.

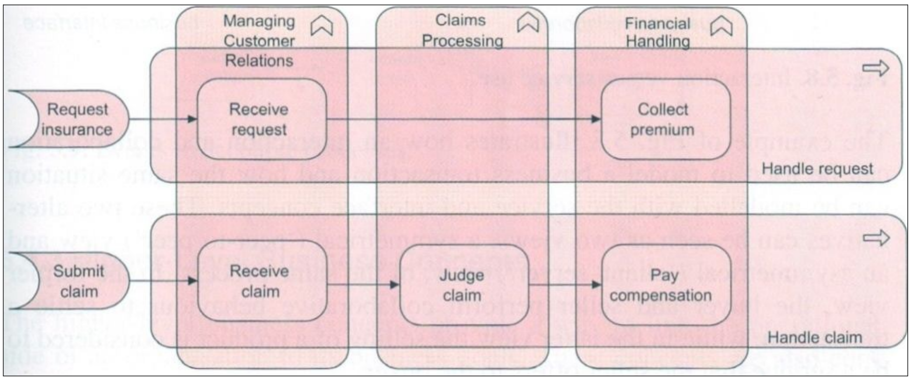


Fig. 3.8. Business processes versus business functions.

The difference between these two lies in the grouping criteria. 'Managing Customer Relations' designate a business function that contains the activity 'Receive request' and 'Receive claim'. The business process 'Handle request' contains by using their grouping criteria 'Receive request' followed by 'Collect premium'. The name of a business process should contain a verb in the present tense e.g., 'Receive request'. It is clear that business processes can relate more than one business functions and conversely.

A business process* groups internal behaviour with the intention to produce a defined set of products and services, whereas a business function* groups internal behaviour based on e.g., required skills, capabilities, resources or support. The name of a business function should keep the following convention: a verb ending with '-ing'. In the example of **Fig. 3.9** this would be 'Managing Customer Relations', 'Claims processing' and 'Financial Handling'. Thus, business processes are defined based on the *products* and *services* that the organization offers, while the business functions are the basis for the assignment of resources to tasks and for the application support.

A business interaction* can be regard as a unit of behaviour which is performed by two or more business roles within the organization. The example of **Fig. 3.9** illustrates how an interaction and collaboration can be used together to model a business transaction. This can be modeled in the same way using service and interface concepts. Two views, respectively symmetrical and asymmetrical view, can be interpreted of the same process. On the left side of **Fig. 3.9** the buyer and seller interact with each other i.e. collaborative behaviour to build a transaction, while on the right side of **Fig. 3.9** illustrate the selling of a product is being considered as a service that the seller offers to the buyer.

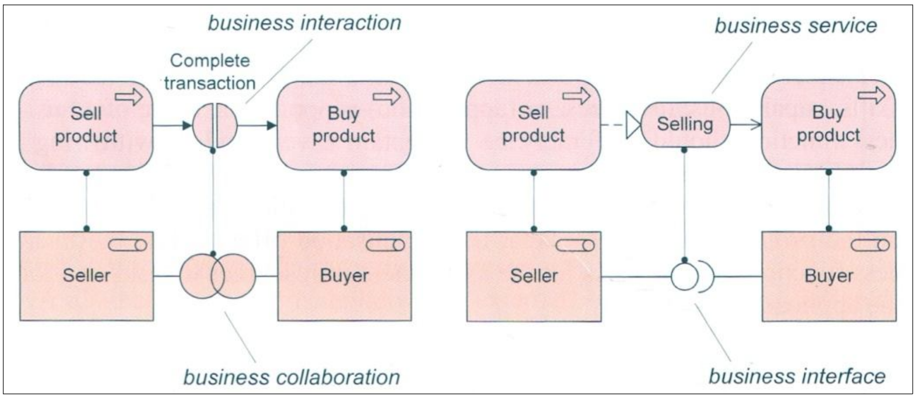


Fig. 3.9. Interaction versus service use.

Business event* is an event which may happen un- or expectedly within the organization (generated by other processes) or inside the environment of the organization (such like events coming from a customer) that influences the business behaviour in terms of business processes, functions, and interactions. It can be used to model events that trigger the behaviour. It is even possible to use other types of events to e.g. interrupt a process. The name of a business event should contain a verb in the past or present tense e.g. 'Claim received' or 'Claim has arrived'. Typical to a business event is that they are instantaneous, which means that it does not have duration unlike business behaviour.

The example of **Fig. 3.10** illustrates how processes can be decoupled by using an event. The left upper part shows how 'Claim received' event, an ingoing event, starts a process called 'Assess claim' and eventually leaves with a 'Payment request sent' event which is considered to be an outgoing event. The right upper part shows how the 'Payment request sent' event triggers the process 'Pay compensation'. These two processes are separately modeled. When combining these two separately processes, the linking event can be omitted by replacing it by the triggering relationship between these two processes. This will leads to the bottom part process.

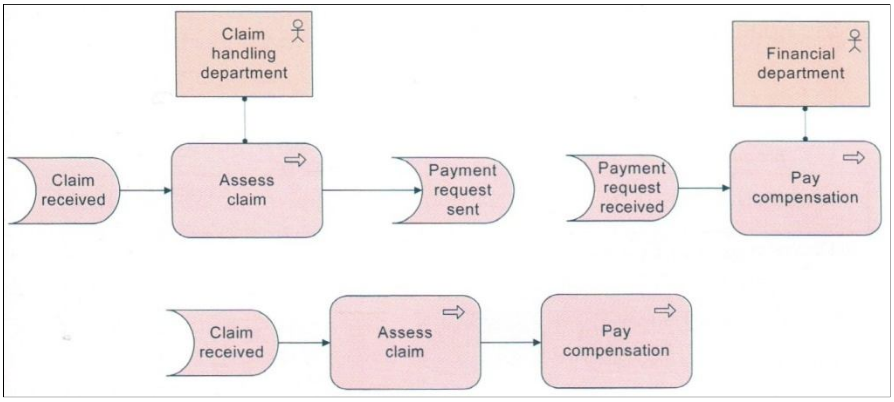


Fig. 3.10. Event to decouple processes.

Higher-Level Business Concepts

The higher-level business concepts have been introduced to seamlessly connect the operational side of an organization with its business goals. A product* is the whole collection of all interrelated services including the rules which encompasses guidelines or set of agreements on how to use these services which is offered as a complete package to customers in- or externally. An example is given in **Fig. 3.11** where a product has been defined, by grouping services with the accompanied contract as a guideline for using the services. The collection of services associates with the offered product often concerns business services, but application services are also conceivable. In general, the product concept is used to specify a product type. Some organizations have a number of product types, which grouped associated services belonging to that specific product type. Compared to the underlying processes that realize the product, product types are quite stable. When a customer decides to insure their travel, the customer becomes an insurer of the travel insurance. This 'buying' activity is one of the services associated with a product, which results in a new instance of that product. This introduces the possibility that some services exists to modify or cease a product. The name of a product is usually for communication purposes towards the customers or a generic noun. In the example of **Fig. 3.11** the product name is called 'Travel Insurance'.

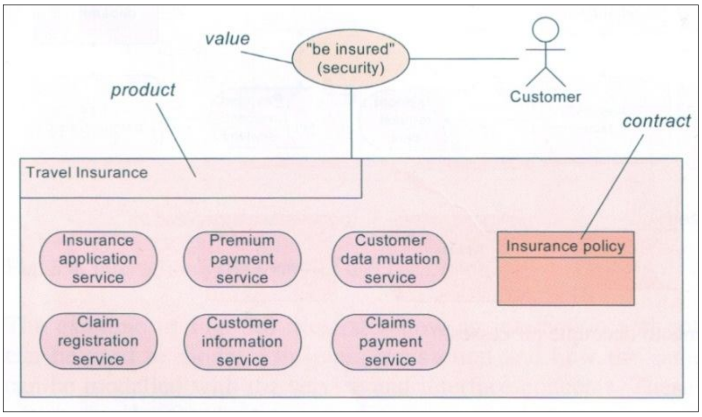


Fig. 3.11. Services grouped into a product.

A contract* concept can be used to model a contract (legally or informally), that is part of a product. A contract, which is a specialization of a business object, sometimes includes or takes the form of a Service Level Agreement (SLA), a specification of agreement concerning the functionality and quality of the services associated with the product.

The value* of a product or service is that which makes a party appreciate it. In the value chain of a product or service a value applies to what a party acquires by offering some product or service, or by obtaining access to it. It can be either way expressed in money, but also in non-monetary value e.g. practical/functional value, the value of information or knowledge. In the proposed example in **Fig. 3.11** the value is more of protective nature i.e. "be insured"/ (security). The name of a value can vary, but there are guidelines for the designation of the name in case of 'functional' value of a service is concerned; express it as an action or state that can be performed or reached as a result of the corresponding service being available.

A meaning* is associated with a business object or its representation and represents the informative value of a business object for a user of such an object. Meaning is sometimes aimed for a specific user or for a particular category of users, when interpreting a representation of the object. The name of a meaning should be a noun or noun phrase that clarifies this to distinguish them from business object and representation.

3.1.6 The Application Layer

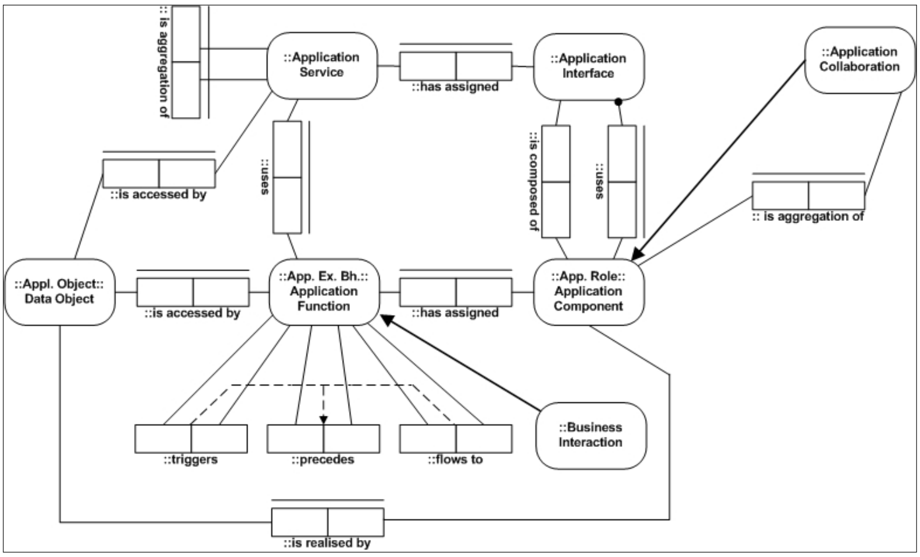


Fig. 3.12. Application layer metamodel.

The example of **Fig. 3.13** illustrates the use of the application concepts. In the previous section we discussed the business layer concepts, now the concepts of the application layer are explained in more detail. We gradually built up the metamodel of the application layer still using the fictitious 'ArchiSurance' example which is elaborated, to clarify the relationship between these layer specific concepts. Eventually an overview of the metamodel can be modeled. After explaining these central concepts, the relationship between the application layer and the business layer (i.e. alignment) can be then modeled.

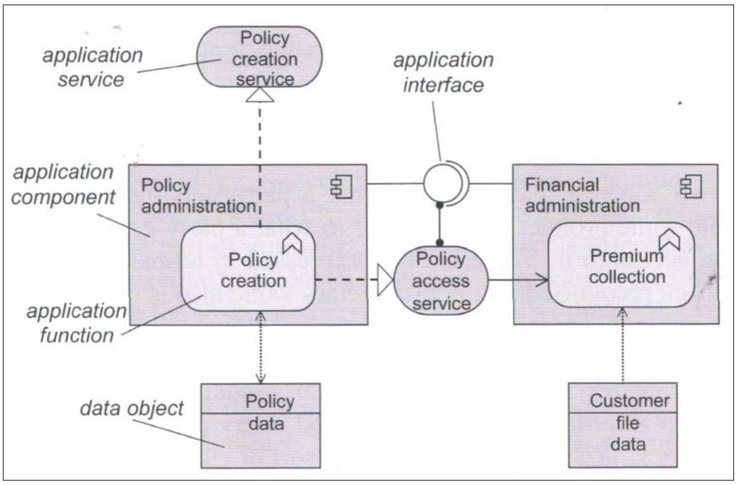


Fig. 3.13. An example of an application layer model.

Application Structure Concepts

The main structural concept is the application component* that is used to model any structural entity in the application layer. Any structural entity refers to software components (might reusable), which can be part of one or more applications, complete software applications, subapplications, or information systems. As we can see in Fig. 3.13, an application component can possess application functions e.g., the 'Policy creation' and makes the functionality of its contents available through a service and an interface (see also Fig. 3.9). The name of the application component is preferred to be a noun.

Another concept is an application collaboration* which is like the business collaboration an interrelationship, but between components. In the application architecture this is an important feature. Thus, application collaborations are aimed to perform application interactions between two or more application components. In the example the application components that cooperate are 'Policy administration' and 'Financial administration'; they are communicating through the application service 'Policy access service' and the *application interface*. Also the name of application collaboration is preferred to be a noun.

Cooperation of application components can be done through the application interface* that is the location where the services of a component can be accessed to provide its functionality. It also defines some fundamental characteristics of behaviour namely a set of operations or events that are made available by the component. Conversely, a set of operations or events that is required from the environment. It is useful to make a distinction between a *provided interface* and a *required interface* in order to model application-to-application interfaces and application-to-business interfaces. The first one provides the application services internally (to components), while the latter one provides application services externally (to e.g. the business processes).

Generally, we can say that an application interface provide components a way to connect with its environment. The name of an application interface is preferred to be a noun. Similarly to *business object* in the business layer, the application layer used a data object* associated with a component, which is a unit of coherent information that can be perfectly used for automated processing. Like business objects, data objects have a passive character. The name of the data object is preferred to be a noun.

Application Behaviour Concepts

In the previous section business services are provided by the processes. In the application layer these services are provided by the components. Similarly to an application service*, it provides a way to describe explicitly the functionality that components share with each other and the functionality that they make available to the environment. The name of an application service is preferred to be a verb ending with '-ing' or contained the word 'service' explicitly. Application services expose application functions to its environment.

Application function* can be used to model the internal behaviour of an application. The name of an application function is preferred to be verb ending with '-ing' such as 'Accounting'. Application interaction* The name of an application interaction is preferred to be a verb in the present tense.

Business-Application Alignment

The relationship types between business layer and the application layer concepts are:

1. *Used by relationships,* which are located between application service and the different types of business behaviour elements, and between application interface and business role, representing the behavioural and structural aspects of the support of the business by applications.

2. *Realization relationship* from a data object to a business object, indicating that the data object is a digital representation of the corresponding business object.

3. *Assignment relationships,* which are located between application component and the different types of business behaviour elements, and between application interface and business service, indicating that, for example, business processes or business services, are completely automated.

The example of **Fig. 3.14** illustrates the interrelationship (i.e. business application alignment) between the business layer concepts and the application layer concepts, which is realized by the relationship type 1 and 2.

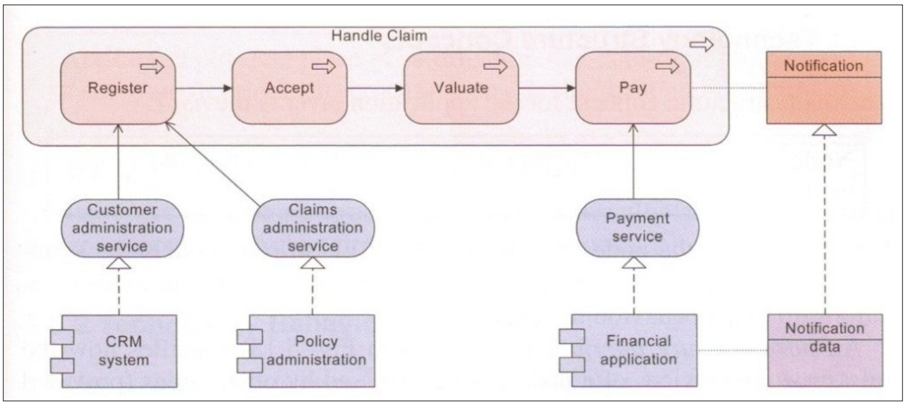


Fig. 3.14. An example of a business application alignment model.

3.1.7 The Technology Layer

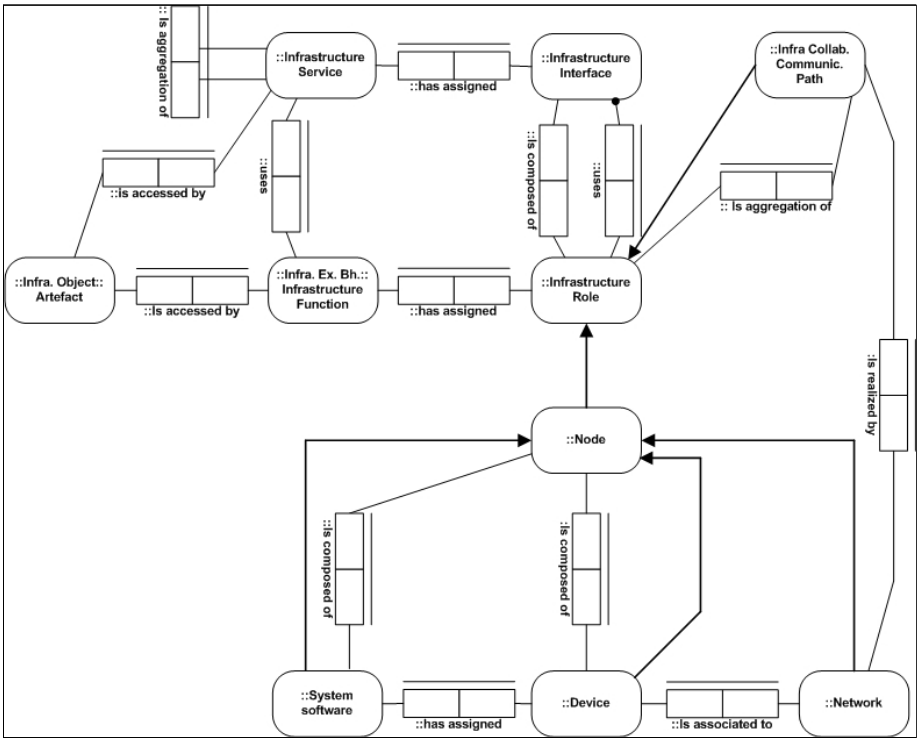


Fig. 3.15. Technology layer metamodel.

Technology Structure Concepts

In the previous section we discussed the application layer concepts, now the concepts of the technology layer can be explained in more detail. The metamodel of the technology layer is shown in Fig. 3.15, where all relevant concepts are modeled in ORM. As the fictitious 'ArchiSurance' example can still be used, this section describes how the relationships between these relevant layer-specific concepts are related. The example of Fig. 3.16, illustrates the use of the technology concepts. Consequently, the relationship between the technology layer and the application layer (i.e. alignment) can be then modeled analogues to the business-application alignment (see Fig. 3.17).

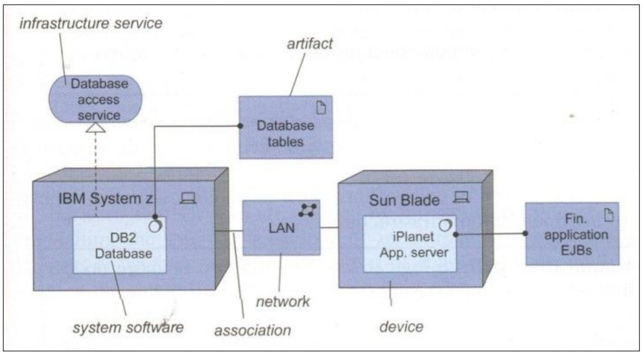


Fig. 3.16. An example of a technology layer model.

In the technology layer the main structural concept is the node*, which models the structural aspect of an infrastructure. A node represents a (logical) resource with computing capability, which may be assigned to an *artifact* for its execution purposes (e.g. 'IBM System z', 'Sun Blade'). An infrastructure interface*, similarly to business and application interfaces, specifies how the infrastructure services of a node makes available to other nodes or application components to provide its functionality, or conversely which functionality of the node can be required from its environment.

A specialization of a node can be denoted as a device* (e.g. 'Sun Blade'). It is a physical resource with processing capability, which can be also used by an artifact for its execution purposes. Typically, a device is used to model hardware systems e.g. mainframes, PC, or routers. It is allowed that a device Nodes possibly encapsulate subnodes e.g. a server and an execution environment to model the operating system as shown in the example of **Fig. 3.16**.

To model the interrelationships between technology components, two types can be discerned: a communication path* and a network*. The first one concerns about the exchange of information through a logical connection between two or more nodes, while the latter one realizes a connection between two or more devices ('LAN'). For modeling the representation of files, data objects, applications components. Artifacts are suitable to represent e.g. files, data objects, or application components and can be assigned to a node.

An artifact* represents a unit of physical information that can be used or realized in software development processes or by systems (e.g. 'Database tables' and 'Fin. Application EJBs'). There are no strict rules for naming conventions of the technology structure concepts, but rather taken directly from the corresponding product e.g. 'Sun Blade'.

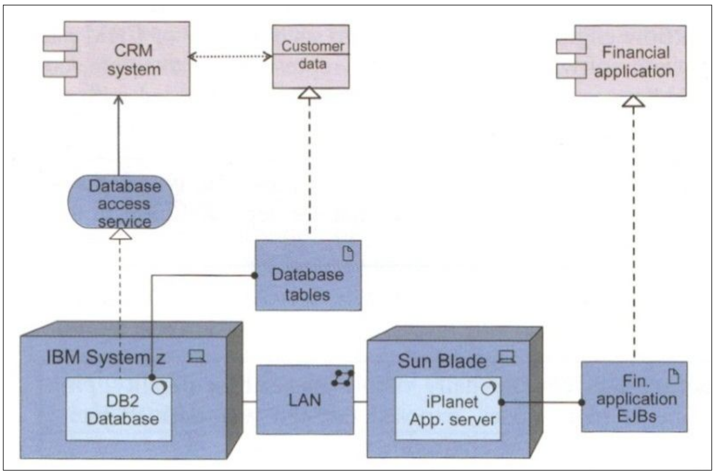


Fig. 3.17. An example of applications supported by infrastructure.

Technology Behaviour Concepts

Another specialization of a node is system software* that is used to model the software environment in which artifacts run. It can be also used to represent communication middleware. Typically, system software is combined with a device representing the hardware environment to form a general node. Services that are exposed from the technology layer used by applications are denoted with the infrastructure service* concept. Infrastructure services are realized by nodes that exposes the functionality to its environment through interfaces. Only external behaviour of the infrastructure components are relevant due to abstractions at the enterprise level. The naming convention for an infrastructure service either must contain a verb in the '-ing' or the word 'service'. In the example given,

Application-Technology Alignment

As the central concepts of the technology layer has been explained, the application layer and the technology layer can be aligned using the two types of relations (see Fig. 3.17), similarly to business-application alignment (see Sect. 3.1.6):

1. *Used by relationship*; infrastructure services can be *used by* application functions and infrastructure interfaces are *used by* application components, which means that the application layer is supported by the technology layer,

2. *Realize relationship*; artifacts can *realize* data objects and application components, indicating that the technology layer (i.e. implementation layer towards the application layer) realizes these application concepts.

3.1.8 Relations

A fundamental view on the enterprise architecture is that it describes the *coherences* within different as well as among domains. We can define a limited number of structural relations. These set of relation concepts with their properties are summarized in ascending order by 'strength' (excluding *grouping*) in table 3.1 which is elementary for describing the relationships. Some of the structural relations are derived from other existing standards like UML (*composition*, *association*, *specialization*) and BPMN (*triggers*). Table3.2 shows the behavioral relations.

Table 3.1. Structural relations.

Relation | Weight	Description (property)
Association | 1	Association relation concept is aimed to model a relation between objects.
Access | 2	Access relation concept is aimed to model the access of behaviour concepts to business or data objects.
Used by | 3	Used by relation concept is aimed to model the use of services by processes, functions, or interactions and the access to interfaces by roles, components, or collaborations.
Realization | 4	Realization relation concept is aimed to link a logical entity with a more concrete entity that realizes it.
Specialization	Specialization relation concept is aimed to indicate that an object is a specialization of another object.
Assignment | 5	Assignment relation concept is aimed to link units of behaviour with active elements (e.g. roles, components) that perform them, roles with actors that fulfill them, or artifacts that are deployed on nodes.
Aggregation | 6	Aggregation relation concept is aimed to indicate that an object groups a number of other objects.
Composition | 7	Composition relation concept is aimed to indicate that an object consists of a number of other objects.
Grouping | N.A.	Grouping relation concept is aimed to indicate that objects belong together based on elementary characteristics.

Table 3.2. Behavioral relations.

Relations	Description
Triggering	The 'triggering' relation describes the temporal or causal relations between behavioral elements, processes, functions, interactions, and events.
Flow	The 'flow' relation describes the exchange or transfer of, for example, information, goods, or value between processes, function, interactions, and events.
Junction	A 'junction' is used to connect dynamic relations of the same type. It can be used to model splits or joins of triggering or flow relations.

3.1.9 Language Extension Mechanisms

Specialization of Concepts

Specialization can be used to define new concepts based on the existing ones. These types of concepts inherit the properties of their 'parent' concepts and possibly may have additional restrictions. In some cases, relationships might only apply to the 'parent' concept, while it is forbidden or not applicable for the specialization concept. An essential characteristic of these types enables extra flexibility in means of customizing the language to the users' preferences and needs, while holding the exact nature of its 'parent' concept. In practice the commonly used the possible situations, where specializations might be needed (see **Fig. 3.18**). Some slight graphical changes or modification of the icon at the 'parent' concept might lead to a new graphical notation for the specialized concept.

Adding Attributes to Concepts

Predefined profiles: these are profiles that have a predefined attribute structure and which can be attached to concept and relations and implemented beforehand in any tool supporting the ArchiMate language.
User defined profiles: via a profile definition language, the user should be able to define his own profiles, and subsequently to extend the definition of any ArchiMate concept or relation with supplementary attribute sets.

Composition of Concepts

Composite concepts can be considered as the combination of two or more concepts of the ArchiMate language, which may be core concepts, specialized concepts or even composite concepts themselves. A composition of a number of concepts may be seen as *multiple inheritances*. This means that the composite concept inherits the properties of more than one ArchiMate concept. Such as a UML *class* concept can be implicitly assumed as the composition of Application function, Application component, Data Object of the ArchiMate concepts.

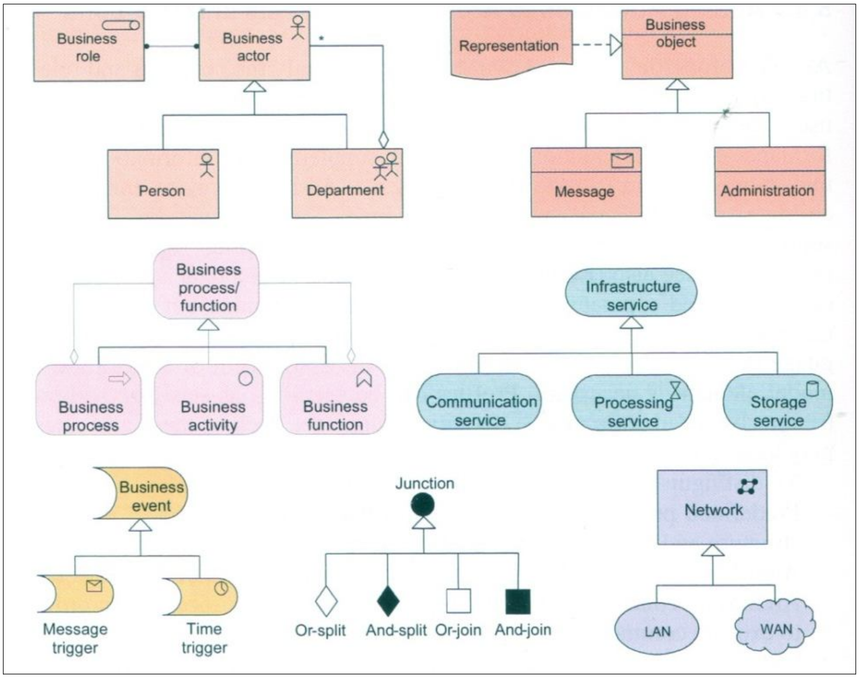


Fig. 3.18. Common used concepts of specialization.

3.1.10 Summary of the ArchiMate Architecture

ArchiMate is a language for describing the enterprise architecture, which allows enterprises to integrate architectural models. ArchiMate focusses on interdomain relations, which means that it is able to either model the global structure *within* each domain (i.e. high-level modeling) and the relevant relations *between* the domains (i.e. modeling relations). This approach seems to be meaningful to also inter-relate modeling tools. In the field of model based system development this could serve as a fundamental basis. One of the key designs in this language is the service orientation aspect, where services are exposed at the three distinct layers respectively *the business layer*, *the application layer* and *the technology layer*. The services act as an inter-layer binding concept that is introduced for *alignment* purposes with respect to the domain specific layers. Through this service oriented character, different domains can be integrated that are essential for providing *coherency* in the description of enterprise architectures. Each layer comprises central concepts that are essential to model relevant aspects of the enterprises' architecture. It also provides a basis for *visualization* and *analysis* techniques [Boer 06], [Gust 09]. A full metamodel of the enterprise modeling language is given (see also **Appendix A**). Also a graphical notation of the language (see **Appendix B**.) and the permitted relations of the completed core concepts are provided (see **Appendix C**).

3.2 The Business Process Modeling Notation Language

Business Process and Workflow Modeling

One of the contemporary enterprise modeling approaches is the BPMN language developed by the recently merged Business Process Management Initiative (BPMI) with the Object Management Group (OMG). The Business Process Management Notation (BPMN) [02] is a standard restricted for process modeling for describing the business processes as well as the workflows of an enterprise. In comparison with ArchiMate, the BPMN standard are not covered the application and technology aspects. The core of this language focuses on the processes and workflows [Zang 07] that are associated with the 'businesses' to be modeled and has been designed to provide a uniform notation for modeling business processes in terms of activities and their relationships. The BPMN standard provides also a mapping between the graphical notation of BPMN and the underlying constructs of execution languages, particularly Web Services for Business Process Execution Language (WS-BPEL) and XML Process Definition Language (XPDL). Furthermore, the Business Process Definition Metamodel (BPDM) has been (partly) defined to provide a formal underpinning for BPMN.

3.2.1 A Language for Modeling the Business Architecture

BPMN is a modeling language for describing the business architecture (*scalability principle*) by providing visualization techniques. The primary goal of the BPMN language is to provide a common understanding and generate easily understandable models to business users from business analytics till technical developers [Whit 04a], [Chin 12]. BPMN is designed to model the business architecture of enterprises and provides a set of graphical notations and data structures that is formally and expressively sufficient for its end users. Besides, the BPMN language is aimed to create a mechanism for drawing simple business models by using a limited number of concepts (*simplicity principle*), while at the same time not losing the complexity of the business. BPMN defines a Business Process Diagram (BPD), based on a flowcharting technique, which is aimed for creating graphical models with respect to business process operations. A Business Process Model (BPM) is denoted as a composition of *objects*, which are in essence activities (i.e., work), and *flow controls* that defines the relationship between objects and their order of performance. To span the bridge to process implementation, BPMN contains a mechanism that generates executable environments in terms of BPELs and XLPD's for automated purposes. From this, BPMN provides a way to create a connection between business process design and process implementation (i.e. achieving the principle of *seamlessness* and *reversibility*).

3.2.2 Business Process Definition Metamodel

The BPDM is a standard definition of concepts used to express business process models (a metamodel), adopted by the OMG. Metamodels define concepts, relationships, and semantics for exchange of user models between different modeling tools. The exchange format is defined by XSD (XML Schema) and XMI (XML for Metadata Interchange), a specification for transformation of OMG metamodels to XML. Pursuant to the OMG's policies, the metamodel is the result of an open process involving submissions by member organizations, following a Request for Proposal (RFP). BPDM provides abstract concepts as the basis for consistent interpretation of specialized concepts used by business process modelers. For example, the ordering of many of the graphical elements in a BPMN diagram is depicted by arrows between those elements, but the specific elements can have a variety of characteristics. For example, all BPMN events have some common characteristics, and a variety of specific events are designated by the type of circle and the icon in the circle. The abstract BPDM concepts ensure implementers of different modeling tools will associate the same characteristics and semantics with the modeling elements to ensure models are interpreted the same way when moved to a different tool. BPDM extends business process modeling beyond the elements defined by BPMN and BPEL to include interactions between otherwise-independent business processes executing in different business units or enterprises (choreography). A choreography can be specified independently of its participants, and used as a requirement for the specification of the orchestration implemented by a participant. BPDM provides for the binding of orchestration to choreography to ensure compatibility. Many current business process models focus on specification of executable business processes that execute within an enterprise (orchestration). For exchange of business process models, BPDM is an alternative to the existing process interchange format XPDL (XML Process Definition Language) from the Workflow Management Coalition (WfMC). The two specifications are similar in that they can be used by process design tools to exchange business process definitions. They are different in that BPDM provides a specification of semantics integrated in a metamodel, and it includes additional modeling capabilities such as choreography. In addition, XPDL has many implementations needed for interchanging BPMN. BPDM implementations are in preparation, including support for BPMN, and translation to XPDL.

3.2.3 Business Process Modeling

The BPMN is the standard to represent in an expressive graphically way the *business processes* of an enterprise. End users may model or describe the business process informally (e.g. business analysts) using a set of graphical notations (*flow objects, connecting objects, swimlanes and artifacts*), which allow users to produce easily models. Thereby, some specific users (e.g. developers, business experts) want to describe the business process in a more formal way (*elements*). To this end, a formal graphical notation should ensure the need to execute a process in a distributed environment like web services. This introduces the technical-oriented part of the BPMN language, which allows process implementation (i.e. execution capabilities to generate machine readable standard language). Well-known machineries are WS-BPEL and XPDL (see **Sect. 3.2.6**). BPMN focuses also on the workflow of an enterprise [Whit 04a] i.e. that is the *flow* in which the work as a set of activities takes place in an organization: the work somehow flow through the process in order to exchange the required or produced work.

3.2.4 Business Process Modeling Notation (BPMN) concepts

BPMN itself only defines a concrete syntax, i.e. a uniform (graphical) notation for business process modeling concepts. A partial overview of the concepts of the BPMN language for the graphical notation is given (see **Appendix E)**. This section describes the relevant basic concepts of the BPMN language that enables the creation of models tailored to business processes (i.e. BPD's). The examples that are given are inspired from [With 04] to clarify the use of the different concepts. The BPMN consists of a number of core concepts that are relevant to model business processes and workflows: Event, Activity, Gateway, Sequence Flow, Message, Flow, Association, Pool, Lane, Data Object, Group and Annotation. Several examples are given to illustrate the use of these concepts. For principal modeling reasons (i.e. enabling *consistency* and *simplicity*), a distinction is made in four basic categories to subdivide the enumerated concepts.

The abstract syntax of BPMN is given, which contains the formal definition of the language in terms of a metamodel, expressed in ORM. The metamodel are built up step wisely to provide more insight for understanding the characteristics of each language constructs. It also aims to clarify the relationships to other language constructs. Modeling is an extensive effort and the result of the desired model depends on the modeler's domain of interests or concerns. Therefore, some important modeling decision needs to be made, while modeling the metamodel.

Modeling Decisions

It is not the intention to provide an entire view of the complete BPMN language concepts, but merely has the intention to only delve into the parts that are of relevant value regarding this thesis research. The following decisions have been made with respect to underlying architecture of the BPMN language:

1. Covering basic aspects of the BPMN specification instead of providing complex models with many details of available concepts. This idea needs to clarify the concepts at a higher abstraction level (conceptual) that restricts the BPMN models. Hereby, relative simple models are given, to illustrate the use of these basic concepts.
2. The focus lies on the basic concepts of the BPMN language, rather than providing abundance of specific concepts. Only similar subsets of concepts are considered to be valuable for comparing the ArchiMate business coverage.
3. The BPMN specification does not provide a metamodel, but it is useful with respect to the modelers' point of view and relating concerns to determine the language metamodel for a well understanding of BPMN models.

Flow Objects
The first category is the flow object, in which the active process occurs. Typically, a flow object* contains the actual work that has to be performed by business entities (e.g. a unit like the organization, or a single person e.g. a doctor or a patient). They are connected to each other through *connecting objects* to indicate a sequence. Flow objects can be partitioning a number of core elements which comprises the following ones:

1. ***Event;***
   An event* affects the flow of the *process* with respect to the business. They have a cause (i.e. trigger), and can graphically modeled with a '*start event*' that starts the process actually. Within the process, it is also possible to have triggers that influence the flow *within* the process, which are affected by the '*immediate event*'. Eventually the outcome of the entire process (i.e. an impact) results in an *'end event'*.

2. ***Activity;***
   An activity* can be atomic or non-atomic (i.e. compound). It consists of two types that is denoted as a '*Task'* and a '*Sub-Process*'.

3. ***Gateway;***
   A gateway* controls the divergence and convergence of *Sequence Flows* (SF). Internal markers that are graphically displayed in the diamond indicate the type of the behaviour control.

The example of **Fig. 3.19** illustrates the use of the flow object concepts concerning about the handling of a payment process in order to provide a package to a customer. A 'Start Event' (e.g. a customer wants to pursue a good) triggers the initial process to start the first Activity that is a Task ('Identify Payment Method'). The relationship between theses flow objects are explained at a later section (see ***Connection Objects***). After determination of the payment method, using a Gateway (e.g. a "decision" must be taken) that forked the incoming flow, one of the two Activities is treated ('Check or Cash' or 'Credit Card'). The two branched Activities, which are also of type Task ('Accept Cash' or 'Check',) and the Activity ('Process Credit Card'), are merging together by the next Activity ('Prepare Package for Customer'). At this moment it is clear that the desired package is prepared after payment is done by the customer, resulting in an 'End Event' that indicates the end of the payment process.

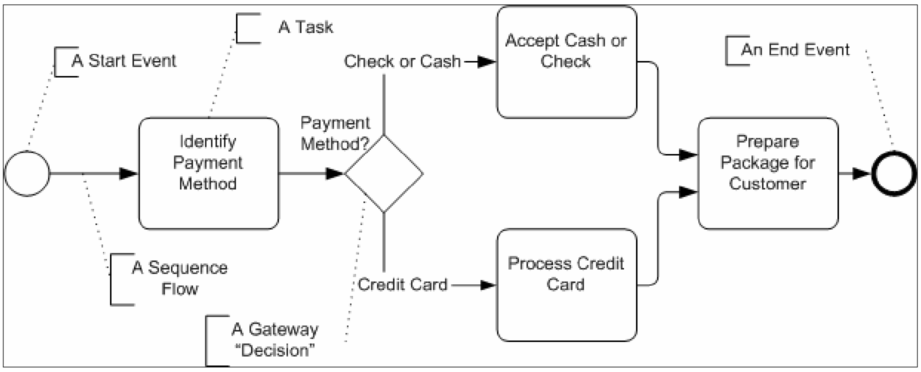


Fig. 3.19. An example of a business process model using flow objects.

An abstract model is given (see **Fig. 3.27**) that represents a part of the entire BPMN language metamodel and defines the flow objects and their relationships. A *flow object* is subdivided in three elements, which is of type *gateway, event*, and *activity*. Gateways are specialized in *data-based gateway* or *event-based gateways*. An event can be of type: *start event, intermediate event* or *end event*. Furthermore, an activity can be either way an *atomic activity* (i.e. a task) or part of a set of grouped non-atomic activities, i.e. *compound activity* comprising multiple tasks.

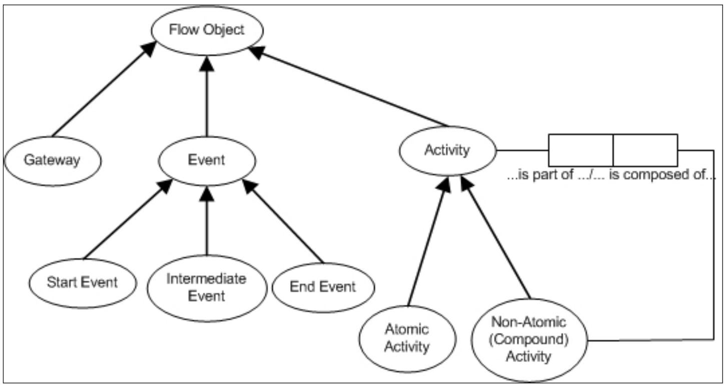


Fig. 3.27. The structure of flow objects.

Connecting Objects

The second category is called the connecting object*, which represents the connection between flow objects. These connecting objects are divided into:

1. ***Sequence flow;***
   A sequence flow* shows the order of activities to be performed in a *process*. Sequence flows may cross the boundaries of *lanes* within a *pool*.

2. ***Message flow;***
   A message flow* shows the flow of messages between two separate process *participants*, which can be business entities or business roles. A *participant* is represented as a pool separately, which constitute a process. It is not permitted that message flows are being used between flow objects in lanes of the same pool.

3. ***Association;***
   An association* binds data, text and artifacts with flow objects. It shows the inputs and outputs of the activities.

The example of **Fig. 3.20** illustrates the use of the flow objects and connecting objects in a more advanced way with details. The depicted process, which comprises a part of the entire business process, concerning about the handling of quotes if suppliers are involved in order to find an optimal quote. A sequence flow connects the input of a Gateway ('Any Suppliers?'); if any suppliers are involved in this issue, then the subprocess activity ('Repeat for Each Supplier') has to be performed. This subprocess includes multiple activities ('Send RFQ', 'Receive Quote' and 'Add Quote'). An additional feature is the initial marker, denoted in the bottom of the center rectangle with a black arrow circling around: the subprocess may be iteratively performed depends on the number of suppliers. An intermediate event ('*time event*') displays a time limit that is given for a certain time of period. When the time exceeds, the sequence flow ('Time Limit Exceed') goes to the next activity. Then, after the subprocess is performed, the next Activity ('Find optimal Quote') can be determined leaving with a sequence flow.

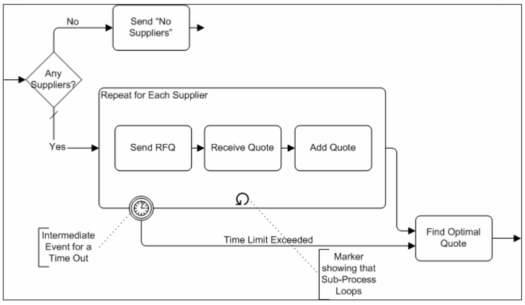


Fig. 3.20. A business process using advanced constructs.

The metamodel of connecting objects (see **Fig. 3.28**), defines the associated elements that describes the flow, which is either a *sequence flow* or a *message flow* and the *association*. Subtypes of supertypes can inherent (behaviour) properties of their supertypes, but constraints imposed on subtypes can be exclusive applicable to the subtype itself.

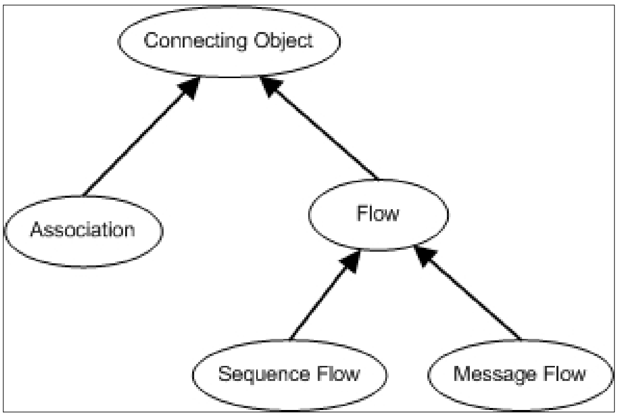


Fig. 3.28. The structure of a connecting object component.

Swimlanes
The third main category is swimlanes*, which is used to categorize and organize activities into *pools* and *lanes* to illustrate different capabilities or responsibilities associated with participants in the process. Swimlanes are divided into two different types:

1. ***Pools***
   A pool* represents one or more participants in terms of comprising multiple lanes in a process. Thus, a pool acts as a container for partitioning a set of activities from other pools.

2. ***Lanes***
   A lane* represents also a participant in a process but is part of a pool. So, a lane can be seen as a subpartition within a pool. Lanes can extend the length of the pool either vertically or horizontally. Lanes are often used to separate the activities associated with a specific company function or role.

The example of **Fig. 3.21** illustrates the use of swimlane concepts discerning participants inherent to activities in which they are responsible for or capable of doing so. Two pools, representing participants ('Doctor's Office' and 'Patient') are involved in this business process, where they interact with each other through the connection object *message flow*. The message flow is depicted as a dashed line with an open arrowhead. Typically, a message flow may not cross the boundaries of a lane between objects *within* a pool.

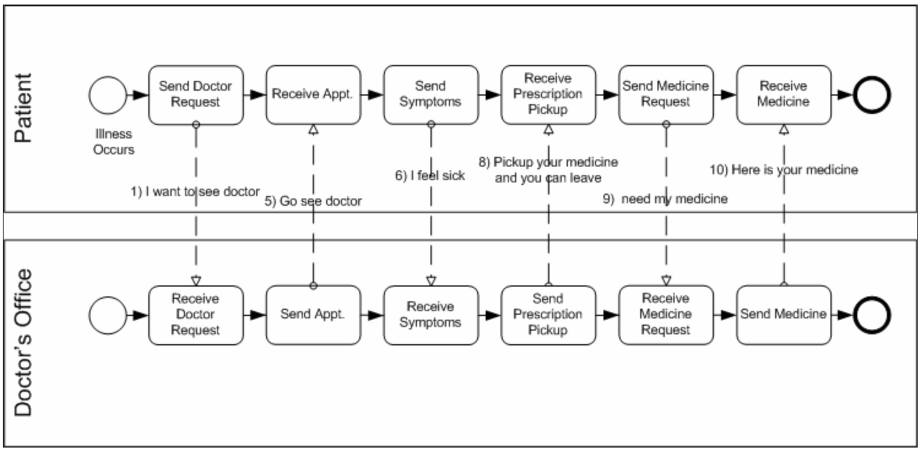


Fig. 3.21. An example of a business process illustrates the use of pools.

A patient becomes ill (i.e. the trigger) and request a doctor (Activity: ‘Send Doctor Request’). The patient wants to see a doctor (Message Flow: 1. ‘want to see a doctor’). The doctor’s office receives the request for a doctor from the patient (Activity: ‘Receive Doctor Request’). Subsequently, the doctor’s office makes an appointment and sends it to the patient (Activity: ‘Send Appt.’) with the message that the patient sees the doctor at the agreed appointment (Message Flow: 5. ‘Go see doctor’). The patient then receives the appointment coming from the doctor’s office (Activity: ‘Receive Appt.’). Now, the patient states which symptoms are shown by his complaints (Activity: ‘Send Symptoms’) and pointed out to be sick (Message Flow: 6. ‘I feel sick’). At the doctor’s office the complaint is received (Activity: ‘Receive Symptoms’) and a prescription is created, which the doctor’s office provides it to the patient (Activity: ‘Send Prescription Pickup’). Next, the patient is told that the medicine can be picked up (Message Flow‘: 8. ‘Pickup your medicine and you can leave’). Afterwards, the patient receives the prescription at the doctor’s office (Activity: ‘Receive Prescription Pickup’) and ask for the prescribed medicine (Activity: ‘Send Medicine Request’): the patient wait until the medicine is given (Message Flow: 9. ‘Need my medicine’). In turn, the doctor’s office receives the patient’s need for a medicine (Activity: ‘Receive Medicine Request’) and provided the patient (Activity: ‘Send Medicine’) with a medicine (Message Flow: 10. ‘Here is your medicine’). Eventually, the patient receives the prescribed medicine (Activity: ‘Receive Medicine’) and use it.

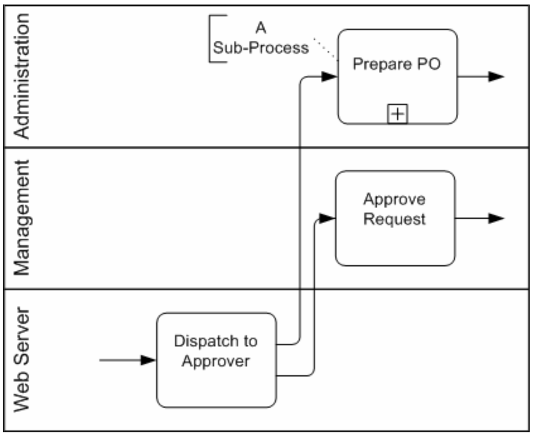


Fig. 3.22. Modeling processes with lanes concept.

The example given in **Fig. 3.22** illustrates the use of *lanes* that are part of a specific pool to clarify the responsibility of multiple business *roles* or -*functions* e.g. the specific departments of the enterprise ('Web Server', Management' and 'Administration'). Lanes are used for the distribution of activities over these roles/functions. In this case, a web server performs its activity (Activity: 'Dispatch to Approver') that is related to the management and administration part of the enterprise's division for the approval. Next, the management department needs to approve the request (Activity: 'Approve Request'). At the same time, the administration department then, prepares the associated work to dispatch personal & organization issues (Subprocess: 'Prepare PO'). With regard to flows, in particular *sequence flows*; the above example shows that these types of flows may cross the boundaries of lanes within a pool.

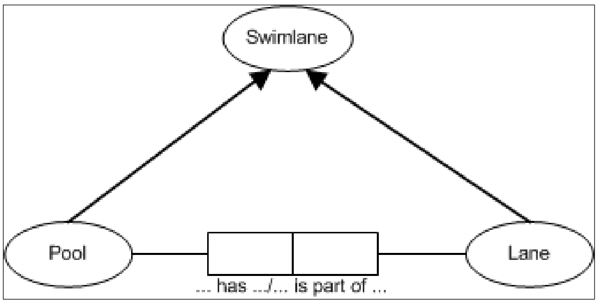


Fig. 3.29. Subtyping of the swimlane component.

Artifact

The last main category is the artifact, which composed of the following elements:

1. ***Data Object;***
   A data object* aims to represent data that is required or produced by activities. Typically, data objects are connected to activities through *associations*.
2. ***Group;***
   A group* is graphically represented as a rounded corner rectangle with a dashed line. The grouping mechanism is used for documentation or analysis purposes, while at the same time not affecting the *sequence flow*. It aggregates elements that shares common properties.
3. ***Annotation;***
   An annotation* aims to provide the modeler additional text for communicative purposes. It is supposed for the '*business reader*' that enhanced the *readability* of the BPMN diagram.

To illustrate the use of artifacts concepts, the previous example is expended and can be graphically represented (see **Fig. 3.23**). Data objects are graphically represented as a rectangle with a folded corner (Data Object: 'Purchase Info' and 'Data Object: 'Approval Request Email'). They are even produced or required by the associated activity: the webserver produced the data objects that both the management and the administration require for performing their activities appropriately. The grouping mechanism aggregates flow objects to enable analysis on the grouping part or providing comments. In this example, the group concept is used to provide the end user with additional text that the grouped activities can be performed simultaneously. The *annotation*, represented graphically with a dashed line attached to a black open parenthesis including additional text, is intended for the *business user* of the BPD that aims for communication purposes: to support the readability of the business process models.

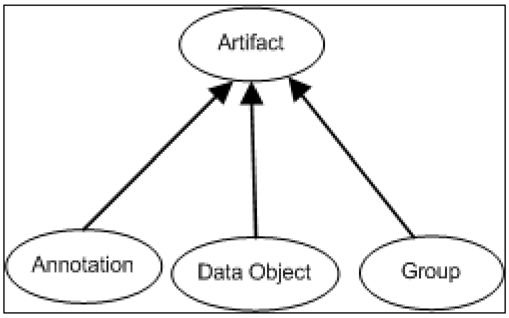


Fig. 3.30. Subtype of the artifact component.

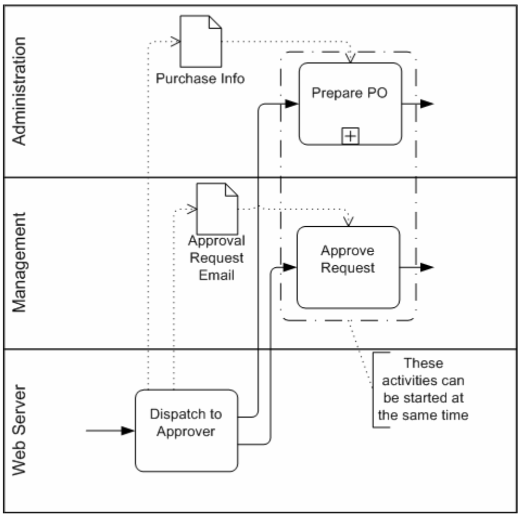


Fig. 3.23. Modeling artifacts.

Business Process Levels

Sometimes, it is useful to only provide a certain level of detail. Two types of levels can be discerned: a business process can be either way (see **Fig. 3.24**) modeled at a *high-level*, showing only relevant parts (subprocesses) and their relations without any detail, and the *low-level* business process providing details of the performed activities (see **Fig. 3.25**). If the business user wants to see an overview of compound activities performed in a process, high-level models are more suitable. If they require inter-relations within subactivities or its relationship with high-level processes (i.e. internal behaviour of business processes), business processes at the low-level are more preferred.

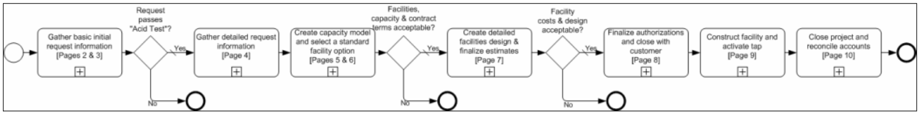

Gather basic initial request information [Pages 2 & 3]
Request passes "Acid Test"?
Yes
No
Gather detailed request information [Page 4]
Create capacity model and select a standard facility option [Pages 5 & 6]
Facilities, capacity & contract terms acceptable?
Yes
No
Create detailed facilities design & finalize estimates [Page 7]
Facility costs & design acceptable?
Yes
No
Finalize authorizations and close with customer [Page 8]
Construct facility and activate tap [Page 9]
Close project and reconcile accounts [Page 10]

Fig. 3.24. An example of a high-level business process.

When zooming in on the subprocesses at the high-level (see **Fig. 3.24**), several lanes from another pool are involved and are connected through associations (see **Fig. 3.25**).

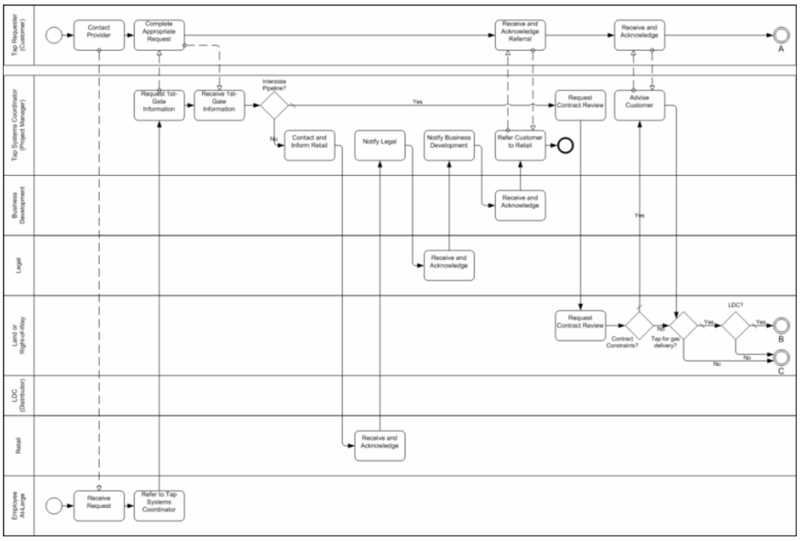


Fig. 3.25. An example of a low-level business process.

One of the main goals of BPMN is to connect the gap between business process design and process implementation. Process implementation executes the business processes define in the BPD's: it concerns about the ability to enable the generation of executable BPEL by using a BPMN Diagram.

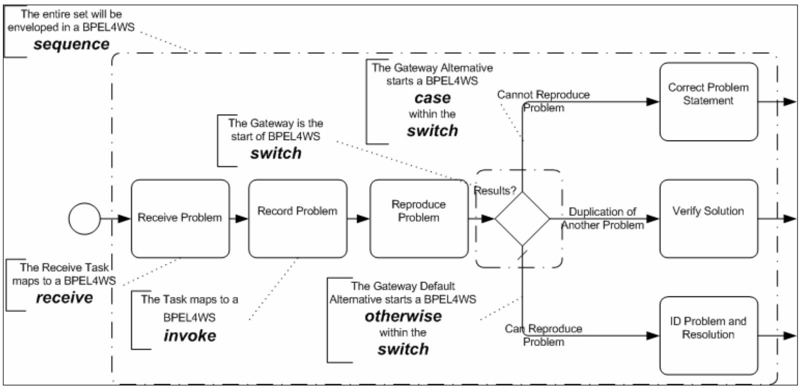

The entire set will be enveloped in a BPEL4WS
sequence
The Gateway Alternative starts a BPEL4WS
case
within the
switch
Cannot Reproduce Problem
Correct Problem Statement
The Gateway is the start of BPEL4WS
switch
Results?
Receive Problem
Record Problem
Reproduce Problem
Duplication of Another Problem
Verify Solution
The Receive Task maps to a BPEL4WS
receive
The Task maps to a BPEL4WS
invoke
The Gateway Default Alternative starts a BPEL4WS
otherwise
within the
switch
Can Reproduce Problem
ID Problem and Resolution

Fig. 3.26. BPMN concepts and their related WSBPEL concepts.

The execution language Web Services for Business Process Execution Language (WSBPEL) is explained in detail (see **Sect. 3.2.5**). The example given (see **Fig. 3.26**) illustrates the correlation between BPMN concepts and WSBPEL concepts. Table 3.1 summarizes the BPMN concepts used in the example for mapping to the technical-oriented execution language concepts. The opportunity to map concepts from BPMN Diagrams to BPEL is designed to support alignment between business and IT: this approach translates the

BPD's into an executable environment in order to implement the processes within a business process management system (BPMS).

Table 3.1. Concept mapping from BPMN Diagrams to WSBPEL based on Fig. 3.26.

BPMN concepts	**WSBPEL concepts**
Group (entire set)	Sequence
Receive Task	Receive
Task	Invoke
Gateway	Switch
Gateway (alternative)	Switch / Case
Gateway (default alternative)	Switch / Otherwise

Some parts of the above constructed metamodel needs further explanations (see **Appendix D.1**). A trigger has many types, which are the enumerated triggers: Message Trigger, Time Trigger, Rule Trigger, Link Trigger, Error Trigger, Cancel Trigger, and Compensation Trigger. These triggers may influence the behaviour of flow objects, in particularly the activities. An intermediate event may affect the associated activity. Commonly, a trigger is something that happens suddenly or expected by its environment. The gateway has also many types that consist of a XOR gateway, OR gateway, AND gateway, and a Complex gateway. Finally, XOR gateways are divided in event-based gateways and data-based gateways.

3.2.5 Execution Languages (WS-BPEL, XPDL)

One of the underlying architecture of the BPMN language is the execution language by Web Services Business Process Execution Language (WS-BPEL), which supports business process modeling to enable implementation of business processes. From this point, a mapping from BPMN to WS-BPEL then can be performed [Whit 05a], [Whit 05b]. Mainly, the BPMN specification v2.0 used the block structure (the *sequence* element), while it is possible to persist the graph structure (the *flow* element).

Web Services

Web services interactions can be discerned in two types:

1. ***Executable business processes***; executable business processes model actual behavior of a participant in a business interaction. It serves a executable role.

2. ***Abstract business processes***; abstract business processes are partially specified processes that are not intended to be executed. An abstract process may hide operational details and fulfills a descriptive role.

WS-BPEL is intended to model the behavior of both executable and abstract processes: it allows specifying the executable and abstract business processes. In line with this, WS-BPEL enables the support of business transactions and defines a model including a grammar for describing the behaviour of the business process. In addition, WS-BPEL provides a mechanism for handling business exceptions and process faults. It also contains a mechanism to define how activities (either atomic or compound) have to be compensated in exceptional cases or when a reversal is required. Related to the earlier mentioned fact of deploying BPMN (**Sect. 3.2.3**), WS-BPEL has been chosen by the BPMI to be a preferred serialization format for BPMN diagrams. Thus, WS-BPEL is considered to be the best suitable one as execution language for BPMN diagrams.

Another extension of the BPMN standard is the XML Process Definition Language (XPDL). XPDL is suitable for interchanging business process definitions between workflow products, i.e. between different modeling tools and Business Process Management Systems (BPMS). XPDL defines an XML schema for specifying the declarative part of workflow and business process. This machinery language is designed to exchange the process definition, both the graphics and the semantics of a workflow and business process. Currently, XPDL seems to be the most suitable file format for exchange of BPMN diagrams; due to the fact that it has been designed specifically to store all aspects of a BPMN diagram. Thus, XPDL is particularly used to retain graphical aspects, such as the X- and Y- coordination of the objects, as well as the executable aspects that can be used to run a process. The differences between XPDL and WS-BPEL could be explained as follows: WS-BPEL does not contain elements to represent the graphical aspects of a process diagram, whereas XPDL contains elements to represent botch the graphical and the executable aspects. This distinguishes XPDL from BPEL which focuses exclusively on the executable aspects of the process. From this, it can be assumed that XPDL is the XML Serialization of BPMN. Depending on the enterprises' goal, one might deploy BPMN in several different purposes. For descriptive purposes only, BPMN itself is likely to be useful. When simulation purposes are considered, BPMN could be used in combination with XPDL, while for execution purposes WS-BPEL seems be fruitful to translate BPMN diagrams into directly executable code.

3.2.6 Summary of the BPMN Architecture

The Business Process Management Initiative (BPMI), part of the Object Management Group (OMG), developed a modeling language, the BPMN standard, which specifies a graphical notation aimed to provide a common basis for business process modeling and execution languages. BPMN focused exclusively on the *processes* and *workflows* at the business level that are the domain of interest. The main purpose is to provide a uniform notation for modeling business process in terms of activities and their relationships. The Business Process Definition Metamodel (BPDM) defines a standard definition of concepts used to express business process models. Since the metamodel can be derived from the BPMN standard, the abstract syntax of the BPMN language is given in terms of a metamodel. Finally, the result of the complete metamodel of the language, described in ORM, is shown (see **Appendix D**). The BPMN metamodel serves as a starting point, where all core concepts can be abstracted from. The underlying architecture of BPMN provides a way to create a bridge between business process design and process implementation, in which the execution language WS-BPEL and XPDL plays a central role.

3.3 The Petri Net Language

Petri Nets

The International Standard provides a well-defined semi-graphical technique for the specification, design and analysis of systems. The technique, High-level Petri Nets (HLPN) is mathematically defined [01], and may be used to provide unambiguous specifications and descriptions of applications. It is also an executable technique, allowing specification prototypes to be developed to test ideas at the earliest phase of system development. Specifications written in the technique may be subjected to analysis methods to prove properties about the specifications, before implementation begins, thus saving on testing and maintenance time. Petri Nets can be supported by the Petri Net Markup Language (PNML), which is a standard, an XML-based interchange format for Petri nets [Webe 03], [Bill 03].

3.3.1 Petri Nets

Petri Nets has been proven to be a useful technique for verification purposes e.g. for ensuring correctness of configurable process models by presenting a novel verification approach used for partner synthesis, which seems to be very complex [Aals 11]. But also for analysis purposes [Aloi 12], that shows how Colored Petri Nets (CPNs) can be used to model risk factors in ERP projects in order to deal with the problem of interdependence in risk assessment. These Petri net based approaches, allows practitioners to perform different kind of analysis on systems. Fields of applications where Petri net based approaches can be deployed for:

- ✓ requirements analysis;
- ✓ development of specifications, designs and test suites;
- ✓ descriptions of existing systems prior to re-engineering;
- ✓ modeling business and software processes;
- ✓ providing the semantics for descriptive (non-formal) languages;
- ✓ simulation of systems to increase confidence;
- ✓ formal analysis of the behaviour of critical systems;
- ✓ development of Petri net support tools.

As there are a variety of discrete event and distributed systems in the mentioned generic fields of application of Petri nets, it is obviously that the underlying language architecture provides the formal underpinning to achieve this.

3.3.2 High-Level Petri Net Graph (HLPNG) Concepts

High-level Petri Nets - (HLPN) - are represented in a graphical form, which allows visualization of system dynamics e.g. flows of data and control. This approach is taken, that is most appropriate for industrial use. The graphical form is referred to as a High-level Petri Net Graph (HLPNG). It provides a graphical notation for places and transitions and their relationships, and determines the inscriptions of the graphical elements.

High-level Petri Net Graph Components

A High-level Petri Net Graph (HLPNG) comprises the following number of components:

- ✓ *A Net Graph*, which consists of sets of nodes of two different kinds: The first kind are *places* and *transitions* and the second kind are *arcs* that connects places to transitions and transitions to places.
- ✓ *Place Types*, which are non-empty sets. One type is associated with each place.
- ✓ *Place Marking*, which is a collection of elements (data items), chosen from the place's type and associated with the place. It is allowed to repeat items. A collection of items that allows repetitions is known in mathematics as a multiset. The items associated with places are called *tokens*.
- ✓ *Arc Annotations*, whereby arcs are inscribed with expressions that may comprise constants, typed variables (e.g., $x < y$) and function images (e.g. $f(x)$). The expressions are evaluated by assigning values to each of the variables. When an arc's expression is evaluated, it must result in a collection of items taken from the type of the arc's place. The collection may have repetitions.
- ✓ *Transition Condition*, which is a Boolean expression (e.g., $x < y$) that inscribes a transition.
- ✓ *Declarations*, which comprises definitions of place types, typing of variables, and definitions of functions.

Net execution

HLPNGs are executable in terms of visualizing the flow of tokens in the Petri net. To this end, HLPNGs are used to illustrate *the flow of control* and *the flow of data* within the same model. Both *enabling and the occurrence of transitions* controls the execution (see **Glossary** '*transitions occurrence*' and '*enabling a transition*').

Enabling

The enabling of transitions are closely related to the set of all place markings of the HLPN. Transitions are enabled in a particular transition mode, which means that values are assigned to the transition variables that satisfies the transition condition. Variables that occur in the expressions associated with the transition are called transition's variables. These transition's variables are the transition condition and the annotations of arcs that involve the transition. To enable a transition, it's input places must be marked. An input place of a transition is a place connected by an arc, that connect that place with the transition. This connecting arc is called an input arc of the transition. A transition is enabled in a specific transition mode, for a particular set of all place markings of the HLPN. When an input arc expression is evaluated, this leads to a multiset of tokens. The tokens originate from a transition mode are of the same type as the tokens in the input place. Transitions are enabled in a specific mode, if the multiset of tokens of input place's marking have at least the multiset of tokens of the input arc, i.e. the input arc's enabling tokens. Transition modes are concurrently enabled if each input place's marking have at least the sum of each of its input arc multiset of tokens that are linked to the input place's. An example is given (see **Fig. 3.32**) to illustrate the enabling of transitions.

Transition Rules for a Single Transition Mode

When an enabled transition occurs, tokens are removed from its input places, and tokens are added to its *output places*. An output place of a transition is a place which is connected to the transition by an arc directed from the transition to the associated place. An arc that leads from a transition to a place (an output place of the transition) is called an *output arc* of the transition. If a transition is enabled in a mode, it may occur in that mode. On the occurrence of the transition in a specific mode, the following actions occur:

1. For each input place of the transition the following rule applies:
   - ✓ The enabling tokens of the input arc with respect to that mode are subtracted from the input place's marking,
2. For each output place of the transition the rule applies:
   - ✓ The multiset of tokens is added to the marking of the output place. The added multiset of tokens results from the evaluation of the output arc expression for the mode. It is possible that a place fulfills both an input place and at the same time an output place of the same transition.

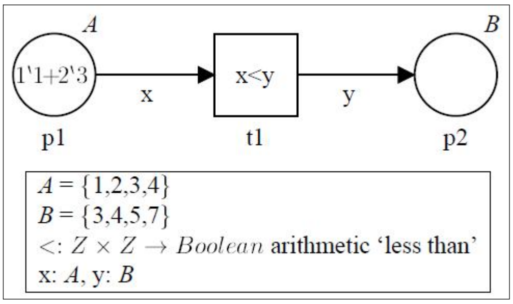


Fig. 3.32. HLPN graph with transition conditions.

The graphical representation of the net graph is illustrated by the simple example of a HLPNG (see **Fig. 3.32**). This example comprises two places, named $p1$ and $p2$ one transition called $t1$, and two arcs from $\underline{p1\ to\ t1}$ and from $\underline{t1\ to\ p2}$. The places are represented by circles, the transition by a square, and the arcs by arrows.

The declarations define two types A and B that are different subsets of positive integers. Variable x is of type A whereas variable y is of type B. The transition is inscribed with the Boolean expression $x < y$, where the *less than* operator is defined in the declarations.

Arc $(p1, t1)$ is annotated with the variable x, while arc $(t1, p2)$ is annotated with y. Place $p1$ is typed by A and has an initial marking $M_0(p1)$ = 1′1+2′3. This states that associated with the place $p1$ are the value 1 (one) and two instances of the value 3 (three). Place $p2$ is typed by B, and is empty representing the empty multiset $M_0(p2) = \emptyset$.

A convention adopted in high-level nets, is that arcs may be annotated by variables or constants of the same type as the arc's place. The evaluation of the arc expression is thus an element of the place's type. By convention, this is interpreted as the corresponding singleton multiset over the place's type. For example, for x bound to 1, and y bound to 3, the value associated with arc $(p1, t1)$ is 1, which is interpreted to mean the multiset 1′1. Similar to the value associated with arc $(t1, p2)$ is 3 which are interpreted to mean the multiset 1′3. Where there is any possibility of ambiguity, the multiset conversion operator (′) should be used. For example, more formally, the annotation on the arc $(p1, t1)$ should be 1′x instead of x.

In the initial marking, t1 can be enabled in the following modes:
{(1,3),(1,4), (1,5), (1,7), (3,4), (3,5), (3,7)}

The first element of each pair represents a binding for x, while the second represents a binding for y, which satisfies the transition condition $x < y$. It can be seen that the multiset of modes, 1′(1,3) + 2′(3,5), is concurrently enabled. Another example of the concurrent enabling of modes is the multiset 1′(1,5) + 1′(3,4) and yet another is 1′(1,7) + 1′(3,5) + 1′(3,7).

If transition $t1$ occurred in mode 1′(3,5) then the resultant marking would be:
$M(p1)$ = 1′1+1′3
$M(p2)$ = 1′5

Alternatively, if the multiset of modes 1′(1,3) + 2′(3,5) occurred concurrently, the resultant marking would be:
$M(p1)$ = $\emptyset$
$M(p2)$ = 1′3 + 2′5.

3.3.3 High-Level Petri Net Graph (HLPNG) Syntax

As Petri Nets has many (formal) notations, this section describes the graphical form which comprises two parts: a *Graph*, which represents the net elements graphically with textual inscriptions, and a *Declaration*, defining all the types, variables, constants and functions that are used to annotate the graph part. The declaration may also include the initial marking and the typing function if these cannot be inscribed on the graph part, because of the limited space. There needs to be a visual association between an inscription and the net element to which it belongs.

Places
Places are graphically represented by ellipses or might often use by circles.
A place has the following associated annotations:

- ✓ the place name;
- ✓ the name of the type associated with the place$\left(Type(p)\right)$;
- ✓ the initial marking $M_0(p)$.

A mechanism is needed to unambiguous regarding the association of these annotations with the correct place. There are no strict rules for positioning the annotations with regard to places. To give an example, the initial marking might be shown inside the ellipse/circle and its name and type outside or vice versa. If the initial marking is empty, then it might be leaving out.

Transitions
A transition is graphically represented by a rectangle and is annotated by a name and an expression that is of Boolean type; the *Transition Condition*. If the Transition Condition turns to be true, i.e. (*TC* (*t*) = *true*), it can be leave out. An example of a transition can be given as follows;

$\boxed{x < y}$

$t1$

It represents a transition with a name appointed as $t1$, and a transition condition, $x < y$ (inside the rectangle), where the variables x and y and the operator $<$ (less than) are defined in the declaration part. Also a mechanism is required that prevents ambiguity with respect to the association of these annotations with the correct transition. The position of the annotations with respect to transitions is not mandated: e.g., the transition condition might be shown inside the rectangle and its name outside or vice versa.

Arcs
An arc is graphically represented by an arrow. For $(p, t) \in F$ an arrow is drawn from place p to transition t and conversely, $(t, p) \in F$, an arrow is drawn from transition t to place p. It may possible to represent a single arc with an arrowhead at both ends including an annotation by a single inscription, when the following situation occurs:

If (p, t) and (t, p) has the same notation, then the formula holds the equation A(p, t) = A(t, p). Arcs are annotated with terms for which the notation is not mandated by the International Standard. However, usual mathematical conventions might be determined if desirable.

Markings & Tokens
A token is a member of $\bigcup_p \in P\ Type(p)$. A marking of the net may be shown graphically by annotating a place with its multiset of tokens $M(p)$ using the symbolic sum representation.

3.3.4 High-Level Petri Net Graphs Examples

This section provides some basic key examples to explain the use of HLPNG's.

Net Graphs
In HLPNGs, an action is modeled by a *transition*, which is graphically represented by a rectangle. A collection is modeled by a *place*, which is graphically represented by a circle or an ellipse. Places and transitions are called the *nodes* of a *net graph*. Arrows, called *arcs*, show which places a transition operates on. Each arc connects a place and a transition in one direction. Arcs never connect a place with a place nor a transition with a transition. The graphical representations of components of a net graph can be illustrated as follows (see **Fig. 3.32**):

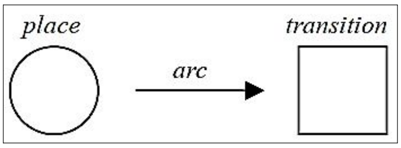


Fig.3.32. Components of a net graph.

Place & Tokens

The objects of the system are modeled by (arbitrarily complex) data items called *tokens*. Tokens reside in places. The contents (i.e. the tokens) of a place are called the *marking* of the place. The tokens form a collection (known in mathematics as a multiset), i.e., several instances of the same token can reside in the place. A *marking* of a net consists of the markings of each place. A net graph (see **Fig. 3.33**) consists of a single place, *AlicesPurse*, which models that Alice's purse contains two 1 cent, three 10 cent and two 50 cent coins. The set of coins is defined in a textual part of the HLPNG called the Declarations. The place, *AlicesPurse*, is typed by the set, *Coins*. This means that only coins (belonging to *Coins*) can reside in Alice's purse. In this example, the tokens correspond to coins.

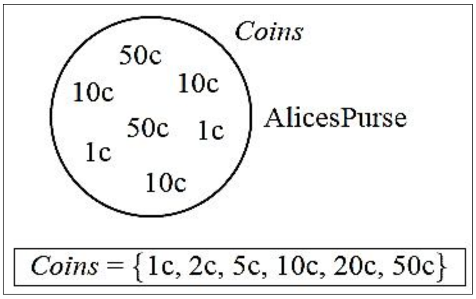


Fig.3.33. Example of a graph net to illustrate tokens which representing coins.

The net graph (see **Fig. 3.33**) has no transitions and no arcs. As no actions are modeled, nothing ever happens and nothing ever changes in this system. When a particular instance of HLPNG is defined, each place is defined with a special marking, called the *initial marking*, because other markings will usually evolve, once a net is executed. As a place can be marked with a large number of tokens, the initial marking may be declared textually instead of pictorially. Thus, Alice's present coin collection can be written as the initial marking, M_0(*AlicesPurse*) = 2′1c + 3′10c + 2′50c and the net graph is then drawn without tokens. The number of each of the coins in the purse is separated from the value of the coin by a back prime (′) to avoid any confusion. To address this issue, there may be both a 5c and a 25c coin. In order to distinguish these different coins respectively two 5c coins and a 25c coin, a suitable separator is needed; otherwise the notation 25c can be ambiguously interpreted.

Transitions

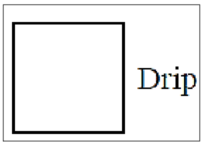


Fig.3.34. An example of a simple transition.

Modeling transitions can be simple graphically represented as shown in **Fig. 3.34**. It models the dripping of a tap. Transition Drip can always happen, any number of times. This simple transition example is considered to be a net, even though no arcs and places are involved in this model.

Arcs

An arc from a place to a transition indicates that this transition consumes objects from the place. An arc in the opposite direction indicates that this transition produces tokens on the place. The following example is given (see **Fig. 3.35**), which models the spending's behaviour of Alice purse. Alice purse comprising a collection of coins: one ten cent and two fifty cent coins. In this example Alice are allowed to spend any number of coins at a time. *Arc annotations* determine the kinds and numbers of tokens that are produced or consumed. Here, the annotation x indicates that any coin (from Alice's purse) can be spent. However, it has to be declared in the textual part of *Example C* that x denotes a variable for coins. Alice could spend: a ten cent coin; a fifty cent coin; a ten cent and a fifty cent coin; two fifty cent coins; and all her coins in one transaction that is by the occurrence of transition spend.

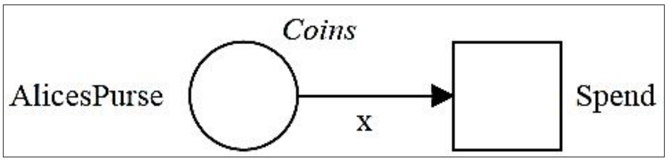


Fig.3.35. Example of the function of an arc.

Coins = {1c, 2c, 5c, 10c, 50c}
x: *Coins*
M_0 $(AlicesPurce)$ = ∅

Net Graphs

Mathematical descriptions of a net graph are often used to provide ambiguity. Graphically, the size and position of the nodes, as well as the size and shape of the arcs, that together forms the *net graph*, contributes to the readability of net graphs. These aspects are considered irrelevant to mathematical descriptions of nets. Informally, the net has one place, called *AlicesPurse*, one transition called *Spend*, and one arc starting from the place

AlicesPurse to the transition *Spend*. Thus, formally the model can be expressed as:

P = {AlicesPurse}
T = {Spend}
F = {(AlicesPurse, Spend)}

The International Standard uses the abbreviations P, T and F, where P represents the set of places, T denotes the set of transitions, and F denotes the set of arcs. Typically, each arc is described as the pair consisting of its node where it starts flowing from *AlicesPurse* and its node where it ends flowing to the transition called *Spend*.

Transition Conditions

The graph net (see **Fig. 3.36**) concerns about Bob receiving coins, which he collects in his purse. The model shows that Bob purse starts with an empty purse and subsequently collects 10 cent coins. It is not relevant from where the coins may come from as this is not been modeled. It only shows what happens to Bob's purse as a consequence of an arbitrary number of occurrences of *Receive*.

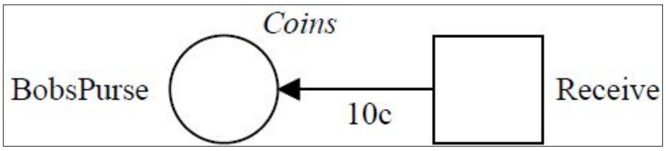


Fig.3.36. A graph net illustrates a transition condition.

Coins = {1c, 2c, 5c, 10c, 50c}
$M_0\ (BobsPurse)$ = ∅

In the next example (see **Fig. 3.37**), an expanded version of the previous example, Alice becomes the initiator of giving Bob any of her coins from her purse. However as a transition condition, Bob accepts only 10c coins from Alice to collect in his purse.

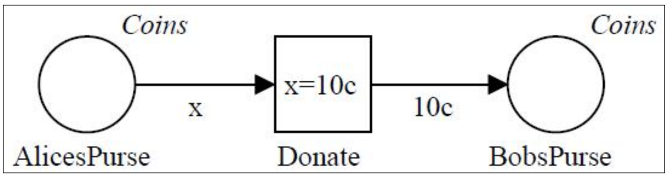


Fig.3.37. Expanded graph net illustrating the transition condition.

Coins = {1c, 2c, 5c, 10c, 50c}
x : *Coins*
M_0 $(AlicePurse)$ = 1'10c + 2'50c
M_0 $(BobsPurse)$ = $\emptyset$

A *transition condition* has been added, requiring that x = 10c. The transition *Donate* takes place when the condition is satisfied. If there are no appropriate tokens (i.e., 10c) in *AlicesPurse*, then *Donate* cannot be occurred. In order to prevent that Bob (see **Fig. 3.37**) can collect coins infinitely to his purse, some modification is made. This results to the updated graph net (see **Fig. 3.38**) that restricts the number of 10 cent coins in such a way that Bob can receive 200c maximal by the transition condition 'n < 200'.

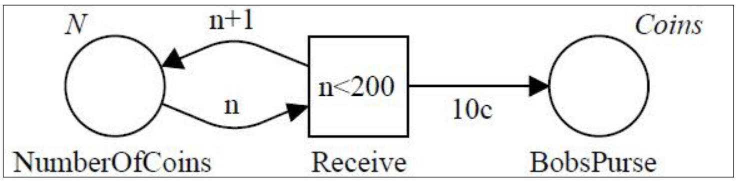


Fig. 3.38. Transition condition defines restrictions of places.

ℕ set of natural numbers
Coins = {1c, 2c, 5c, 10c, 50c}
n : ℕ
<: *N* × *N* → *Boolean (normally used as 'less than')*
+ : *N* × *N* → *N* is used as an arithmetic addition
M_0 $(BobsPurse)$ = $\emptyset$
M_0 $(NumberOfCoins)$ = 1'0

Two companies respectively *Company A* and C*ompany B* are located at different places. *Company A* distributes big crates to C*ompany B* one by one and all of them have the same size. C*ompany B* stores the crates in a store room. From the store room, crates are used for processing e.g., distributed to consumers. *B's* C*ompany* store room has a certain storage capacity of crates denoted as MAX. To prevent the store room to be overcrowded, C*ompany B* concludes an agreement with *Company A* on a protocol that *controls the flow* (see **Fig. 3.39**).

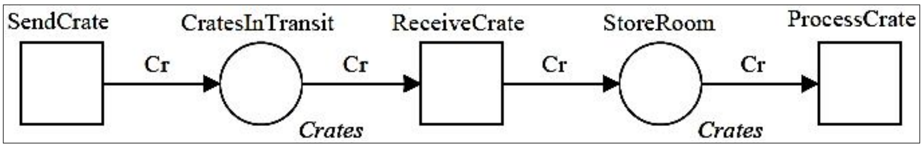


Fig. 3.39. Procedure for controlling the distributed crates.

Company A implements the agreed protocol by keeping a record of *SendingCredits*, while *Company B* keeps a record of empty slots available for storing crates in the store room. Any time that there are empty slots, *Company B* may reserve them and give the number, via a letter by setting the number of unreserved slots to 0, of reserved (and empty) crates as sending credits for crates to *Company A*. Whenever *Company A* receives such a letter, it increases *Sending-Credits* by the number written in it. Sending a crate, which is only possible if *SendingCredits* is 1 or more, lowers *SendingCredits* by 1, and processing a crate raises the number of empty and hence unreserved slots by one. Initially, the situation is as follows: no crate or letter is in transit; the store room is empty; there is no sending credit; and all slots are empty and unreserved (see **Fig. 3.40**). Note that this HLPNG models infinitely many different systems. It is a parameterized HLPNG with a parameter MAX, that may take any natural number as a value. Each such value *val*, substituted for MAX instantiates an 'ordinary' HLPNG without parameters.

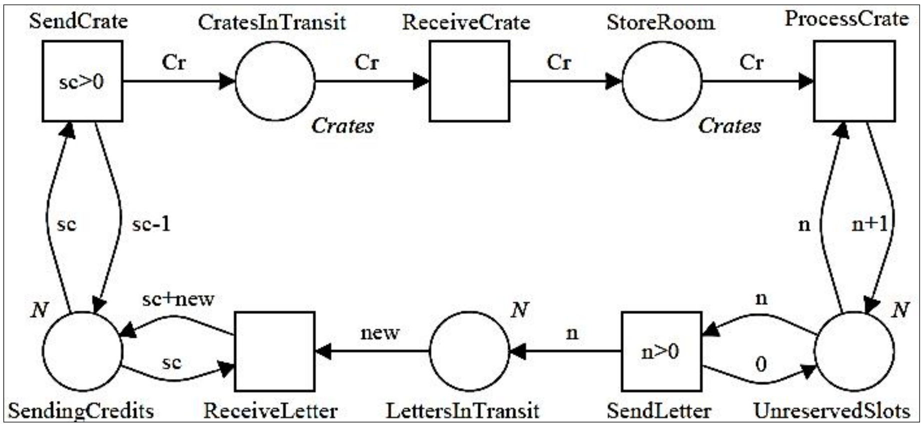


Fig.3.40. An extended distributed system with respect to the agreed protocol.

Crates = {Cr}
ℕ = {0, 1, 2…} the natural numbers
ℤ = {} the set of integers
n, new, sc: ℕ
+: *Z* × *Z* → *Z* is an arithmetic addition
- : *Z* × *Z* → *Z* is an arithmetic substraction
MAX: *N*
M_0 $(CratesInTransit)$ = M_0 $(LettersInTransit)$ = M_0 $(StoreRoom)$ = ∅
M_0 $(SendingCredits)$ = 1'0
M_0 $(UnreservedSlots)$ = 1'MAX

3.3.5 Petri Net Markup Language (PNML) for Petri Nets

The standardization process of HLPNs was one of the main reasons for developing PNML. Arising from the previous sections, this standard defines HLPNGs as a syntactic representation of HLPNs. PNML is designed to be a Petri net interchange format that is independent of specific tools and platforms [Kind 06].

PNML Design Principles

The design of PNML was governed by the principles of *flexibility, ambiguity* and *compatibility*.

Flexibility is related to the PNML specific extensions and features that contribute to representing any kind of Petri net. Petri nets can be converted to PNML and to this fact it provides the possibility to record all relevant specific information extracted from a Petri net, while not losing any detail. Furthermore, PNML has the ability to offer features to be applied by all kinds of Petri nets. Since PNML considers a Petri net as a labeled graph, additional information are stored in labels, which are attached to the net itself, to the nodes of the net, or to its arcs.

Ambiguity is achieved by ensuring that the original Petri net and its particular type can be traced precisely from its PNML representation. Consequently, PNML supports the definition of different *Petri net types*. A *Petri net type definition* (PNTD) declares the allowable labels for a particular Petri net type. PNML assigns a fixed type to each Petri net. Through this way the description are interpreted to be unambiguous.

Compatibility has to deal with information exchange of different types of Petri nets. PNML ensures this exchange of information as much as possible between different types of Petri nets by using conventions on how to define a label with a certain meaning. The syntax as well as the intended meaning of all kinds of extensions is predefined in a *Conventions Document*. To add a new Petri net type, labels are selected from this Conventions Document. When a new Petri net type conforms to these conventions for defining its own labels, the meaning of its labels is well-defined. Thus, Petri net based tools are allowed to interpret the net, even when the new Petri net type is unknown.

PNML Structure

PNML consists of several components that together form the internal structure of the PNML language (see **Fig. 3.32**). From a bottom-up view, the metamodel component, positioned in the PNML technology, defines the basic structure of a PNML file. A PNML file conforms to its metamodel.

PNML Technology components (*metamodel, feature definition interface, type definition interface*) are fixed.

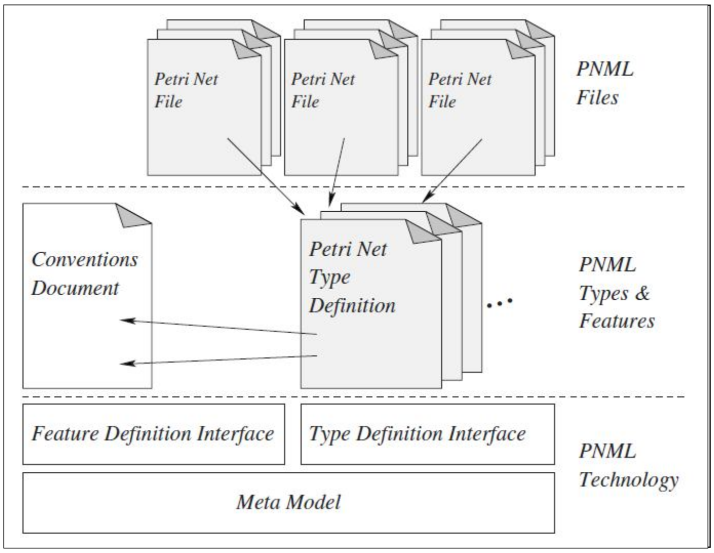


Fig. 3.32. Components of the Petri Net Markup Language.

The *Type Definition Interface* allows the definition of new Petri net types, whereas the *Feature Definition Interface* allows the definition of new features. To this end, the component states how the files must be structured.

The *Conventions Document* is extensible in the sense of adding new labels to the standard collection including description of their semantics and their typical use. From here, a new Petri Net Type Definition (PNTD) then can be built based on these predefined labels or possibly new ones. In contrast to the PNML Technology part, the Conventions Document, part of the PNML Types and Features layer, evolves. It contains definitions of a set of standard features. Similar to *Standard Petri Net Types* that defines a standard collection of types. These both conforms the feature and type definition interface. New *features* can be added to the Conventions Document and new types to the Standard Petri Net Types, which uses features from the Conventions Document. Due to this fact, the Conventions Document can be distributed and maintained efficiently through a web-based medium.

3.3.6 Summary of the Petri Net Architecture

The International Standard provides a well-defined semi-graphical technique for the specification, design and analysis of systems. The technique, High-level Petri Nets (HLPN), is mathematically defined. A more syntactical graphical way of HLPN is the High-Level Petri Net Graphs (HLPNG) that allows practitioners to describe the behaviour of a wide variety of discrete events and distributed systems; it allows visualization of system dynamics (i.e. flows of data and control). In practice Petri Nets can be deployed in different application fields, which could provide costs benefits and governance of systems. Petri Nets are executable by the Petri Net Markup Language [Kind 06], an XML based Petri net interchange format that is independent of specific tools and platforms.

Chapter 4

Concept Modeling

This chapter describes the concept mapping of business process from the ArchiMate modeling language ***directly*** to Petri Nets and ***indirectly*** via BPMN. It can be done analogously to the BPMN language with ArchiMate as intermediary language. Concept mapping [Wier 04],[03] between ArchiMate and BPMN is needed to concretize and to explicit the relation with respect to the integration of business processes. The semantics of concepts [04] are evaluated by comparing the directly and indirectly mappings. An basic example is given to clarify the key elements in mapping business process concepts.

Semantic Modeling using Petri Nets

Transformation of BPMN models to a formal modeling language such as Petri Nets could enable behavior analysis [Raed 07], [Dijk 08],[Chri 10]. This is very useful for verification and validation purposes. In this chapter, the modeling approach could provide a way for model verification and validation by transforming equivalent ArchiMate and BPMN models into Petri Nets. Each modeling language describes their architecture for a specific domain such as processes, products, infrastructure, organizations etc. by using their own characteristic elements. At different stages of transformation processes, models are required to be remodeled: modeling architectures of domains deals with different stakeholders from different backgrounds and their concerns. By doing this, it is important that semantically the models are equivalent in order to ensure quality of models without losing any information. Petri Net models are ambiguous and suitable for analysis. For this reason, the applied modeling approach should ensure whether the semantics of the remodeled ArchiMate / BPMN models are semantically identical by checking the correctness of the models using Petri Net.

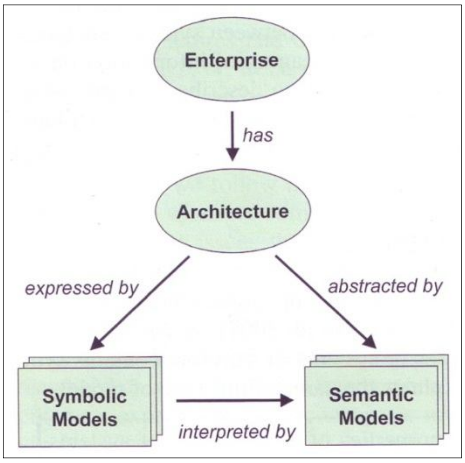


Fig. 4.1. Architecture covers symbolic and semantic models.

For a fundamental understanding of modeling enterprises, it is indispensable to consider the following approach in terms of *symbolic models* and *semantic models* (see **Fig. 4.1**). Enterprises may have architectures ranging from generic architectural models till specific architectural designs each expressed by their own characteristic symbolic models representing the real world as the domain of interests and abstracted by semantic models that are interpretations of symbolic models i.e. the actual meaning of the symbolic models in which they are used.

Semantics in Model Transformations by Means of Concepts

In Chapter 2 the underlying principles of ArchiMate and BPMN are already explained. The architecture of these languages, described in Chapter 3, differs in granularity (i.e. level of detail: generalization vs. specialization) and abstraction levels from each other, but shares some common similarities. The ArchiMate covers business, application and technology aspects, while BPMN comprises only the business part. From this perspective, BPMN can be seen as a specialization of ArchiMate and conversely, ArchiMate could be seen as a generalization of BPMN. When modeling the domain of interests, inter-relating these architectural domains could support modeling tools to easily collaborate: to model ArchiMate models in BPMN modeling tools and conversely, to model BPMN models in ArchiMate modeling tools. Applying the modeling approach of integration should effectuate this goal.

4.1 Abstract syntax of BPMN and ArchiMate

First of all, the syntax of both BPMN and ArchiMate are defined. The mixed constructs of both ArchiMate and BPMN makes it is possible to obtain models with a range of semantic errors. The ability to check the semantic correctness of models is a desirable feature for modeling tools based on these modeling techniques.

4.1.1. Abstract syntax of BPMN

It is necessary to demarcate what to be considered as to be part of a core BPMN process. Based on the metamodel of the BPMN language in Chapter 3 (**Sect. 3.23**), the abstract syntax of BPMN is defined. For this reason, some concepts from the BPMN specification have been omitted as such with respect to the overlapping parts in the ArchiMate language.

Definition of a well-formed core BPMN process:

A core BPMN process is a tuple, i.e. an ordered list of elements, where

$$\mathcal{P} = \begin{pmatrix} \mathcal{O},\mathcal{A},\mathcal{E},\mathcal{G},\mathcal{V},\mathcal{Z},\mathcal{R},\mathcal{T},\mathcal{S},\mathcal{E}^{\mathcal{S}},\mathcal{E}^{\mathcal{I}},\mathcal{E}^{\mathcal{E}},\mathcal{G}^{\mathcal{X}},\mathcal{G}^{\mathcal{I}},\mathcal{G}^{\mathcal{P}},\mathcal{V}^{\mathcal{P}},\mathcal{V}^{\mathcal{L}},\mathcal{Z}^{\mathcal{D}}, \\ ,\mathcal{Z}^{\mathcal{G}},\mathcal{R}^{\mathcal{S}},\mathcal{R}^{\mathcal{M}},\mathcal{R}^{\mathcal{A}} \end{pmatrix}$$

Notation	**Meaning**
$\mathcal{O}$	A set of *objects* which can be partitioned into disjoint sets of *activities* $\mathcal{A}$, *events* $\mathcal{E}$, *gateways* $\mathcal{G}$, *swimlanes* $\mathcal{V}$ and *artifacts* $\mathcal{Z}$.
$\mathcal{A}$	A set of *activities (an activity can be an atomic activity or non-atomic activity (compound))* can be partitioned into disjoint sets of *tasks* $\mathcal{T}$ and *subprocesses* $\mathcal{S}$.
$\mathcal{E}$	A set of *events* $\mathcal{E}$ can be partitioned into disjoint sets of *start events* $\mathcal{E}^{\mathcal{S}}$, *intermediate events* $\mathcal{E}^{\mathcal{I}}$ and *end events* $\mathcal{E}^{\mathcal{E}}$.
$\mathcal{G}$	A set of *gateways* can be partitioned into disjoint sets of *exclusive-gateways* $\mathcal{G}^{\mathcal{X}}$, *inclusive-gateways* $\mathcal{G}^{\mathcal{I}}$ and *parallel gateways* $\mathcal{G}^{\mathcal{P}}$.
$\mathcal{V}$	A set of *swimlanes* can be partitioned into disjoint of *pools* $\mathcal{V}^{\mathcal{P}}$ and *lanes* $\mathcal{V}^{\mathcal{L}}$.
$\mathcal{Z}$	A set of *artifacts* can be partitioned into disjoint of *data objects* $\mathcal{Z}^{\mathcal{D}}$ and *groups* $\mathcal{Z}^{\mathcal{G}}$.
$\mathcal{R}$	A set of *connecting objects* can be partitioned into disjoint sets of *sequence flow relations* $\mathcal{R}^{\mathcal{S}}$, *message flow relations* $\mathcal{R}^{\mathcal{M}}$ and *associations* $\mathcal{R}^{\mathcal{A}}$.
$\mathcal{R} \subseteq \mathcal{O}\ x\ \mathcal{O}$	The set of *relations* between *flow objects*.

Requirements
$\forall\, s \in \mathcal{E}^{\mathcal{S}}, incoming(s) = \emptyset\ \wedge \mid outcoming(s)\mid = 1$
$\forall\, e \in \mathcal{E}^{\mathcal{E}}, incoming(e) = 1\ \wedge \mid outcoming(e)\mid = \emptyset$
$\forall\, i\ \in \mathcal{E}^{\mathcal{I}}, incoming(i) =\ 1\ \wedge \mid outcoming(i)\mid = 1$
$\forall\, g \in \mathcal{G}^{\mathcal{X}} \cup \mathcal{G}^{\mathcal{I}} \cup \mathcal{G}^{\mathcal{P}}, incoming(g) =\ 1\ \wedge \mid outcoming(g)\mid\ > 1$ ***(SPLITS)***
$\forall\, g \in \mathcal{G}^{\mathcal{X}} \cup\ \mathcal{G}^{\mathcal{I}} \cup\ \mathcal{G}^{\mathcal{P}}, incoming(g)\ >\ 1\ \wedge \mid outcoming(g)\mid = 1$ ***(JOINS)***
$\forall\ \ x \in \mathcal{O}, \exists\, s \in \mathcal{E}^{\mathcal{S}}, \exists\, e\, \mathcal{E}^{\mathcal{E}}, s\mathcal{R}x\ \wedge x\mathcal{R}^{*}e$

$\forall\, s \in \mathcal{E}^{\mathcal{S}}, incoming(s) = \emptyset\ \wedge \mid outcoming(s)\mid = 1$, means that *every* single start event has <u>*not an incoming flow*</u> denoted by the empty collection ∅. A start event does have an <u>*outcoming flow*</u>, thus has the value 1.

When looking at end events $\forall\, e \in \mathcal{E}^{\mathcal{E}}, incoming(e) = 1\ \wedge \mid outcoming(e)\mid = \emptyset$, it shows the opposite of start events. End events have an incoming flow with value '1' and not having an outcoming flow denoted by the empty collection ∅.

Further reasoning, $\forall\, i\ \in \mathcal{E}^{I}, incoming(i) =\ 1\ \wedge \mid outcoming(i)\mid = 1$ has both the value 1 for incoming and outcoming flows in the case of an intermediate event. Advanced mechanism of this event type is excluded (e.g. exception handling, intermediate timer or message events that cause an exception).

Gateways are a special mechanism that join or split the <u>*incoming flows.*</u> In case of $\forall\, g \in \mathcal{G}^{\mathcal{X}} \cup \mathcal{G}^{\mathcal{I}} \cup \mathcal{G}^{\mathcal{P}}, incoming(g) = 1\ \wedge \mid outcoming(g)\mid\ > 1$, for <u>*each gateway g*</u> that is an element of the union of gateways $\mathcal{G}^{\mathcal{X}} \cup \mathcal{G}^{\mathcal{I}} \cup \mathcal{G}^{\mathcal{P}}$, have a single incoming flow, thus 1 and an outgoing flow that is > 1.

$\forall\, g \in \mathcal{G}^{\mathcal{X}} \cup\ \mathcal{G}^{\mathcal{I}} \cup\ \mathcal{G}^{\mathcal{P}}, incoming(g)\ >\ 1\ \wedge \mid outcoming(g)\mid = 1$. With regards to joins, it shows the opposite way of the previous one; it has for incoming flow the condition $>\ 1$ and when a join occurs, outgoing flow results in the condition = 1.

Finally, for all *start events* there exists a path to an *object* and *each object* is on a path from a *start event* to an *end event* denoted with $s\mathcal{R}x\ \wedge\ x\mathcal{R}^{*}e$, i.e. a reflexive transitive relation of $\mathcal{R}$. Thus, $\forall\ \ x \in \mathcal{O}, \exists\, s \in \mathcal{E}^{\mathcal{S}}, \exists\, e\, \mathcal{E}^{\mathcal{E}}, s\mathcal{R}x\ \wedge x\mathcal{R}^{*}e$ determines the above mentioned relationship.

A BPMN process might have multiple start events and end events. These situations are excluded to facilitate the translation process having only single events and single end events. The following requirements must be satisfies in order to be considered as a <u>*well-formed core BPMN model*</u>.

Definition of a well-formed core BPMN model:
A core BPMN model is a tuple $\mathcal{M}^B = (\varphi, \phi, \psi,\ \omega)$ where;

Notation	Meaning
φ	A set of *core BPMN processes* $(\mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_{n,})$
$\phi = \cup_{\mathcal{P}\in\varphi}\ \mathcal{S}_{\mathcal{P}}$	A collection of all *subprocesses* $\mathcal{S}$ *(i.e. compound activities).* $S_{\mathcal{P}}$ Indicates the *set of all subprocesses in* $\mathcal{P}$.
$\psi: \phi\ \rightarrow\ \varphi$ *(one-to-one mapping)* $\forall\ \phi, \varphi\ \in\ \psi:\ \psi(\phi) = \psi(\varphi)\ \rightarrow\ \phi =\ \varphi$	An ***injective function*** that ***maps*** *each subprocess* $\mathcal{S}$ (ϕ) *(atomic or non-atomic activity)* to *a core BPMN process*(φ).
$\omega = \{\ (\mathcal{P}_x, \mathcal{P}_y) \in\ \varphi\ \times\ \varphi\ \lvert \exists_{s\in\mathcal{S}_{\mathcal{P}x}}\ \psi(s) = \mathcal{P}_y\}$	*The relationship between a core BPMN process* and *its subprocess* (i.e. compound activity) is a connected graph.
$\mathcal{R}^M\ \subseteq \begin{pmatrix} \cup_{\mathcal{V}_p^{\mathcal{P}}\in\varphi}\ (\mathcal{T}_{\mathcal{P}}\ \cup \mathcal{S}_{\mathcal{P}}) \\ \times \\ \cup_{\mathcal{V}_p^{\mathcal{P}}\in\varphi}\ (\mathcal{T}_{\mathcal{P}}\ \cup \mathcal{S}_{\mathcal{P}}) \end{pmatrix}$ or $\mathcal{R}^M\ \subseteq (\cup_{\mathcal{P}\in\varphi}\ (V_{\mathcal{P}}^{\mathcal{P}}) \times \cup_{\mathcal{P}\in\varphi}\ (V_{\mathcal{P}}^{\mathcal{P}}))$	The set of *message flows* between *processes* in terms of ***pools***.

Requirements	Meaning
$\forall\omega\ \in\ \varphi \mid \omega = DAG$	Each ω is a *directed acyclic graph* (DAG)

A well-formed core BPMN model might consists of multiple business processes. Therefore φ is introduced that is a collection of defined sets of core BPMN processes. Distinction is made for subprocesses (i.e. compound activities), which is also considered as a set of core BPMN processes denoted with ϕ.

An injective function called ψ maps each *subprocess* $\mathcal{S}$ from the collection of ϕ to a core BPMN process that adds to the collection of φ. In addition, when dealing with relations between a well-formed core BPMN processes with respect to its *subprocess* $\mathcal{S}$, they have to be connected with each other as a connected graph.

A well-formed core BPMN model satisfies the requirement of a *directed acyclic graph.* In fact, a core BPMN process is a directed graph of *flow objects* $\boldsymbol{\mathcal{O}}$

and *connecting objects* $\mathcal{R}$, in particular *sequence flows* $\boldsymbol{\mathcal{R}}^{S}$. Furthermore, it is not allowed that there exists a path in the BPMN model such that the model in consideration is cyclic, i.e. there is no path starting at a certain *subprocess* $\boldsymbol{\mathcal{S}}$ that follows a sequence of flows $\boldsymbol{\mathcal{R}}^{S}$ and eventually return back to the *originate subprocess* $\boldsymbol{\mathcal{S}}$ again. In line with this, the path of the BPMN model is considered as a *directed acyclic graph* (DAG) in order to be a well-formed core BPMN model.

4.1.2. Abstract syntax of ArchiMate

At the same way the abstract syntax of the ArchiMate language can be defined, based on its metamodel described in Chapter 3 (**Sect. 3.1.5)**. As much as possible the ArchiMate business layer concepts, which are relevant for translating ArchiMate to BPMN, are included. Both the *business service* and the *business interface* aspect are not part of the definition, since it cannot be expressed in the BPMN language. This leads to the ArchiMate definition of well-formed core business process.

Definition of a well-formed core ArchiMate business process:

A core ArchiMate business process is a tuple of elements where $\mathbb{P}$ =

$$\begin{pmatrix} \mathcal{BE}, \mathcal{BSE}, \mathcal{BBE}, \mathcal{BO}, \mathcal{J}, \mathcal{RT}, \mathcal{BP}, \mathcal{BF}, \mathcal{BSE}^{\mathcal{A}}, \mathcal{BSE}^{\mathcal{R}}, \mathcal{BBE}^{\mathcal{E}}, \mathcal{J}^{\mathcal{N}}, \mathcal{J}^{\mathcal{A}}, \mathcal{J}^{\mathcal{O}}\mathcal{BP}^{\mathcal{A}}, \mathcal{RT}^{\mathcal{T}}, \\ \mathcal{RT}^{\mathcal{F}}, \mathcal{RT}^{\mathcal{C}}, \mathcal{RT}^{\mathcal{S}}, \mathcal{RT}^{\mathcal{G}} \end{pmatrix}$$

Notation	Meaning
$\mathcal{BE}$	A set of ***business elements*** which can be partitioned into disjoint sets of *business structure elements* $\mathcal{BSE}$, *business behaviour elements* $\mathcal{BBE}$ and *business objects* $\mathcal{BO}$.
$\mathcal{BSE}$	A set of ***business structure elements*** can be partitioned into disjoint sets of *business actors* $\mathcal{BSE}^{\mathcal{A}}$, *business roles* $\mathcal{BSE}^{\mathcal{R}}$.
$\mathcal{BBE}$	A set of ***business behavioral elements*** can be partitioned into disjoint sets of *business events* $\mathcal{BBE}^{\mathcal{E}}$ *business processes* $\mathcal{BP}$ and *business functions* $\mathcal{BF}$.
$\mathcal{BO}$	A set of ***business objects***.
$\mathcal{J}$	A set of ***junctions*** can be partitioned into disjoint sets of *Junctions* $\mathcal{J}^{\mathcal{N}}$, *AND-junctions* $\mathcal{J}^{\mathcal{A}}$ and *OR-junctions* $\mathcal{J}^{\mathcal{O}}$.
$\mathcal{BP}^{\mathcal{A}} \subseteq \mathcal{BP}$	A set of ***business activities*** (***specialization*** of $\mathcal{BP}$)
$\mathcal{RT}$	A set of ***relation types*** can be partitioned into disjoint sets of *trigger relations* $\mathcal{RT}^{\mathcal{T}}$, *flow relations* $\mathcal{RT}^{\mathcal{F}}$ and *access relations* $\mathcal{RT}^{\mathcal{C}}$, *association relations* $\mathcal{RT}^{\mathcal{S}}$ and *grouping relations* $\mathcal{RT}^{\mathcal{G}}$.
$\mathcal{RT} \subseteq \mathcal{BE} \times \mathcal{BE}$	A set of ***all relation types*** between *business elements*.
$\mathcal{RT}^{\mathcal{T}} \subseteq \mathcal{RT}$	*Trigger relations* are a subset of *relations*.

$\mathcal{RT}^{\mathcal{F}} \subseteq \mathcal{RT}$	*Flow relations* are a subset of *relations*.
$\mathcal{RT}^{\mathcal{C}} \subseteq \mathcal{RT}$	*Access relations* are a subset of *relations*
$\mathcal{RT}^{\mathcal{S}} \subseteq \mathcal{RT}$	*Association relations* are a subset of *relations*.
$\mathcal{RT}^{\mathcal{G}} \subseteq \mathcal{RT}$	*Grouping relations* are a subset of *relations*.

The following requirements must be satisfies in order to be a *well-formed core ArchiMate business process.*

Requirements
$\forall\, x,y \in \mathcal{BE}, x\mathcal{RT}^{\mathcal{T}}y \mid$ *if incoming(x) =* $\{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BBE}^{\mathcal{E}} \mid \mathcal{BP} \mid \mathcal{BR}\},\ incoming(y) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BBE}^{\mathcal{E}} \mid \mathcal{BP} \mid \mathcal{BF}\}$ $Otherwise\ incoming(x) = \{\mathcal{BSE}^{\mathcal{A}}, \mathcal{BF}\},\ incoming(y) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BR}\}$
$\forall\, x,y \in \mathcal{BE}, x\mathcal{RT}^{\mathcal{F}}y \mid$ $if\ incoming(x) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BBE}^{\mathcal{E}} \mid \mathcal{BP}\},\ incoming(y) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BBE}^{\mathcal{E}} \mid \mathcal{BP} \mid \mathcal{BF}\}$ $if\ incoming(x) = \{\mathcal{BSE}^{\mathcal{A}} \mid \mathcal{BR}\},\ incoming(y) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BR} \mid \mathcal{BF}\}$ $otherwise\ outcoming(x) = \{\mathcal{BF}\},\ incoming(y) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BP} \mid \mathcal{BSE}^{\mathcal{A}} \mid \mathcal{BR} \mid \mathcal{BF}\}$
$\forall\, x,y \in \mathcal{BE}, x\mathcal{RT}^{\mathcal{C}}y \mid$ outcoming$(x) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BBE}^{\mathcal{E}} \mid \mathcal{BP} \mid \mathcal{BSE}^{\mathcal{A}} \mid \mathcal{BR} \mid \mathcal{BF}\},\ incoming(y) = \{\mathcal{BO}\}$
$\forall\, x,y \in \mathcal{BE}, x\mathcal{RT}^{\mathcal{S}}y \mid$ $if\ incoming(x) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BBE}^{\mathcal{E}} \mid \mathcal{BP} \mid \mathcal{BSE}^{\mathcal{A}} \mid \mathcal{BR} \mid \mathcal{BF}\}\ then$ $incoming(y) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BBE}^{\mathcal{E}} \mid \mathcal{BP} \mid \mathcal{BSE}^{\mathcal{A}} \mid \mathcal{BR} \mid \mathcal{BF} \mid \mathcal{BO}\}$
$\forall\, x,y \in \mathcal{BE},\ x\mathcal{J}yy,\ xx\mathcal{J}y,\ RelType(x,y){:}\, flow \mid trigger \wedge x = y,$ $if\ incoming(x) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BBE}^{\mathcal{E}} \mid \mathcal{BP} \mid \mathcal{BSE}^{\mathcal{A}} \mid \mathcal{BR} \mid \mathcal{BF} \mid \mathcal{J}\}\ then$ $outcoming(y) = \{\mathcal{BP}^{\mathcal{A}} \mid \mathcal{BBE}^{\mathcal{E}} \mid \mathcal{BP} \mid \mathcal{BSE}^{\mathcal{A}} \mid \mathcal{BR} \mid \mathcal{BF} \mid \mathcal{J}\}$
$\forall\, j \in \mathcal{J}^{\mathcal{N}} \cup \mathcal{J}^{\mathcal{A}} \cup \mathcal{J}^{\mathcal{O}},\ if\ split\ then\ incoming(j) = 1 \wedge \lvert outcoming(j) \rvert > 1$
$\forall\, j \in \mathcal{J}^{\mathcal{N}} \cup \mathcal{J}^{\mathcal{A}} \cup \mathcal{J}^{\mathcal{O}}, if\ joins\ incoming(j) > 1 \wedge \lvert outcoming(j) \rvert = 1$
$\forall\ x,y \in \mathcal{BE}, \exists\, e \in \mathcal{BBE}^{\mathcal{E}},\ e\mathcal{RT}x \wedge x\mathcal{RT}y$
$\mathcal{RT}^{\mathcal{G}} = \cup_{\mathcal{BP} \in \mathbb{P}} \mathcal{BE}_{\mathcal{BP}}$

According to the definition of a well-formed ArchiMate business process, its structure reflects the core of the ArchiMate language (see **Sect. 3.1.4**). An ArchiMate business process is in fact a relation between *business structure elements* that triggers the behaviour, *business behaviour elements* that shows the actual behaviour which is assigned to a business structure element (i.e. initiator) and *business objects* on which behaviour is performed.

A *business activity* is a special business behaviour element that is created via specialization of the *business process* (i.e. parent). Furthermore, five special kinds of relations are considered to be important: *trigger, flow, access, association, and grouping* relations.

To become a well-formed core business process, the above defined requirements needs to apply in order to prevent unwell-formed erroneous business processes. The trigger relation requirement $\forall\, x, y \in$

$\mathcal{BE}, x\mathcal{RT}^{\mathcal{T}}y$ means that there exists a path from a $\mathcal{BE}$-element to another $\mathcal{BE}$-element.

For a precise definition that governs the permitted relations of trigger relations, some constraints are added. The requirement $\forall\, x \in\ \mathcal{BP}^{\mathcal{A}}, incoming(y) = \{\mathcal{BP}^{\mathcal{A}}, \mathcal{BBE}^{\mathcal{E}}, \mathcal{BP}, \mathcal{BF}\}$ indicates that a business activity only may connect to another business activity, a business event, a business process, or a business function.

This is done at the same way for all associated business elements of different relation types. ArchiMate is able to handle dynamic relationships. A *junction* is used to connect dynamic relationships of the same type. It is used in a number of situations to connect dynamic (triggering or flow) relationships of the same type to indicate **splits** or **joins**.

Three types can be discerned into joins, AND-joins and OR-joins. Joins can *split* and *join* both incoming and outcoming flows, whereby *at least one* of the two flows needs to be activated before it proceeds. AND-joins mean that *both incoming and outcoming flows* are required to activate, while OR-joins proceeds when *exactly one* of the in- and outcoming flow satisfies. Splits have incoming flows that results in $= 1$ and an outcoming flow of > 1. Joins is exactly the opposite. Joins can handle only flow or trigger relations for in- and outcoming flows, where it is permitted to use different relations types for each incoming and outcoming flows.

Next, $\forall\ x, y \in \mathcal{BE}, \exists\ e \in \mathcal{BBE}^{\mathcal{E}},\ e\mathcal{RT}x \wedge\ x\mathcal{RT}y$ states that there exists a path from a business event to a business element. From then, there is a path from a certain business element that finally ends with another business element. The requirement $\mathcal{RT}^{\mathcal{G}} = \cup\, \mathcal{BE}$ declares that objects of the same type or different types belong together (based on some common characteristic).

<u>Definition of a well-formed core ArchiMate business model</u>:
A core ArchiMate model is a tuple $\mathcal{M}^{\mathcal{A}} = (\delta, \lambda)$ where:

Notation	Meaning
δ	A set of *core ArchiMate processes* $(\mathcal{BP}_{1,}\mathcal{BP}_{2,}\ \ldots\ \mathcal{BP}_{n,})$
$\lambda = \{(\mathbb{P}_{1,}\mathbb{P}_2) \in\ \delta\ \times\ \delta\}$	A relation between ArchiMate processes is a connected graph.

A well-formed core ArchiMate business model might consists of multiple business processes. Therefore δ is introduced that is a collection of defined sets of core ArchiMate processes. Eventually, the relation between ArchiMate processes mutually is a connected graph to satisfy a well-formed core ArchiMate business model.

4.2 Mapping to Petri Nets (Directly)

The abstract syntax is given in the previous section of both ArchiMate and BPMN languages. Furthermore, a definition of well-formed core business model serves as a basis for models being created in these languages and thus a way to restrict models. To this end, selecting those parts as being considered to be an essential part can be demarcated. As the mapping to Petri Nets can be performed, the semantic modeling approach are simply graphically represented as:

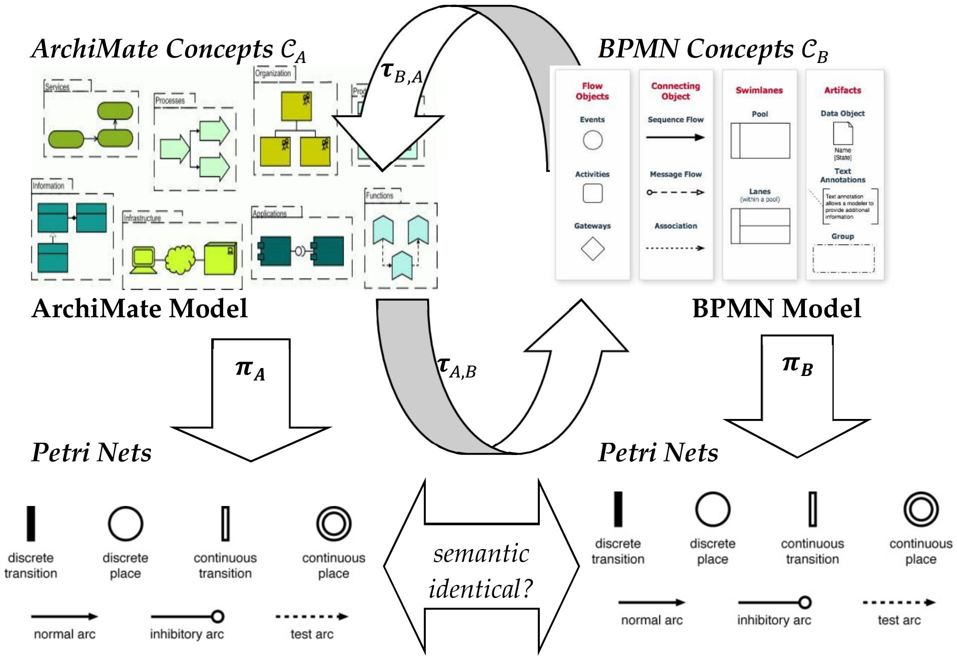


Table 4.1 Notation and their meaning are summarized.

Notation	Meaning
$\mathcal{C}_A$	From the perspective of an ArchiMate concept.
$\mathcal{C}_B$	From the perspective of a BPMN concept.
π_A	The semantic of an ArchiMate concept in terms of Petri Nets.
π_B	The semantic of a BPMN concept in terms of Petri Nets.
$\tau_{A,B}$	The transformation from an ArchiMate concept towards a BPMN concept.
$\tau_{B,A}$	The transformation from a BPMN concept towards an ArchiMate concept.

When checking the semantic correctness of both the ArchiMate and BPMN concepts in terms of Petri Nets, the equation of semantic correctness can be described as follow:

Hypotheses

1. $\pi_A(\mathcal{C}_A) \quad \equiv? \quad \pi_B(\tau_{A,B}(\mathcal{C}_A))$

2. $\pi_B(\mathcal{C}_B) \quad \equiv? \quad \pi_A(\tau_{B,A}(\mathcal{C}_B))$

The hypothesis 1 and 2 holds when $\vDash \pi_B\left(\tau_{A,B}(\mathcal{C}_A)\right) \leftrightarrow \pi_A(\mathcal{C}_A)$ and $\vDash \pi_A\left(\tau_{B,A}(\mathcal{C}_B)\right) \leftrightarrow \pi_B(\mathcal{C}_B)$ on the assumption that $\pi_A(\mathcal{C}_A) \equiv \pi_B\left(\tau_{A,B}(\mathcal{C}_A)\right)$ and $\pi_B(\mathcal{C}_B) \equiv \pi_A\left(\tau_{B,A}(\mathcal{C}_B)\right)$ satisfies.

Instead of starting with complex ArchiMate or BPMN models, this section introduces the comparison at a conceptual level as well as the operational level in terms by checking the semantics in terms of Petri Net concepts. These two comparison mechanism provides a much better understanding of the distinct modelling concepts. At this way, the features of each concept can be unambiguously defined and graphically expressed. Thereby it is much easier to gain a comprehensive view of the development of constructing an entire Petri Net model. Each single concept needs to translate to another equivalent concept. The translation describes a way from both ArchiMate and BPMN concepts to Petri Net equivalent concepts which has two ways:

Directly translation at conceptual level

Using concepts from ArchiMate can directly translate to equivalent Petri Net concepts $\pi_A(\mathcal{C}_B)$. At the same way, concepts from BPMN can directly translate to Petri Net concepts, denoted by the notation $\pi_B(\mathcal{C}_B)$.

Indirectly translation at operational level

Using concepts from ArchiMate can indirectly translate via the BPMN language concepts denoted by $\tau_{A,B}(\mathcal{C}_A)$. At the same way, concepts from BPMN can indirectly translate to Petri Net concepts via ArchiMate as intermediate language denoted by the notation $\tau_{B,A}(\mathcal{C}_B)$.

Comparison (Conceptual vs. Operational)

When taken this semantic modelling approach, it is possible to see what are either the differences or similarities. Based on the translation process, the hypotheses 1 and 2 can be evaluated by doing so.

4.2.1 BPMN π_B

In this stage, BPMN concepts are directly mapped to equivalent Petri Net concepts.

<u>Petri Net notation</u>

$p_{letter} = places$
$t_{letter} = transitions\ (labled\ with\ a\ name)$
$(p_{letter}, t_{letter}) = arcs(from\ places\ to\ transitions, name)$
$(t_{letter}, p_{letter}) = arcs(from\ transitions\ to\ places, name)$
$M_0(p_{letter}) = initial\ marking\ of\ a\ certain\ place - p_{letter}$

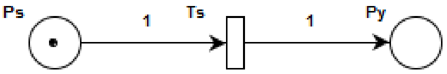
$Let\ C_1 \dots C_{15}\ be\ concepts\ from\ \mathcal{P}\ then\ the\ following\ contructs\ can\ be\ translated$:

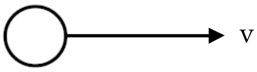
<u>NOTE:</u> Let the variable $x\ and\ y$ be input and output places. Sometimes $s\ and\ e$ are used for places that represents the *start place* and the *end place*. Letters used for transitions starts with one or two letter(s) of the concept name (e.g., t_{SP} for sub-process and t_s for start event).

<u>Start event</u>

$\pi_B\ (C_1) = \ \mathcal{E}^{\mathcal{S}}$

A *start event* triggers 'others', so a flow direction is included where the desired next destination is called y. This leads to the following notation in Petri Net: $\mathcal{E}^{\mathcal{S}} = \left((p_s, t_s), (t_s, p_y)\right)$ with the assumption that the initial marking is $M_0(p_s) = \ t_s$. Note that $p_y\ in\ (t_s, p_y)$ illustrates the destination place that is on the background (graphically displays a dotted eclipse or circle). Transitions are intended to model the execution of the concepts behaviour, in this case the trigger originating from a start event. The associated transition is used to signal when the process starts. Thus, $\pi_B(\mathcal{E}^{\mathcal{S}}) = \left((p_s, t_s), (t_s, p_y)\right)$.





Start event

<u>Intermediate event</u>

$\pi_B\ (C_2) = \ \mathcal{E}^{\mathcal{I}}$

An *intermediate event* is distinguished from start/end events by having an incoming and an outcoming flow. In Petri Net: $\mathcal{E}^{\mathcal{I}} = \left((p_x, t_i), (t_i, p_y)\right)$ with the assumption that the initial marking is $M_0(p_x) = \ t_i$. Note that variable $p_x\ in\ (p_x, t_i)\ and\ p_y\ in\ (t_i, p_y)$ illustrates both the originating and destination place that plays on the background (graphically displays a dotted eclipse or circle). In this case the invoked trigger (such as *message* or *time* trigger event) conveys the tokens (e.g. message or the time) to place p_y.

Thus, $\pi_B(\mathcal{E}^{\mathcal{I}}) = \left((p_x, t_i), (t_i, p_y)\right)$.

End event

$\pi_B\,(C_3) = \mathcal{E}^{\mathcal{E}}$

An *end event* is similar to start events, but in reverse. This leads to the following notation in Petri Net: $\mathcal{E}^{\mathcal{E}} = \big((p_x, t_e), (t_e, p_e)\big)$ with the assumption that the initial marking is $M_0(p_e) = t_e$. Note that variable $p_x\ in\ (p_x, t_e)$ illustrates the input place that serves on the background (graphically displays a dotted eclipse or circle). The transition t_e is intended to signal the end of the associated task or process, resulting in an end event denoted with the place p_e. Thus, $\pi_B(\mathcal{E}^{\mathcal{E}}) = \big((p_x, t_e), (t_e, p_e)\big)$.

Task

$\pi_B(C_4) = \mathcal{T}$

A *task* $\mathcal{T}$ in Petri Net is given by $\mathcal{T} = \Big((p_x, t_{task}), \big(t_{task}, p_y\big)\Big)$ with the assumption that the initial marking is $M_0(p_x) = t_{task}$. Note that variable $p_x\ in\ (p_x, t_{task})$ and $p_y\ in\ \big(t_{task}, p_y\big)$ illustrates the input and output place that graphically displays a dotted eclipse or circle, since it is not known what the associated places are.

The translation would be $\pi_B(\mathcal{T}) = \Big((p_x, t_{task}), \big(t_{task}, p_y\big)\Big)$.

Subprocess

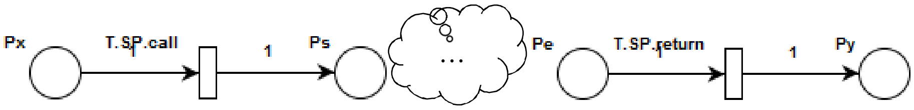
$\pi_B(C_5) = \mathcal{S}$

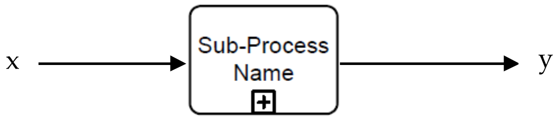
A *Subprocess* $\mathcal{S}$ in Petri Net is given by:

$$\mathcal{S} = \Bigg(\Big((p_x, t_{SPcall}), (t_{SPcall}, p_s)\Big) \ldots \Big((p_e, t_{SPreturn}), \big(t_{SPreturn}, p_y\big)\Big)\Bigg)$$

Assume that the initial marking is $M_0(p_x) = t_{SPcall}$. A *subprocess* $\mathcal{S}$ is invoked by the transition t_{SPcall} and after accomplishing the associated subprocess, it returns to the next object by the transition $t_{SPreturn}$. Note that the variable $p_x\ in\ \big(p_x, t_{SP,call}\big)$ and $p_y\ in\ \big(t_{SP,return}, p_y\big)$ illustrate the input and output place that graphically displays a dotted eclipse or circle, since it is not known what the associated places are. This leads to the following translation of a subprocess:

$$\pi_B(\mathcal{S}) = \Bigg(\Big((p_x, t_{SPcall}), (t_{SPcall}, p_s)\Big) \ldots \Big((p_e, t_{SPreturn}), \big(t_{SPreturn}, p_y\big)\Big)\Bigg).$$





Exclusive Gateway

$\pi_B(C_6) = \mathcal{G}^{\mathcal{X}}$

An *exclusive gateway* $\mathcal{G}^{\mathcal{X}}$ in Petri Net is given by $\mathcal{G}^{\mathcal{X}} = \left(\left(p_x, (t_{EG1}, t_{EG2})\right), \left((t_{EG1}, p_{y1}), (t_{EG2}, p_{y2})\right)\right)$. This applies for single inputs. Assume that the initial marking is $M_0(p_x) = t_{EG1}, t_{EG2}$. The Petri Net formulation for joining two flows is given by: $\mathcal{G}^{\mathcal{X}} = \left(\left((p_{x1}, t_{EG1}), (p_{x2}, t_{EG2})\right), \left((t_{EG1}, p_{y1}), (t_{EG2}, p_{y1})\right)\right)$. Assume that the initial markings are $M_0(p_{x1}) = t_{EG1}$ and $M_0(p_{x2}) = t_{EG2}$.

For **splits** $\pi_B(\mathcal{G}^{\mathcal{X}}) = \left(\left(p_x, (t_{EG1}, t_{EG2})\right), \left((t_{EG1}, p_{y1}), (t_{EG2}, p_{y2})\right)\right)$ and

for **joins** $\pi_B(\mathcal{G}^{\mathcal{X}}) = \left(\left((p_{x1}, t_{EG1}), (p_{x2}, t_{EG2})\right), \left((t_{EG1}, p_{y1}), (t_{EG2}, p_{y1})\right)\right)$

Parallel Gateway

$\pi_B(C_7) = \mathcal{G}^{\mathcal{P}}$

A *parallel gateway* $\mathcal{G}^{\mathcal{P}}$ in Petri Net is given by:

$\mathcal{G}^{\mathcal{P}} = \left((p_x, t_{PG}), \left(t_{PG}, (p_{y1}, p_{y2})\right)\right)$. This is the case of splitting flows. Assume that the initial marking is $M_0(p_x) = t_{PG}$. The Petri Net formulation for joining two flows is given by $\mathcal{G}^{\mathcal{P}} = \left(((p_{x1}, p_{x2}), t_{PG}), (t_{PG}, p_y)\right)$. Assume that the initial markings are $M_0(p_{x1}) = t_{PG}$ and $M_0(p_{x2}) = t_{PG}$.

With regards to **splits** $\pi_B(\mathcal{G}^{\mathcal{P}}) = \left((p_x, t_{PG}), \left(t_{PG}, (p_{y1}, p_{y2})\right)\right)$.

For **joins** $\pi_B(\mathcal{G}^{\mathcal{P}}) = \left(((p_{x1}, p_{x2}), t_{PG}), (t_{PG}, p_y)\right)$.

Inclusive Gateway

$\pi_B(C_8) = \mathcal{G}^{\mathcal{I}}$

An *inclusive gateway* $\mathcal{G}^{\mathcal{I}}$ in Petri Net is given by:

$\mathcal{G}^{\mathcal{I}} = \left(\left(p_x, (t_{IG1}, t_{IG2})\right), \left((t_{IG1}, p_{y1}), (t_{IG2}, p_{y2})\right)\right)$This is the case for splits. Assume that the initial marking is $M_0(p_x) = t_{IG1}, t_{IG2}$. The Petri Net formulation for two joining flows is given by $\mathcal{G}^{\mathcal{I}} = \left(((p_{x1}, p_{x2}), t_{IG}), (t_{IG}, p_y)\right)$. The two paths should have been activated to proceed. Assume that the initial markings are $M_0(p_{x1}) = t_{IG}$.

With regards to **splits** $\pi_B(\mathcal{G}^{\mathcal{I}}) = \left(\left(p_x, (t_{IG1}, t_{IG2})\right), \left((t_{IG1}, p_{y1}), (t_{IG2}, p_{y2})\right)\right)$.

For **joins** the translation is $\pi_B(\mathcal{G}^{\mathcal{I}}) = \left(((p_{x1}, p_{x2}), t_{IG}), (t_{IG}, p_y)\right)$.

Pool

$\pi_B(C_9) = \mathcal{V}^{P}$

A *pool* $\mathcal{V}^{\mathcal{P}}$ is used for grouping a number of *tasks* or *subprocesses* to participants. When considering the pool singly, it cannot be defined one-to-one in a Petri Net graph as a pool does not considered to be an event type that triggers 'others'. Assigning relationships cannot be defined in Petri Net semantics.

Lane

$\pi_B(C_{10}) = \mathcal{V}^{\mathcal{L}}$

A *lane* $\mathcal{V}^{\mathcal{L}}$ is used for grouping a number of *tasks* or *subprocesses* to participants *within* pools. When considering the lane singly, it cannot be defined neither one-to-one in terms of a Petri Net graph due to the fact that a lane refer to an assigning relationship.

Data Object

$\pi_B(C_{11}) = \mathcal{Z}^{\mathcal{D}}$

A *data object* $\mathcal{Z}^{\mathcal{D}}$ can be considered as a *message*. A data object represents information that flows through the process, including business documents, e-mail or letters. A data object (e.g. e-mail) flows into an activity, i.e. tasks or subprocess, by means of *message start events*. $\mathcal{Z}^{\mathcal{D}} = (p_s, t_{task,DO})$. Note that $p_y \in \mathcal{T}, \mathcal{S}$ in $(t_{task,DO}, p_y)$ and $p_s \in$ message $\mathcal{E}^{\mathcal{S}}$ in $(p_s, t_{task,DO})$ illustrate the input/output place that graphically displays a dotted eclipse or circle. The initial marking of $M_0(p_s) = t_{task,DO}$ that leads to $\pi_B\ (\mathcal{Z}^{\mathcal{D}}) = (p_s, t_{DO})$. When data objects are *exchanged from tasks/subprocess* the following notation applies where a data object (e.g. an e-mail) is exchanged $\pi_B\ (\mathcal{Z}^{\mathcal{D}}) = (t_{task,DO}, p_y)$ and the initial marking of $M_0(p_y) = t_{task,DO}$. The same holds for subprocesses by replacing *task* by *SP*.

Group

$\pi_B(C_{12}) = \mathcal{Z}^{\mathcal{G}}$

A *group artifact* $\mathcal{Z}^{\mathcal{G}}$ is used for demarcation. Therefore a group concept in Petri Net is not possible to define separately, because it depends on which *objects* belongs to the group including its *relations*.

Sequence Flow

$\pi_B(C_{13}) = \mathcal{R}^{\mathcal{S}}$

A *sequence flow* $\mathcal{R}^{\mathcal{S}}$ in Petri Net is associated with *objects* and therefore cannot be defined separately, as the relation in Petri Net does not differentiate.

Message Flow

$\pi_B(C_{14}) = \mathcal{R}^{\mathcal{M}}$

A *message flow* $\mathcal{R}^{\mathcal{M}}$ in Petri Net is associated with *objects* and therefore cannot be defined separately, as the relation in Petri Net does not differentiate. Note, a pool as a coherent whole of *tasks/subprocesses* and *events* that is assigned to a participant allows the transmission of messages between pools in terms of *message flows*. When this situation occurs, the transmissions of messages can be discerned into (a) task to task/subprocess, subprocess to subprocess/task (b) end event to task/subprocess, (c) task/subprocess to start event and (d) end event to start event.

Let x and y be concept from different pools that are connected by a message flow, then $(x, y)$is in one of the following formats:

(a) $(t_{task(x)}, t_{task(y)}), (t_{SP(x)}, t_{SP(y)}), (t_{task(x)}, t_{SP(y)}), (t_{SP(x)}, t_{task(y)})$

(b) $(p_e, t_{task}), (p_e, t_{SP})$

(c) $(t_{task}, p_s), (t_{SP}, p_s)$

(d) (p_e, p_s)

Thus, $\pi_B(\mathcal{R}^{\mathcal{M}}) = (\mathrm{a} \wedge b \wedge\ c \wedge\ d)$

Association

$\pi_B(C_{15}) =\ \mathcal{R}^{\mathcal{A}}$

An *association* $\mathcal{R}^{\mathcal{A}}$ is associated with *objects* and therefore cannot be defined separately, as the relation in Petri Net does not differentiate.

Table 4.2 BPMN objects and its equivalent Petri Net modules

BPMN object		**Petri Net equivalent module**
Start event	$\mathcal{E}^{\mathcal{S}}$	$\mathcal{E}^{\mathcal{S}} = \big((p_s, t_s)(t_s, p_y)\big)$
Intermediate event	$\mathcal{E}^{\mathcal{I}}$	$\mathcal{E}^{\mathcal{I}} = \big((p_x, t_i), (t_i, p_y)\big)$
End event	$\mathcal{E}^{\mathcal{E}}$	$\mathcal{E}^{\mathcal{E}} = \big((p_x, t_e), (t_e, p_e)\big)$
Tasks	$\mathcal{T}$	$\mathcal{T} =\ \big((p_x, t_{task}), (t_{task}, p_y)\big)$
Subprocess	$\mathcal{S}$	$\mathcal{S} = \begin{pmatrix} \big((p_x, t_{SPcall}), (t_{SPcall}, p_s)\big) \\ \cdots \\ \big((p_e, t_{SPreturn}), (t_{SPreturn}, p_y)\big) \end{pmatrix}$
Exclusive Gateway	$\mathcal{G}^{\mathcal{X}}$	$splits = \begin{pmatrix} \big((p_x, (t_{EG1}, t_{EG2}))\big), \\ \big((t_{EG1}, p_{y1}), (t_{EG2}, p_{y2})\big) \end{pmatrix}$ $joins = \begin{pmatrix} \big((p_{x1}, t_{EG1}), (p_{x2}, t_{EG2})\big), \\ \big((t_{EG1}, p_{y1}), (t_{EG2}, p_{y1})\big) \end{pmatrix}$
Parallel Gateway	$\mathcal{G}^{\mathcal{P}}$	$splits = \big((p_x, t_{PG}), (t_{PG}, (p_{y1}, p_{y2}))\big)$ $joins =\ \big(((p_{x1}, p_{x2}), t_{PG}), (t_{PG}, p_y)\big)$
Inclusive Gateway	$\mathcal{G}^{\mathcal{I}}$	$splits =\ \Big((p_x, (t_{IG1}, t_{IG2})), \big((t_{IG1}, p_{y1}), (t_{IG2}, p_{y2})\big)\Big)$ $joins = \big(((p_{x1}, p_{x2}), t_{IG}), (t_{IG}, p_y)\big)$
Pool	$\mathcal{V}^{\mathcal{P}}$	Not available (N.A.)

Lane	$\mathcal{V}^{\mathcal{L}}$	Not available (N.A.)
Data Object	$\mathcal{Z}^{D}$	$\mathcal{Z}^{D} = \left((p_s, t_{a,DO}), (t_{a,DO}, p_y)\right)\ t_a = \{t_{task}\ \vert t_{SP}\}$
Group	$\mathcal{Z}^{\mathcal{A}}$	Not available (N.A.)
Sequence flow	$\mathcal{R}^{\mathcal{S}}$	Not available (N.A.)
Message flow	$\mathcal{R}^{\mathcal{M}}$	$\mathcal{R}^{\mathcal{M}} = (\mathrm{a} \wedge b \wedge\ c \wedge\ d)$ (a) $(t_{task(x)}, t_{task(y)}), (t_{SP(x)}, t_{SP(y)}),$ $(t_{task(x)}, t_{SP(y)}), (t_{SP(x)}, t_{task(y)})$ (b) $(p_e, t_{task}), (p_e, t_{SP})$ (c) $(t_{task}, p_s), (t_{task}, p_{SP})$ (d) (p_e, p_s)
Association	$\mathcal{R}^{\mathcal{A}}$	Not available (N.A.)

4.2.2 ArchiMate π_A

Previous section already shows the mapping from BPMN core concepts to Petri Net semantics. Now, the ArchiMate concepts are directly mapped to the semantics of Petri Net at the same way. Table 4.3 summarizes the comparison of graphical symbols between ArchiMate and Petri Nets.

$Let\ C_1 \ldots C_{15}\ be\ concepts\ from\ \mathbb{P}\ then\ the\ following\ contructs\ can\ be\ translated$:

<u>Business Actor</u>

$\pi_A(C_1) = \mathcal{BSE}^{\mathcal{A}}$

A *business actor* $\mathcal{BSE}^{\mathcal{A}}$ is similar to participants in BPMN, so it cannot be defined in Petri Net terms. Business actors fulfill a certain business role in a business process, so a business actor is assigned to a business role.

<u>Business Role</u>

$\pi_A(C_2) = \mathcal{BSE}^{\mathcal{R}}$

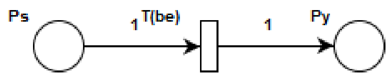
A *business role* $\mathcal{BSE}^{\mathcal{R}}$ is not applicable in terms of a Petri Net graph for the reason that a business role can be assigned to a business actor (assigning relationship) that indicates that resources (i.e. the actors) are granted or deployed for business tasks, which are related to which positions the actors hold (i.e. roles).

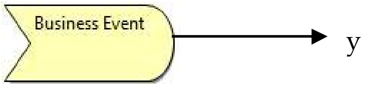
<u>Business Event</u>

$\pi_A(C_3) =\ \mathcal{BBE}^{\mathcal{E}}$

A *business event* $\mathcal{BBE}^{\mathcal{E}}$ in Petri Net is given by:

$\mathcal{BBE}^{\mathcal{E}} = \left((p_s, t_{be}), (t_{be}, p_y)\right)$. A business event triggers or triggered by a business process, but internal triggers may occur. The initial marking is $M_0(p_s) =\ t_{be}$. Note that $p_y\ (should\ be\ a\ \mathcal{BP}\ in\ (t_{be}, p_y))$ illustrates the destination place that is on the background (graphically displays a dotted eclipse or circle). The translation is $\pi_A(\mathcal{BBE}^{\mathcal{E}}) = \left((p_s, t_{be}), (t_{be}, p_y)\right)$ for events that *triggers* others. For *intermediate triggers* $\pi_A(\mathcal{BBE}^{\mathcal{E}}) = \big((p_x, t_{be}), (t_{be}, p_y)\big)$ and business events that are *triggered by* others that ends the entire business process $\pi_A(\mathcal{BBE}^{\mathcal{E}}) = \big((p_x, t_{be}), (t_{be}, p_e)\big)$.





Business Process / Business Activity

$\pi_A(C_4) = \mathcal{BP}$

A *business process* $\mathcal{BP}$ / *business activity* $\mathcal{BP}^{\mathcal{A}}$ in Petri Net is given by:

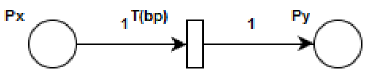
$\mathcal{BP} = \big((p_x, t_{bp}), (t_{bp}, p_y)\big)$ and $\mathcal{BP}^{\mathcal{A}} = \big((p_x, t_{ba}), (t_{ba}, p_y)\big)$ Note that variable p_x should be of type $\mathcal{BSE}^{\mathcal{E}}, \mathcal{BP}, \mathcal{BF}$ in (p_x, t_{bp}) and (p_x, t_{ba}).

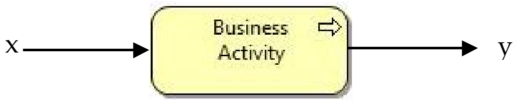
It represents the related place, where the incoming flow starts from. The variable p_y in (t_{bp}, p_y) and (t_{ba}, p_y) should be of type $\mathcal{BSE}^{\mathcal{E}}, \mathcal{BP}, \mathcal{BF}$. These places are graphically depicted as a dotted eclipse or circle. The initial marking is $M_0(p_x) = t_{bp} / t_{ba}$.

A business process is triggered by / or triggers a business event, a business function or other business processes. This leads to the translation $\pi_A(\mathcal{BP}) = \big((p_x, t_{bp}), (t_{bp}, p_y)\big)$ and $\pi_A(\mathcal{BP}^{\mathcal{A}}) = \big((p_x, t_{ba}), (t_{ba}, p_y)\big)$.

The differences between a *process* and a *activity* is that an activity is a specialization of business process, and considers as the smallest peace of work in a business process.

Initial phase would be described as
$\pi_A(\mathcal{BP}) = \big((p_s, t_{bp}), (t_{bp}, p_y)\big)$ or $\pi_A(\mathcal{BP}) = \big((p_s, t_{bp}), (t_{bp}, p_e)\big)$ when consider one business process. For the final phase this leads to $\pi_A(\mathcal{BP}) = \big((p_x, t_{bp}), (t_{bp}, p_e)\big)$. This also applies to a business activity.





Business Function

$\pi_A(C_5) = \mathcal{BF}$

A *business function* $\mathcal{BF}$ in Petri Net is given by:

$\mathcal{BF} = \big((p_x, t_{bf}), (t_{bf}, p_y)\big)$. See requirements for the permitted starting and destination places of variable p_x and p_y based on the selected relation type.

Since it includes all relations, an abstraction is needed to obtain relevant objects by means of *sequence flows* and *message flows*. These places are graphically depicted as a dotted eclipse or circle. The initial marking is $M_0(p_x) = t_{bf}$.
A business function captures business processes for supporting purposes. Derived from this, a business process is part of a business function. The translation would be $\pi_A(\mathcal{BF}) = \left((p_x, t_{bf}), (t_{bf}, p_y)\right)$.

Business Object

$\pi_A(C_6) = \mathcal{BO}$
A *business object* $\mathcal{BO}$ in Petri Net is given by:
$\mathcal{BO} = (p_x, t_{BP,BO})$. Note that the variable p_x should be a *business process* $\mathcal{BP}$. This illustrates the access relation from a *business process* to a business object. Conversely, the perspective of a business object seems to be irrelevant due to the assigning relationship. The initial marking is $M_0(p_s) = t_{BP,DO}$. The translation becomes $\pi_B(\mathcal{BO}) = \left((p_x, t_{BP,BO})\right)$.

Junction

$\pi_A(C_7) = \mathcal{J}^{\mathcal{N}}$
A *junction* $\mathcal{J}^{\mathcal{N}}$ in Petri Net is given by:
$\mathcal{J}^{\mathcal{N}} = \left(\left(p_x, (t_{J1}, t_{J2})\right), \left((t_{J1}, p_{y1}), (t_{J2}, p_{y2})\right)\right)$. This is the case for **splits** and the initial marking is $M_0(p_x) = t_{J1}, t_{J2}$. At least one incoming path should be activated in order to proceed. When paths are **joining**, the following Petri Net applies $\mathcal{J}^{\mathcal{N}} = \left(((p_{x1}, p_{x2}), t_J), (t_J, p_y)\right)$. At least one outcoming path should be activated to continue the flow. Looking at $((p_{x1}, p_{x2}), t_J)$,the initial marking of place p_x is $M_0(p_{x1}) = t_J$ and $M_0(p_{x2}) = t_J$.
For **splits** $\pi_A(\mathcal{J}^{\mathcal{N}}) = \left(\left(p_x, (t_{J1}, t_{J2})\right), \left((t_{J1}, p_{y1}), (t_{J2}, p_{y2})\right)\right)$.
For **joins** $\pi_A(\mathcal{J}^{\mathcal{N}}) = \left(((p_{x1}, p_{x2}), t_J), (t_J, p_y)\right)$.

AND-Junction

$\pi_A(C_8) = \mathcal{J}^{\mathcal{A}}$
An *AND-junction* $\mathcal{J}^{\mathcal{A}}$ in Petri Net is given by:
$\mathcal{J}^{\mathcal{A}} = \left((p_x, t_{AJ}), (t_{AJ}, (p_{y1}, p_{y2}))\right)$. This is the case of **splitting** flows. Assume that the initial marking is $M_0(p_x) = t_{AJ}$. The Petri Net formulation for **joining** two flows is given by $\mathcal{J}^{\mathcal{A}} = \left(((p_{x1}, p_{x2}), t_{AJ}), (t_{AJ}, p_y)\right)$. Assume that the initial markings are $M_0(p_{x1}) = t_{AJ}$ and $M_0(p_{x2}) = t_{AJ}$.
For **splits** $\pi_A(\mathcal{J}^{\mathcal{A}}) = \left((p_x, t_{AJ}), (t_{AJ}, (p_{y1}, p_{y2}))\right)$.
For **joins** the translation is $\pi_A(\mathcal{J}^{\mathcal{A}}) = \left(((p_{x1}, p_{x2}), t_{AJ}), (t_{AJ}, p_y)\right)$.

OR-Junction
$\pi_A(C_9) = \mathcal{J}^O$
An *OR-junction* $\mathcal{J}^O$ in Petri Net is given by:
$\mathcal{J}^O = \Big(\big(p_x, (t_{OR1}, t_{OR2})\big), \big((t_{OR1}, p_{y1}), (t_{OR2}, p_{y2})\big)\Big)$. This applies for **splits**. Assume that the initial marking is $M_0(p_x) = t_{OR1}, t_{OR2}$. The Petri Net formulation for **joining** two flows is described as $\mathcal{J}^O = \Big(\big((p_{x1}, t_{OR1}), (p_{x2}, t_{OR2})\big), \big((t_{OR1}, p_{y1}), (t_{OR2}, p_{y1})\big)\Big)$. Assume that the initial markings are $M_0(p_{x1}) = t_{OR1}$ and $M_0(p_{x2}) = t_{OR2}$.

So **splits** $\pi_A(\mathcal{J}^O) = \Big(\big(p_x, (t_{OR1}, t_{OR2})\big), \big((t_{OR1}, p_{y1}), (t_{OR2}, p_{y2})\big)\Big)$.
For **joins** $\pi_A(\mathcal{J}^O) = \Big(\big((p_{x1}, t_{OR1}), ((p_{x2}, t_{OR2})\big), \big((t_{OR1}, p_{y1}), (t_{OR2}, p_{y1})\big)\Big)$.

Business Activity
$\pi_A(C_{10}) = \mathcal{BP}^{\mathcal{A}}$
A *business activity* $\mathcal{BP}^{\mathcal{A}}$in Petri Net is given by:
$\mathcal{BP}^{\mathcal{A}} = \big((p_x, t_{ba}), (t_{ba}, p_y)\big)$. Note that variable p_x should be of type $\mathcal{BSE}^{\mathcal{E}}, \mathcal{BP}, \mathcal{BF}$ in (p_x, t_{ba}). It represents the related place, where the incoming flow starts from. The variable p_y in (t_{ba}, p_y) should be of type $\mathcal{BSE}^{\mathcal{E}}, \mathcal{BP}, \mathcal{BF}$. These places are graphically depicted as a dotted eclipse or circle. The initial marking is $M_0(p_x) = t_{ba}$.

A business activity is a **<u>specialization</u>** of a business process and inherits the property of its parent. Business activities are triggered by / or triggers a business event, a business function or other business processes. This leads to the translation $\pi_A(\mathcal{BP}^{\mathcal{A}}) = \big((p_x, t_{ba}), (t_{ba}, p_y)\big)$.

Trigger Relation
$\pi_A(C_{11}) = \mathcal{RT}^{\mathcal{T}}$
A *trigger relation* $\mathcal{RT}^{\mathcal{T}}$ is associated with *business elements* and therefore cannot be defined separately, as the relations in Petri Net do not differentiate.

Flow Relation
$\pi_A(C_{12}) = \mathcal{RT}^M$
A *flow relation* $\mathcal{RT}^M$ in comparison to other relation types can be translated into a Petri Net graph. Consider the flow relations as an *<u>exchange of messages</u>* between business elements. See the requirements of the flow relation type that shows the permitted incoming and outcoming flows. The places p_x and p_y can be traced by the definition of its constraints. $\mathcal{RT}^M = \big((p_x, t_{fr,m}), (t_{fr,m}, p_y)\big)$. Note that the places are depicted as a dotted eclipse or circle as the focus is on flow relations.

The initial marking of place p_x is $M_0(p_x) = t_{fr,m}$. This would lead to the following translation $\pi_A(\mathcal{RT}^{M}) = \big((p_x, t_{fr,m}), (t_{fr,m}, p_y)\big)$.

Access Relation

$\pi_A(C_{13}) = \mathcal{RT}^{\mathcal{C}}$

An *access relation* $\mathcal{RT}^{\mathcal{C}}$ is used to model access to passive elements e.g. business objects associated with *business processes* or *business functions*. An access relation is a structural relation where it seems not to be considered as an 'event' flow in terms of tokens in Petri Nets. Therefore it cannot be defined separately, as the relations in Petri Net do not differentiate.

Association Relation

$\pi_A(C_{14}) = \mathcal{RT}^{\mathcal{S}}$

An *association relation* $\mathcal{RT}^{\mathcal{S}}$ is associated with *business elements* and therefore cannot be defined separately, as the relation in Petri Net does not differentiate. An association indicates simply the 'passive' relationship between *business elements* without having tokens.

Grouping Relation

$\pi_A(C_{15}) = \mathcal{RT}^{\mathcal{G}}$

A *grouping relation* $\mathcal{RT}^{\mathcal{G}}$ is used for (de)composition and aggregation of *business elements*. Therefore a grouping relation concept in Petri Net is not possible to define separately, because it concentrates on the flow of 'events'. Petri Net considers the relationship between places and transitions as a directed graph in terms of sequence flows (whereby message flows is considered to be a special one), capturing the relation by means of directed arcs.

Table 4.3 ArchiMate business elements and its equivalent Petri Net modules

ArchiMate business elements		**Petri Net equivalent modules**
Actor	$\mathcal{BSE}^{\mathcal{A}}$	Not available (N.A.)
Role	$\mathcal{BSE}^{\mathcal{R}}$	Not available (N.A.)
Business event	$\mathcal{BBE}^{\mathcal{E}}$	$\pi_A(\mathcal{BBE}^{\mathcal{E}}) = \big((p_s, t_{be}), (t_{be}, p_y)\big)$ start events $\pi_A(\mathcal{BBE}^{\mathcal{E}}) = \big((p_x, t_{be}), (t_{be}, p_y)\big)$ interm. events $\pi_A(\mathcal{BBE}^{\mathcal{E}}) = ((p_x, t_{be}), (t_{be}, p_e))$ end events
Business process	$\mathcal{BP}$	$\pi_A(\mathcal{BP}) = \big((p_x, t_{bp}), (t_{bp}, p_y)\big)$
Business function	$\mathcal{BF}$	$\pi_A(\mathcal{BF}) = \big((p_x, t_{bf}), (t_{bf}, p_y)\big)$
Business object	$\mathcal{BO}$	$\pi_A(\mathcal{BO}) = ((p_x, t_{BO}))$
Junction	$\mathcal{J}^{\mathcal{N}}$	***Splits*** $\pi_A(\mathcal{J}^{\mathcal{N}}) = \Big(\big(p_x, (t_{J1}, t_{J2})\big), \big((t_{J1}, p_{y1}), (t_{J2}, p_{y2})\big)\Big)$

		Joins $\pi_A(\mathcal{J}^{\mathcal{N}}) = \Big(\big((p_{x1},p_{x2}),t_J\big),\big(t_J,p_y\big)\Big).$
AND-Junction	$\mathcal{J}^{\mathcal{A}}$	***Splits*** $\pi_A(\mathcal{J}^{\mathcal{A}}) = \Big(\big(p_x,t_{AJ}\big),\big(t_{AJ},(p_{y1},p_{y2})\big)\Big)$ ***Joins*** $\pi_A(\mathcal{J}^{\mathcal{A}}) = \Big(\big((p_{x1},p_{x2}),t_{AJ}\big),\big(t_{AJ},p_y\big)\Big)$
OR-Junction	$\mathcal{J}^{\mathcal{O}}$	***Splits*** $\pi_A(\mathcal{J}^{\mathcal{O}}) =$ $\Big(\big(p_x,(t_{OR1},t_{OR2})\big),\Big(\big(t_{OR1},p_{y1}\big),\big(t_{OR2},p_{y2}\big)\Big)\Big)$ ***Joins*** $\pi_A(\mathcal{J}^{\mathcal{O}}) =$ $\Big(\big(((p_{x1},t_{OR1}),((p_{x2},t_{OR2})\big),\Big(\big(t_{OR1},p_{y1}\big),\big(t_{OR2},p_{y1}\big)\Big)\Big)$
Business Activity <u>specialization</u> of $\mathcal{BP}$	$\mathcal{BP}^{\mathcal{A}}$	$\pi_A(\mathcal{BP}^{\mathcal{A}}) = \Big((p_x,t_{ba}),\big(t_{ba},p_y\big)\Big)$
Trigger relation	$\mathcal{RT}^{\mathcal{T}}$	Not available (N.A.)
Flow relation	$\mathcal{RT}^{\mathcal{M}}$	$\pi_A(\mathcal{RT}^{\mathcal{M}}) = \Big(\big(p_x,t_{fr,m}\big),\big(t_{fr,m},p_y\big)\Big).$
Access relation	$\mathcal{RT}^{\mathcal{C}}$	Not available (N.A.)
Association relation	$\mathcal{RT}^{\mathcal{S}}$	Not available (N.A.)
Grouping relation	$\mathcal{RT}^{\mathcal{G}}$	Not available (N.A.)

4.3 Translating BPMN and ArchiMate (Indirectly)

Previous sections 4.1 and 4.2 already show the ***direct*** translation to Petri Nets from both the BPMN and ArchiMate concepts. This section concerns about the ***indirect*** translation that is on the right side (bold) of the hypothesis:

1. $\pi_A(\mathcal{C}_A)$ $\equiv?$ $\boldsymbol{\pi_B(\tau_{A,B}(\mathcal{C}_A))}$
2. $\pi_B(\mathcal{C}_B)$ $\equiv?$ $\boldsymbol{\pi_A(\tau_{B,A}(\mathcal{C}_B))}$

Concepts in ArchiMate ($\mathcal{C}_A$) are taken for translation to BPMN equivalent concepts that is the $\tau_{A,B}(\mathcal{C}_A)$ part. Analogous to the *indirect* translation back starting from BPMN concepts ($\mathcal{C}_B$) and its equivalent ArchiMate concepts $\tau_{B,A}(\mathcal{C}_B)$. Then, the translation occurs arising from the result $\tau_{A,B}(\mathcal{C}_A)$ and $\tau_{B,A}(\mathcal{C}_B)$ as a starting point towards the semantics of Petri Nets in terms of places (p), transitions (t) and flows indicating the relationship between places and transitions (p,t).

4.3.1 BPMN to ArchiMate $\tau_{B,A}(B)$

This section shows the perspective from the BPMN language specific concepts. Table 4.4 summarizes the core concepts part that are essential in mapping concepts into the ArchiMate related business process concepts. Each concept then are further translated into Petri Nets.

An **event** in BPMN is defined as something that "happens" during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). This closely matches the **business event** concept in ArchiMate. However, BPMN defines a large number of specializations of the concept. The main specializations are **start event**, **stop event** and **intermediary events**, but several subtypes of all of these exist. This is in agreement with the focus of BPMN on detailed process modeling, which differs from ArchiMate's goal, i.e., to model the overall structure of an enterprise.

The notation are taken from section 4.2. The notation $\pi_{B'}$ means that $\pi_B\left(\tau_{B,A}(\mathcal{C})\right)$ is mapped to equivalent concepts in ArchiMate. Conversely, the notation $\pi_{A'}$ means that $\pi_A\left(\tau_{A,B}(\mathcal{C})\right)$ is mapped to equivalent concepts in BPMN.

Start Events

$\pi_B\left(\tau_{B,A}(\mathcal{C}_1)\right) = \mathcal{E}^{\mathcal{S}}$

$\tau_{B,A}(\mathcal{E}^{\mathcal{S}}) \equiv \mathcal{BBE}^{\mathcal{E}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{E}^{\mathcal{S}})\right) \equiv \pi_{B'}(\mathcal{BBE}^{\mathcal{E}})$

<u>*In Petri Net terms the translation is:*</u>

$\left((p_s, t_s)(t_s, p_y)\right) \equiv (p_s, t_{be}), (t_{be}, p_y)$

Intermediate Events

$\pi_B\left(\tau_{B,A}(\mathcal{C}_2)\right) = \mathcal{E}^{\mathcal{I}}$

$\tau_{B,A}(\mathcal{E}^{\mathcal{S}}) \equiv \mathcal{BBE}^{\mathcal{E}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{E}^{\mathcal{I}})\right) \equiv \pi_{B'}(\mathcal{BBE}^{\mathcal{E}})$

<u>*In Petri Net terms the translation is:*</u>

$\left((p_x, t_i)(t_i, p_y)\right) \equiv (p_x, t_{be}), (t_{be}, p_y)$

End Events

$\pi_B\left(\tau_{B,A}(\mathcal{C}_3)\right) = \mathcal{E}^{\mathcal{E}}$

$\tau_{B,A}(\mathcal{E}^{\mathcal{S}}) \equiv \mathcal{BBE}^{\mathcal{E}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{E}^{\mathcal{E}})\right) \equiv \pi_{B'}(\mathcal{BBE}^{\mathcal{E}})$

<u>*In Petri Net terms the translation is:*</u>

$((p_x, t_e)(t_e, p_e)) \equiv ((p_x, t_{be}), (t_{be}, p_e))$

An **activity** is a generic term for representing work that needs to be done in an organization. The types of an activity is either way atomic or non-atomic (i.e. compound activities). In ArchiMate terms, the actual work that must be performed (by actors and their associated roles), can be compared with the generic **business behaviour** concept. Specializations of the **activity** concept are (**sub-)processes** and **tasks**, defined as an (non-)atomic activity that is included within a process. In a similar way these concepts matches in succession to the **business process/function** and the **business activity** in ArchiMate.

Tasks

$\pi_B\left(\tau_{B,A}(\mathcal{C}_4)\right) = \mathcal{T}$

$\tau_{B,A}(\mathcal{T}) \equiv \mathcal{BP}^{\mathcal{A}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{T})\right) \equiv \pi_{B'}(\mathcal{BP}^{\mathcal{A}})$

<u>*In Petri Net terms the translation is:*</u>

$(p_x, t_{task}), (t_{task}, p_y) \equiv (p_x, t_{ba}), (t_{ba}, p_y)$

The semantics in Petri Nets of *sub-processes* and *business process* is the same, but syntactical it differs from each other in the graphical sense.

Sub-Processes

$\pi_B\left(\tau_{B,A}(\mathcal{C}_5)\right) = \mathcal{S}$

$\tau_{B,A}(\mathcal{S}) \equiv \mathcal{BP}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{S})\right) \equiv \pi_{B'}(\mathcal{BP})$

<u>*In Petri Net terms the translation is:*</u>

$$\left((p_x, t_{SPcall}), (t_{SPcall}, p_s)\right) \ldots \left((p_e, t_{SPreturn}), (t_{SPreturn}, p_y)\right)$$
$$\equiv$$
$$\left((p_x, t_{bp}), (t_{bp}, p_y)\right)$$

When take a closer look on business process this results in:

$$\left((p_x, t_{bp}), (t_{bp}, p_{x1}) \ldots (p_{x2}, t_{bp}), (t_{bp}, p_y)\right)$$

A **gateway** is used to control the ***divergence*** and ***convergence*** of sequence flow. Thus, it determines the splitting and joining of paths. The gateway concept is intended for detailed process modeling underlying BPMN's principle. BPMN defines several specializations types of the gateway concept. Icons within the diamond shape of the gateway indicate the type of flow control behavior that can be partly divided in to the following types of control: **exclusive gateway** performs the exclusive decision and merging, **inclusive gateway** performs decisions and merging and **parallel gateway** that performs forking and joining. Each type of control affects both the incoming and outgoing flow.

The **junction** relation in ArchiMate, which is used to model splits or joins of relations, can be seen as an abstraction of the gateway concept. Three types can be discerned: Junction, AND-Junction, OR-Junction. An important factor is that the gateway concept often implies behaviour. In the event that the behaviour are considered to be relevant, most obvious a concept mapping to a **business activity** might be used, possibly with multiple outgoing triggering relations or followed by a junction to express this.

Exclusive Gateways

$\pi_B\left(\tau_{B,A}(\mathcal{C}_7)\right) = \mathcal{G}^{\mathcal{E}}$

- *Joins*

$\tau_{B,A}(\mathcal{G}^{\mathcal{E}}) \equiv \mathcal{J}^{\mathcal{O}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{G}^{\mathcal{E}})\right) \equiv \pi_{B},(\mathcal{J}^{\mathcal{O}})$

In Petri Net terms the translation is:

$$((p_{x1},t_{EG1}),((p_{x2},t_{EG2})),\big((t_{EG1},p_{y1}),(t_{EG2},p_{y1})\big)$$
$$\equiv$$
$$((p_{x1},t_{OR1}),((p_{x2},t_{OR2})),\big((t_{OR1},p_{y1}),(t_{OR2},p_{y1})\big)$$

- *Splits*

$\tau_{B,A}(\mathcal{G}^{\mathcal{E}}) \equiv \mathcal{J}^{\mathcal{O}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{G}^{\mathcal{E}})\right) \equiv \pi_{B},(\mathcal{J}^{\mathcal{O}})$

In Petri Net terms the translation is:

$$(p_x,(t_{EG1},t_{EG2})),\big((t_{EG1},p_{y1}),(t_{EG2},p_{y2})\big)$$
$$\equiv$$
$$(p_x,(t_{OR1},t_{OR2})),\big((t_{OR1},p_{y1}),(t_{OR2},p_{y2})\big)$$

Parallel Gateways

$\pi_B\left(\tau_{B,A}(\mathcal{C}_8)\right) = \mathcal{G}^{\mathcal{P}}$

- *Joins*

$\tau_{B,A}(\mathcal{G}^{\mathcal{P}}) \equiv \mathcal{J}^{\mathcal{A}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{G}^{\mathcal{P}})\right) \equiv \pi_{B},(\mathcal{J}^{\mathcal{A}})$

In Petri Net terms the translation is:

$\big(((p_{x1},p_{x2}),t_{PG}),(t_{PG},p_y)\big) \equiv \big(((p_{x1},p_{x2}),t_{AJ}),(t_{AJ},p_y)\big)$

- *Splits*

$\tau_{B,A}(\mathcal{G}^{\mathcal{P}}) \equiv \mathcal{J}^{\mathcal{A}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{G}^{\mathcal{P}})\right) \equiv \pi_{B},(\mathcal{J}^{\mathcal{A}})$

In Petri Net terms the translation is:

$\big((p_x,t_{PG}),(t_{PG},(p_{y1},p_{y2}))\big) \equiv \big((p_x,t_{AJ}),(t_{AJ},(p_{y1},p_{y2}))\big)$

Inclusive Gateways

$\pi_B\left(\tau_{B,A}(\mathcal{C}_9)\right) = \mathcal{G}^{\mathcal{I}}$

- *Joins*

$\tau_{B,A}(\mathcal{G}^{\mathcal{I}}) \equiv \mathcal{J}^{\mathcal{N}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{G}^{\mathcal{I}})\right) \equiv \pi_{B},(\mathcal{J}^{\mathcal{N}})$

In Petri Net terms the translation is:

$\big(((p_{x1},p_{x2}),t_{IG}),(t_{IG},p_y)\big) \equiv \big(((p_{x1},p_{x2}),t_J),(t_J,p_y)\big)$

- *Splits*

$\tau_{B,A}(\mathcal{G}^{\mathcal{I}}) \equiv \mathcal{J}^{\mathcal{N}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{G}^{\mathcal{I}})\right) \equiv \pi_{B},(\mathcal{J}^{\mathcal{N}})$

In Petri Net terms the translation is:

$$\left(\left(p_x,(t_{IG1},t_{IG2})\right),\left((t_{IG1},p_{y1}),(t_{IG2},p_{y2})\right)\right)$$
$$\equiv$$
$$\left(p_x,(t_{J1},t_{J2})\right),\left((t_{J1},p_{y1}),(t_{J2},p_{y2})\right)$$

A **pool** is a *swimlane* and a graphical container for partitioning a set of activities from other pools, usually in the context of business-2-business situations. A **lane** is a sub-partition within a pool and will extend the entire length of the pool, either vertically or horizontally. Lanes are used to organize and categories activities. Pools and lanes can be used to group activities based on arbitrary criteria. In a similar way this matches the ***grouping relation*** in ArchiMate. However, they are most commonly used to represent the actors or roles that perform certain activities. In that case, they can be interpreted as the ***business actor*** *or* ***business role*** concept in ArchiMate. **Message flow** relations are the only relationship that are allowed to cross the boundary of pools to exchange messages to other pools. Placing activities *within* a pool or lane can be seen as a way to specify the ***assignment*** between ***behaviour and business roles and business actors*** in ArchiMate.

Pools & Lanes

$\pi_B\left(\tau_{B,A}(\mathcal{C}_{10})\right) = \mathcal{V}^{\mathcal{P}}$

$\tau_{B,A}(\mathcal{V}^{\mathcal{P}}) \equiv (\mathcal{BSE}^{\mathcal{A}} \cup \mathcal{BSE}^{\mathcal{R}} \cup \mathcal{RT}^{\mathcal{G}})$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{V}^{\mathcal{P}})\right) \equiv \pi_{B'}(\mathcal{BSE}^{\mathcal{A}} \cup \mathcal{BSE}^{\mathcal{R}} \cup \mathcal{RT}^{\mathcal{G}})$

Not Applicable (N.A) in Petri Net terms.

$\pi_B\left(\tau_{B.A}(\mathcal{C}_{11})\right) = \mathcal{V}^{\mathcal{L}}$

$\tau_{B,A}(\mathcal{V}^{\mathcal{L}}) \equiv (\mathcal{BSE}^{\mathcal{A}} \cup \mathcal{BSE}^{\mathcal{R}} \cup \mathcal{RT}^{\mathcal{G}})$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{V}^{\mathcal{L}})\right) \equiv \pi_{B'}(\mathcal{BSE}^{\mathcal{A}} \cup \mathcal{BSE}^{\mathcal{R}} \cup \mathcal{RT}^{\mathcal{G}})$

Not Applicable (N.A) in Petri Net terms.

BPMN defines three types of **artifacts**:
an **annotation** is a textual annotation that can be associated with any concept in order to provide additional information e.g., comments. There is no comparable separate concept in ArchiMate that can express this behaviour. A **data object** represents data that can be accessed by activities, which is very similar to the **business object** in ArchiMate. Finally, a **group artifact** can be used to group arbitrary concepts, that are in the same manner used as the ***grouping relationship*** in ArchiMate.

Data Objects and Groups

$\pi_B\left(\tau_{B,A}(\mathcal{C}_{12})\right) = \mathcal{Z}^{\mathcal{D}}$

$\tau_{B,A}(\mathcal{Z}^{\mathcal{D}}) \equiv \mathcal{BO}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{Z}^{\mathcal{D}})\right) \equiv \pi_{B'}(\mathcal{BO})$

$(p_s, t_{task,DO}) \equiv (p_x, t_{BP,BO})$

$\pi_B\left(\tau_{B,A}(\mathcal{C}_{13})\right) = \mathcal{Z}^{\mathcal{G}}$

$\tau_{B,A}(\mathcal{Z}^{\mathcal{G}}) \equiv \mathcal{RT}^{\mathcal{G}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{Z}^{\mathcal{G}})\right) \equiv \pi_{B'}(\mathcal{RT}^{\mathcal{G}})$

Not Applicable (N.A) in Petri Net terms.

A **sequence flow** is used to model the relations that are intended to display the order of activities that are being performed in a process. In ArchiMate the ***triggering relationship*** corresponds to the sequence flow that expresses the control flow in a quite similar way.

Sequence flow

$\pi_B\left(\tau_{B,A}(\mathcal{C}_{14})\right) = \mathcal{R}^{\mathcal{S}}$

$\tau_{B,A}(\mathcal{R}^{\mathcal{S}}) \equiv \mathcal{RT}^{\mathcal{T}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{R}^{\mathcal{S}})\right) \equiv \pi_{B'}(\mathcal{RT}^{\mathcal{T}})$

Not Applicable (N.A) in Petri Net terms.

A **message flow** is used to model the flow of messages between two entities that are prepared to send and receive them. In BPMN, two separate pools in a Business Process Diagram represents the two participants. In ArchiMate, the corresponding ***flow relation*** is used to express the exchange of messages.

Message flow

$\pi_B\left(\tau_{A,B}(\mathcal{C}_{15})\right) = \mathcal{R}^{\mathcal{M}}$

$\tau_{B,A}(\mathcal{R}^{\mathcal{M}}) \equiv \mathcal{RT}^{\mathcal{M}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{R}^{\mathcal{M}})\right) \equiv \pi_{B'}(\mathcal{RT}^{\mathcal{M}})$

In Petri Net terms the translation is:

(a) $(t_{task(x)}, t_{task(y)}) \cup (t_{SP(x)}, t_{SP(y)}) \cup (t_{task(x)}, t_{SP(y)}) \cup (t_{SP(x)}, t_{task(y)})$

(b) $(p_e, t_{task}) \cup (p_e, t_{SP})$

(c) $(t_{task}, p_s) \cup (t_{SP}, p_s)$

(d) (p_e, p_s)

(a’) $\left((p_x, t_{ba,m}), (t_{ba,m}, p_y)\right) \cup \left((p_x, t_{bp,m}), (t_{bp,m}, p_y)\right) \cup$
$\left((p_x, t_{ba,m}), (t_{bp,m}, p_y)\right) \cup \left((p_x, t_{bp,m}), (t_{ba,m}, p_y)\right)$

(b’) $(p_{be}, t_{ba}) \cup (p_{be}, t_{bp})$

(c’) $(t_{ba}, p_{be}) \cup (t_{bp}, p_{be})$

(d’) $(p_{be}, t_{be}) \cup (t_{be}, p_{be})$

$(a \wedge b \wedge c \wedge d) \equiv (a' \wedge b' \wedge c' \wedge d')$

An **association** is used to associate information with flow objects. Text and graphical non-flow objects can be associated with the flow objects. Depending on the exact use, this is represented by the **access** relation or the **association** relation in ArchiMate.

Association

$\pi_B\left(\tau_{A,B}(\mathcal{C}_{16})\right) = \mathcal{R}^{\mathcal{A}}$

$\tau_{B,A}(\mathcal{R}^{\mathcal{A}}) \equiv \mathcal{RT}^{\mathcal{S}}$ further translation would be $\pi_B\left(\tau_{B,A}(\mathcal{R}^{\mathcal{A}})\right) \equiv \pi_{B'}(\mathcal{RT}^{\mathcal{S}})$

Not Applicable (N.A) in Petri Net terms.

This resulted in the represented example BPMN model (see **Fig. 4.1**). It is clear that only the business layer can be modeled, since BPMN does not offer concepts for modeling the application or technical infrastructure. In contrast to ArchiMate, BPMN cannot model services either. In ArchiMate actors and roles are depicted with different separately concepts respectively 'business actor' and 'business role', while in BPMN the actors or roles that perform certain processes or activities can be graphically drawn by means of pools.

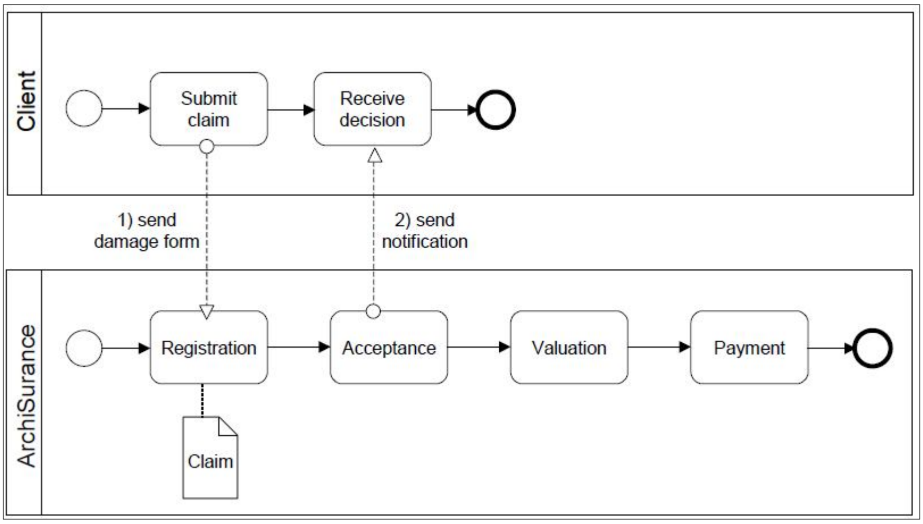
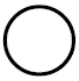
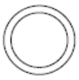
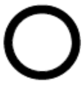
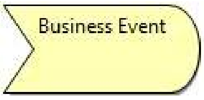
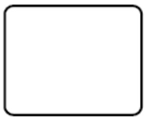
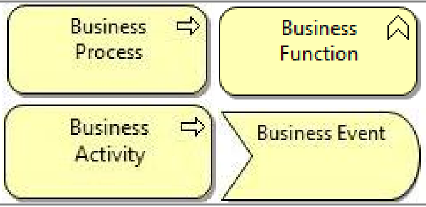
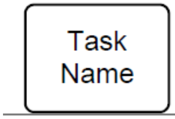
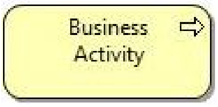
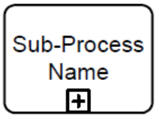
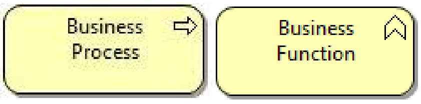
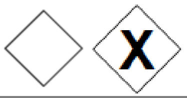
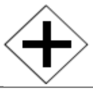
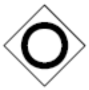
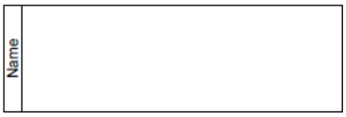
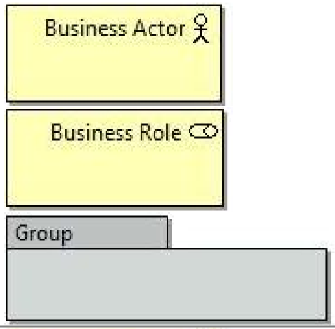
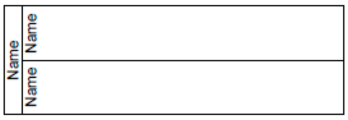
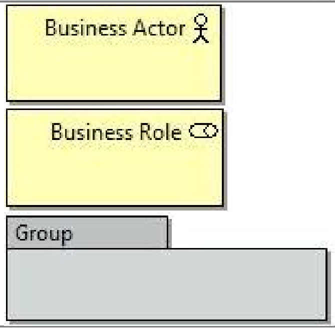
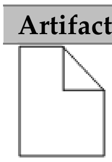
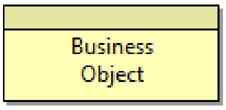
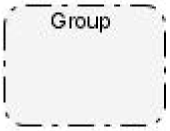
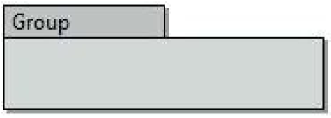


Fig. 4.1. Example model in BPMN.

BPMN models only restricted to the business layer, thus only the business architecture can be modeled as BPMN has lack of concepts for modeling applications or technical infrastructure with respect to ArchiMate. Another 'imperfection' is representing the services, which cannot be modeled either. The actors or roles that perform certain processes or activities can be shown by means of pools and therefore also in terms of lanes, because they are modeled *within* the pools. Moreover, there is a strong emphasis on the behaviour aspect; actors/roles that perform the behaviour that can be modeled in a limited way by means of pools and lanes. All main BPMN concepts have an equivalent concept in ArchiMate except the Annotation summarizes in Table 4.4, while on the other hand not all of the ArchiMate concepts, such as application and technology concepts have a counterpart in BPMN. However, BPMN has a large number of more specific concepts, which are defined as specializations of the main concepts. This is in agreement with the objectives of the two languages: while ArchiMate is designed to describe the high-level architecture of the whole enterprise, BPMN focuses on the detailed description of business processes.

Table 4.4. BPMN symbols related to equivalent ArchiMate symbols.

BPMN notation	ArchiMate equivalent notation
Events	**Business Elements**
 Start  Intermediate  End	
 Activity (*generic*)	 Business Process Business Function Business Activity Business Event
 Task (*specialization*)	
 Sub-Process (*specialization*)	
Gateways	**Junctions**
Exclusive Gateway 	OR-Junction □
Parallel Gateway 	AND-Junction ■
Inclusive Gateway 	Junction ●
Relationships	**Dynamic Relationships**
Sequence flow ——→	Triggering relation ——→
Message flow ○- - - - - - -▷	Flow relation - - - - - - - -→
Association ⋯⋯⋯⋯ ⋯⋯⋯⋯>	**Structural Relationships**
	Access relation; ⋯⋯⋯⋯>
	Association relation ———

Swimlanes	Business Elements / Relationships
Pools & Lanes	
Pool 	
Lane 	
Artifact	
 Data Object	
	
Annotation	Not Available (N.A)

4.3.2 ArchiMate to BPMN $\tau_{A,B}(A)$

Previous section summarizes the BPMN graphical notation and its equivalent ArchiMate notation. This section shows the perspective from the ArchiMate language concepts. Table 4.5 summarizes the core concepts part that are essential in mapping concepts into the BPMN related process concepts. Each concept then are further translated into Petri Nets.

A **business event** in ArchiMate is defined as something that "happens" and may influences business processes and business functions. Typically, a business event is most commonly used to model something that triggers behaviour. *Business processes* and other business behaviour may be triggered or interrupted by a *business event*.

Via specialization mechanisms, other types of events can be defined such as *message* or *time triggers events* that exchange information or interrupts a process. Furthermore, a business event is momentary, which means that the event occurred and then ceased. Business events may come from the environment of the organization for instance from a customer, but may also come from internal events that arise from for instance other processes within the organization. In line with this, *business processes* may raise events that trigger other *business processes and/or business functions*.

The business event closely matches the **event** type in BPMN, but specialize the main event type into **start events**, **intermediate events** and **end events**, as it concentrates on detailed process modeling.

Business Events

$\pi_A\left(\tau_{A,B}(\mathcal{C}_1)\right) = \mathcal{BBE}^{\mathcal{E}}$

$\tau_{A,B}(\mathcal{BBE}^{\mathcal{E}}) \equiv (\mathcal{E}^{\mathcal{S}} \cup \mathcal{E}^{\mathcal{I}} \cup \mathcal{E}^{\mathcal{E}})$ further translation would be formulated as

$\pi_A\left(\tau_{A,B}(\mathcal{BBE}^{\mathcal{E}})\right) \equiv \pi_{A'}(\mathcal{E}^{\mathcal{S}} \cup \mathcal{E}^{\mathcal{I}} \cup \mathcal{E}^{\mathcal{E}})$

In Petri Net terms the translation is:

$$\left(\left((p_s, t_{be}), (t_{be}, p_y)\right) \cup \left((p_x, t_{be}), (t_{be}, p_y)\right) \cup \left((p_x, t_{be}), (t_{be}, p_e)\right)\right)$$

$$\equiv$$

$$\left(\left((p_s, t_s), (t_s, p_y)\right) \cup \left((p_x, t_i), (t_i, p_y)\right) \cup \left((p_x, t_e), (t_e, p_e)\right)\right)$$

A **business process** can be used to group more detailed business processes (i.e. subprocesses) based on common grouping criteria. A business process can be seen as a set of interrelated business processes and/or business functions performed by a business role, with one or more clear starting points in terms of business events that leads to an expected or desired sets of ***products*** and ***services*** as a result.

Business processes might be sometimes referred to ‘customer-to-customer’ relation, where a ‘customer’ might originate from the external environment that triggers the business process or from an ‘internal customer’ in the case of subprocesses within an organization. For a consumer the required behaviour is not of interest so a process is designated "internal". In this sense, it most closely matches the specific **activity** concept including the underlying concepts **task** and **subprocess** in BPMN.

Business Processes

$\pi_A\left(\tau_{A,B}(\mathcal{C}_2)\right) = \mathcal{BP}$

$\tau_{A,B}(\mathcal{BP}) \equiv (\mathcal{T} \cup \mathcal{S})$ further translation would be formulated as

$\pi_A\left(\tau_{A,B}(\mathcal{BP})\right) \equiv \pi_{A'}(\mathcal{T} \cup \mathcal{S})$

<u>*In Petri Net terms the translation is:*</u>

$$\left(\left((p_x, t_{ba}), (t_{ba}, p_y)\right) \cup \left(\left((p_x, t_{bp}), (t_{bp}, p_s)\right) \dots \left((p_y, t_y), (t_y, p_y)\right)\right)\right)$$

$$\equiv$$

$$\left(\begin{array}{c} \left((p_x, t_{task}), (t_{task}, p_y)\right) \cup \\ \left(\left((p_x, t_{SPcall}), (t_{SPcall}, p_s)\right) \dots \left((p_e, t_{SPreturn}), (t_{SPreturn}, p_y)\right)\right) \end{array}\right)$$

A **business function** offers certain functionality that is useful for one or more business processes. A business function aggregates behaviour based on for instance required skills, knowledge, capabilities, resources and (application) support. In comparison with <u>***business processes***</u> that aggregate behaviour based on <u>***products***</u> and <u>***services***</u> that the organization offers, a business function serves the basis for the assignment of resources to tasks and the application support.

A *business function* may be triggered by, or trigger, any other business behaviour element (*business event*, *business process* or *business function*). Compared to the equivalent BPMN concept, a business function matches the **activity** concept, whereby <u>**task**</u> and <u>**subprocesses**</u> are types of atomic and non-atomic (i.e. compound) activities. Quite similar to the business process concept in ArchiMate, but with the addition that **pools** and **lanes** are involved due to the aggregation criteria and the assignment of resources to tasks.

Business Functions

$\pi_A\left(\tau_{A,B}(\mathcal{C}_3)\right) = \mathcal{BF}$

$\tau_{A,B}(\mathcal{BF}) \equiv (\mathcal{A} \cup \mathcal{V}^{\mathcal{P}} \cup \mathcal{V}^{\mathcal{L}})$ further translation would be formulated as

$\pi_A\left(\tau_{A,B}(\mathcal{BF})\right) \equiv \pi_{A'}\left((\mathcal{T} \cup \mathcal{S}) \cup \mathcal{V}^{\mathcal{P}} \cup \mathcal{V}^{\mathcal{L}}\right)$

<u>*In Petri Net terms the translation is:*</u>

$(p_x, t_{bf}), (t_{bf}, p_y)$

When take a closer look, i.e. the *internal* behaviour, on <u>*business functions*</u> this results in:

$$\left(\left((p_x, t_{ba}), (t_{ba}, p_y)\right) \cup \left((p_x, t_{bp}), (t_{bp}, p_s) \dots (p_y, t_y), (t_y, p_y)\right)\right)$$

$$\equiv$$

$$\left(\begin{array}{c} \left((p_x, t_{task}), (t_{task}, p_y)\right) \cup \\ \left(\left((p_x, t_{SPcall}), (t_{SPcall}, p_s)\right) \dots \left((p_e, t_{SPreturn}), (t_{SPreturn}, p_y)\right)\right) \end{array}\right)$$

A **business actor** in ArchiMate is defined as the active entities (i.e. 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 organizations.

It's important to separate the actor from the role because a business actor can perform more than one *business role*, and a *business role* can be performed by more than one *business actor*. *Business actors* are humans, departments, and business units; they may be individuals i.e. persons e.g., customers or employees or groups such as departments and business units.

Similarities are the **swimlane** concept in BPMN that are subdivided into **lanes**, used to organize and categories activities, and **pools**, a graphical container for partitioning a set of activities. Both mechanisms are inextricably linked or assigned to a **participant** as pools and lanes represent it therewith.

Business Actors

$\pi_A\left(\tau_{A,B}(\mathcal{C}_4)\right) = \mathcal{BSE}^{\mathcal{A}}$

$\tau_{A,B}(\mathcal{BSE}^{\mathcal{A}}) \equiv \mathcal{V}$ further translation would be formulated as

$\pi_A\left(\tau_{A,B}(\mathcal{BSE}^{\mathcal{A}})\right) \equiv \pi_{A'}(\mathcal{V}^{\mathcal{P}} \cup \mathcal{V}^{\mathcal{L}})$

Not Applicable (N.A) in Petri Nets.

A **business role** is defined as a specific behaviour of a business actor participating in a given context. The actor performs the behaviour of the role. A business role can be fulfilled by more than one business actor. Conversely, a business actor may fulfill more than one business role. A business role will usually exist in an organization whether or not a given actor fulfills it or not. A business role may be assigned to one or more business processes or business functions. BPMN does not explicitly discern an actor from a role, so **pool** and **lanes** concept covers the business role concept in ArchiMate at the same way.

Business Roles

$\pi_A\left(\tau_{A,B}(\mathcal{C}_5)\right) = \mathcal{BSE}^{\mathcal{R}}$

$\tau_{A,B}(\mathcal{BSE}^{\mathcal{R}}) \equiv \mathcal{V}$ further translation would be formulated as

$\pi_A\left(\tau_{A,B}(\mathcal{BSE}^{\mathcal{R}})\right) \equiv \pi_{A'}(\mathcal{V}^{\mathcal{P}} \cup \mathcal{V}^{\mathcal{L}})$

Not Applicable (N.A) in Petri Nets.

A **business object** is defined as passive entities that are manipulated by behaviour such as business processes or functions. Business objects represent the important concepts in which the business thinks about a domain, which is similar to data objects in BPMN. A *business object* is defined as a unit of information that has relevance from a business perspective. A *business object* is used to model an object type of which several instances may exist within the organization. In this case, it may be realized as a *data object*. It may also be specialized by another *business object*. Business objects are passive. They do not trigger or perform processes. Business objects are very similar to **data objects** in BPMN, which represents data that can be accessed by activities.

Business Objects

$\pi_A\left(\tau_{A,B}(\mathcal{C}_6)\right) = \mathcal{BO}$

$\tau_{A,B}(\mathcal{BO}) \equiv \mathcal{V}^{\mathcal{L}}$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{BO})\right) \equiv \pi_{A'}(\mathcal{Z}^{\mathcal{D}})$

<u>*In Petri Net terms the translation is:*</u>

$$\left(\left(p_x, t_{BP,BO}\right)\right) \equiv \left(\left(p_s, t_{task,DO}\right) \cup \left(t_{task,DO}, p_y\right)\right)$$

A **trigger relationship** describes the temporal or causal relations between processes, functions, interactions, and events. It is used to model the causal relationships between behavioral concepts in a process. Compared to BPMN, it matches the **sequence flow** relationship in BPMN.

Trigger Relationships

$\pi_A\left(\tau_{A,B}(\mathcal{C}_7)\right) = \mathcal{RT}^{\mathcal{T}}$

$\tau_{A,B}(\mathcal{RT}^{\mathcal{T}}) \equiv \mathcal{R}^{\mathcal{S}}$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{RT}^{\mathcal{T}})\right) \equiv \pi_{A'}(\mathcal{R}^{\mathcal{S}})$

Not Applicable (N.A) in Petri Nets.

A **flow relationship** describes the exchange or transfer of information or value between processes, function and events. Flow relationships are used to model the flow of information between behavioral concepts in a process. A flow relationship does not imply a causal or temporal relationship, while a trigger relationship does. The flow relationship matches the **message flow** relationship in BPMN.

Flow Relationships

$\pi_A\left(\tau_{A,B}(\mathcal{C}_8)\right) = \mathcal{RT}^{\mathcal{F}}$

$\tau_{A,B}(\mathcal{RT}^{\mathcal{F}}) \equiv \mathcal{R}^{\mathcal{M}}$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{RT}^{\mathcal{F}})\right) \equiv \pi_{A'}(\mathcal{R}^{\mathcal{M}})$

<u>*In Petri Net terms the translation is:*</u>

$$\left(\left(p_x, t_{fr,m}\right), \left(t_{fr,m}, p_y\right)\right)$$

$$\equiv$$

$$\left(\begin{matrix}\left(t_{task(x)}, t_{task(y)}\right) \cup \left(t_{SP(x)}, t_{SP(y)}\right) \cup \left(t_{task(x)}, t_{SP(y)}\right) \cup \left(t_{SP(x)}, t_{task(y)}\right) \cup \\ \left(p_e, t_{task}\right) \cup \left(p_e, t_{SP}\right) \cup \left(t_{task}, p_s\right) \cup \left(t_{SP}, p_s\right) \cup \left(p_e, p_s\right)\end{matrix}\right)$$

An **access relationship** models the access of behavioral concepts to *business* or *data objects*. The *access relationship* indicates that a process, function, interaction, service, or event "does something" with a (business or data) object. The arrow indicates the flow of information. There is no equivalent relationship in BPMN.

Access Relationships

$\pi_A\left(\tau_{A,B}(\mathcal{C}_9)\right) = \mathcal{RT}^{\mathcal{C}}$

$\tau_{A,B}(\mathcal{RT}^{\mathcal{C}})$ becomes $\pi_A\left(\tau_{A,B}(\mathcal{RT}^{\mathcal{C}})\right)$

NOTE: there exists not an equivalent relationship in BPMN

An **association relationship** models a relationship between objects that is not covered by another, more specific relationship. It is used to model relationships between *business objects* or *data objects* that are not modeled by the standard relationships. Association relationships match the same **association** relationships in BPMN.

Association Relationships

$\pi_A\left(\tau_{A,B}(\mathcal{C}_{10})\right) = \mathcal{RT}^{\mathcal{S}}$

$\tau_{A,B}(\mathcal{RT}^{\mathcal{S}}) \equiv \mathcal{R}^{\mathcal{A}}$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{RT}^{\mathcal{S}})\right) \equiv \pi_{A'}(\mathcal{R}^{\mathcal{A}})$

Not Applicable (N.A) in Petri Nets.

A **group relationship** indicates that objects, of the same type or different types, belong together based on some common characteristic. BPMN might group activities by means of **pools** and **lanes**, and by the **grouping** mechanism that groups activities based on arbitrary criteria. ArchiMate defines a concept that controls the relationships. A **junction** is used to connect dynamic relationships of the same type. A junction is used in a number of situations to connect dynamic (triggering or flow) relationships of the same type e.g., to indicate splits or joins. One condition is to ensure that only relationships of the same type (flow or triggering) are used to connect elements and junctions.

Group Relationships

$\pi_A\left(\tau_{A,B}(\mathcal{C}_{11})\right) = \mathcal{RT}^{\mathcal{G}}$

$\tau_{A,B}(\mathcal{RT}^{\mathcal{G}}) \equiv \mathcal{Z}^{\mathcal{G}}$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{RT}^{\mathcal{G}})\right) \equiv \pi_{A'}(\mathcal{Z}^{\mathcal{G}})$

Not Applicable (N.A) in Petri Nets.

ArchiMate defines three types of **junctions**:

1. **<u>Junctions</u>** either proceed if *<u>at least one</u>* in- or outcoming path is activated. A junction can serve as an AND-junction, but also as an OR-Junction. This is because activation of minimal one path splits or joins meets the condition.
2. **<u>AND-junctions</u>** whereby the condition satisfies, when *<u>all</u>* in- or outcoming paths are activated. So, an AND-junction only functions when all the requirements are met.
3. **<u>OR-junctions</u>** require *<u>only one</u>* of the in- or outcoming paths to be activated in order to proceed. OR-junctions are exactly the opposite way of AND-junctions.

A **gateway** in BPMN matches the **junction** concept in ArchiMate that controls the flow of both diverging and converging sequence flows. BPMN distinguishes many several types of gateways, whereas ArchiMate restricts to three types. Gateways are used to control how the process flows (i.e. flow of tokens) through **sequence flows** as they converge and diverge within a process. If the flow does not need to be controlled, then a gateway is not needed. The term gateway implies that there is a gate-mechanism that either allows or disallows passage through the gateway. As tokens arrive at a gateway, they can be merged together on input and/or split apart on output as the gateway mechanisms are invoked. Junctions are related to the combination of **exclusive and parallel gateways**. AND-junctions matches the **parallel gateway** and for the OR-junction the equivalent **inclusive gateway**.

Junctions

$\pi_A\left(\tau_{A,B}(\mathcal{C}_{12})\right) = \mathcal{J}^{\mathcal{N}}$

- *Joins*

$\tau_{A,B}(\mathcal{J}^{\mathcal{N}}) \equiv (\mathcal{G}^{\mathcal{I}})$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{J}^{\mathcal{N}})\right) \equiv \pi_{A'}(\mathcal{G}^{\mathcal{I}})$

<u>In Petri Net terms the translation is:</u>

$$\left(((p_{x1}, p_{x2}), t_J), (t_J, p_y)\right) \equiv \left(((p_{x1}, p_{x2}), t_{IG}), (t_{IG}, p_y)\right)$$

- *Splits*

$\tau_{A,B}(\mathcal{J}^{\mathcal{N}}) \equiv (\mathcal{G}^{\mathcal{I}})$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{J}^{\mathcal{N}})\right) \equiv \pi_{A'}(\mathcal{G}^{\mathcal{I}})$

<u>In Petri Net terms the translation is:</u>

$$\left(p_x, (t_{J1}, t_{J2})\right), \left((t_{J1}, p_{y1}), (t_{J2}, p_{y2})\right)$$
$$\equiv$$
$$\left((p_x, (t_{IG1}, t_{IG2})), \left((t_{IG1}, p_{y1}), (t_{IG2}, p_{y2})\right)\right)$$

AND-Junctions

$\pi_A\left(\tau_{A,B}(\mathcal{C}_{13})\right) = \mathcal{J}^{\mathcal{A}}$

- *Joins*

$\tau_{A,B}(\mathcal{J}^{\mathcal{A}}) \equiv (\mathcal{G}^{\mathcal{P}})$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{J}^{\mathcal{A}})\right) \equiv \pi_{A'}(\mathcal{G}^{\mathcal{P}})$

In Petri Net terms the translation is:

$$\left(\left((p_{x1},p_{x2}),t_{AJ}\right),\left(t_{AJ},p_y\right)\right) \equiv \left(((p_{x1},p_{x2}),t_{PG}),\left(t_{PG},p_y\right)\right)$$

- *Splits*

$\boldsymbol{\tau_{A,B}(\mathcal{J}^{\mathcal{A}})} \equiv (\boldsymbol{\mathcal{G}^{\mathcal{P}}})$ further translation would be as follows

$\boldsymbol{\pi_A\left(\tau_{A,B}(\mathcal{J}^{\mathcal{A}})\right)} \equiv \boldsymbol{\pi_{A'}(\mathcal{G}^{\mathcal{P}})}$

In Petri Net terms the translation is:

$$\boldsymbol{\left(p_x,t_{AJ}\right),\left(t_{AJ},(p_{y1},p_{y2})\right) \equiv \left((p_x,t_{PG}),\left(t_{PG},(p_{y1},p_{y2})\right)\right)}$$

OR-Junctions

$\pi_A\left(\tau_{A,B}(\mathcal{C}_{14})\right) = \mathcal{J}^{\mathcal{O}}$

- *Joins*

$\tau_{A,B}(\mathcal{J}^{\mathcal{O}}) \equiv \mathcal{G}^{\mathcal{E}}$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{J}^{\mathcal{O}})\right) \equiv \pi_{A'}(\mathcal{G}^{\mathcal{E}})$

In Petri Net terms the translation is:

$$\left(((p_{x1},t_{OR1}),((p_{x2},t_{OR2})),\left((t_{OR1},p_{y1}),(t_{OR2},p_{y1})\right)\right)$$
$$\equiv$$
$$\left(((p_{x1},t_{EG1}),(p_{x2},t_{EG2})),\left((t_{EG1},p_{y1}),(t_{EG2},p_{y1})\right)\right)$$

- *Splits*

$\tau_{A,B}(\mathcal{J}^{\mathcal{O}}) \equiv (\mathcal{G}^{\mathcal{E}})$ further translation would be as follows

$\pi_A\left(\tau_{A,B}(\mathcal{J}^{\mathcal{O}})\right) \equiv \pi_{A'}(\mathcal{G}^{\mathcal{E}})$

In Petri Net terms the translation is:

$$\left((p_x,(t_{OR1},t_{OR2})),\left((t_{OR1},p_{y1}),(t_{OR2},p_{y2})\right)\right)$$
$$\equiv$$
$$\left((p_x,(t_{EG1},t_{EG2})),\left((t_{EG1},p_{y1}),(t_{EG2},p_{y2})\right)\right)$$

A **business activity** in ArchiMate is defined as a specialization of a more generic ***business process***. BPMN used therefore the **task** concept, which is a specialization of the ***activity*** concept. A task can be an atomic-activity or non-atomic activity (i.e. compound).

Business Activities

$\pi_A\left(\tau_{A,B}(\mathcal{C}_{15})\right) = \mathcal{BP}^{\mathcal{A}}$

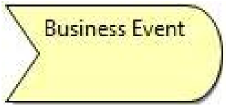
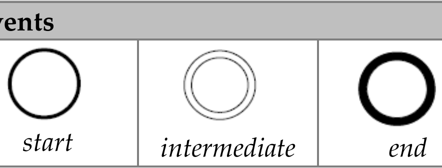
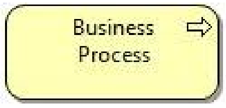
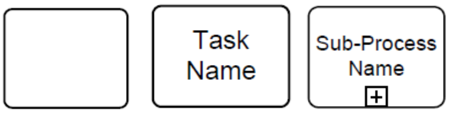
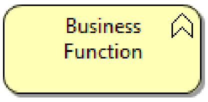
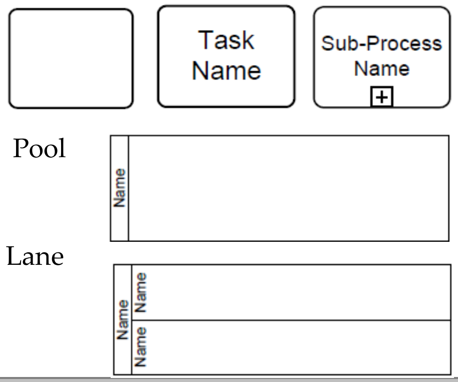
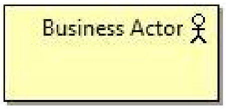
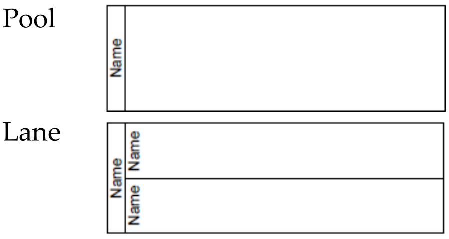
$\tau_{A,B}(\mathcal{BP}^{\mathcal{A}}) \equiv \mathcal{T}$ further translation would be as follows

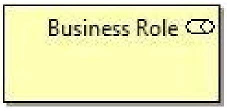
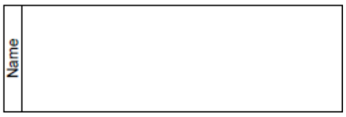
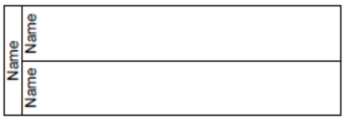
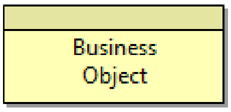
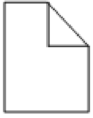
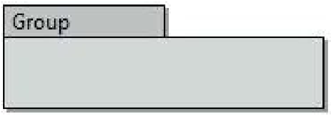
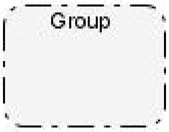
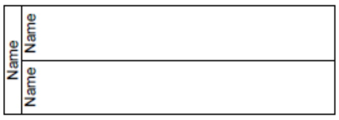
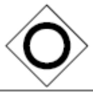
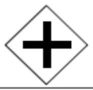
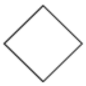
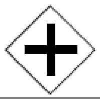
$\pi_A\left(\tau_{A,B}(\mathcal{BP}^{\mathcal{A}})\right) \equiv \pi_{A'}(\mathcal{T})$

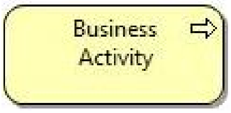
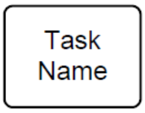
In Petri Net terms the translation is:

$\left((p_x,t_{ba}),(t_{ba},p_y)\right) \equiv \left((p_x,t_{task}),(t_{task},p_y)\right)$

Table 4.5. ArchiMate symbols related to equivalent BPMN symbols.

ArchiMate graphical notation	BPMN equivalent graphical notation
Structure/Behaviour/Passive elements	**Events**
	
	**Activities, Pools & Lanes** 
	 Task Name Sub-Process Name Pool Name Lane Name Name Name
	**Pools & Lanes** 

	Pool  Lane 
	**Data Objects** Data Object 
Relationships	**Relationships**
Dynamic Relationships	
Triggering relation ──────→	Sequence flow ──────→
Flow relation - - - - - - - →	Message flow ○– – – – – – – ▷
Structural Relationships	
Access relation ⋯⋯⋯⋯⋯⋯>	Not Available (N.A.)
Association relation ──────	Association ⋯⋯⋯⋯⋯⋯ ⋯⋯⋯⋯⋯⋯>
Other Relationships	**Groups / Pools & Lanes**
Grouping	
	 Pool Lane 
Junction	**Gateways**
Junction ●	Inclusive Gateway 
AND-Junction ■	Parallel Gateway 
OR-Junction □	Exclusive Gateway  

Specialization	**Activities**
	

4.4 Comparing Concepts via Operational Semantics

Previous section 4.2 defined the concept mapping directly from both ArchiMate as well as BPMN to Petri Net, while section 4.3 is done indirectly via BPMN/ArchiMate as intermediary language. This section compares both ArchiMate and BPMN languages via operation semantics. This means that the semantic of BPMN after translating via ArchiMate are compared to the direct translation of BPMN to Petri Nets. Conversely, ArchiMate concepts after translating via BPMN are compared to the directly concept mapping from ArchiMate to Petri Nets.

4.4.1 A Concept from ArchiMate to BPMN

Now an ArchiMate concept is taken to give a first attempt and serves as an starting point for other concepts that should be elaborated in the same way. As the business concept in practice most commonly used, the comparison would be as follows:

Business Processes **($\mathcal{BP}$)**	
Direct Mapping $\pi_A(\mathcal{BP})$	$\begin{pmatrix} \big((p_x, t_{bp}), (t_{bp}, p_y)\big) \\ \cup \\ \Big(\big(((p_x, t_{bp}), (t_{bp}, p_{x1})\big) \ldots \big((p_{x2}, t_{bsp}), (t_{bp}, p_y)\big)\Big) \end{pmatrix}$
≡	≡
$\pi_B(\tau_{A,B}(\mathcal{BP}))$ *Indirect Mapping*	$\begin{pmatrix} \big((p_x, t_{task}), (t_{task}, p_y)\big) \\ \cup \\ \Big(\big(((p_x, t_{SPcall}), (t_{SPcall}, p_s)\big) \ldots \big((p_e, t_{SPreturn}), (t_{SPreturn}, p_y)\big)\Big) \end{pmatrix}$

In Petri Net the semantics are matching as the business process in ArchiMate may comprises multiple 'smaller' business processes i.e., a *subprocess* denoted with $\Big(\big(((p_x, t_{bp}), (t_{bp}, p_{x1})\big) \ldots \big((p_{x2}, t_{bsp}), (t_{bp}, p_y)\big)\Big)$ and a process without considering subprocesses simplified by $\big((p_x, t_{bp}), (t_{bp}, p_y)\big)$. The result of the ***indirect mapping*** of a *business process*, from the perspective of ArchiMate via BPMN, semantically means the same in Petri Net terms, but the HLPNG might differ from each other in terms of the number of places transitions and arcs. Unfolding the subprocess in BPMN shows the internal behaviour of activities that are translated into the following way:

$((p_x, t_{SPcall}), (t_{SPcall}, p_s)) \dots \left((p_e, t_{SPreturn}), (t_{SPreturn}, p_y)\right)$, that invokes the actual subprocess $(t_{SPcall}, p_s) \dots (p_e, t_{SPreturn})$ in the BPD. Simplifying this subprocess leads to $\left((p_x, t_{process}), (t_{process}, p_y)\right)$ (see **Fig. 4.5**) which matches the simplified version of a business process in ArchiMate.

The example (see **Fig. 4.2**) shows the process for handling claims in an insurance company called 'ArchiSurance'. The business process concept '*Handle Claim*' contains the business concepts '*Register*', '*Accept*', '*Valuate*', '*Pay*' and '*Reject*' concept. Clearly, these concepts together represents the *Handle Claim* concept. If damage occurs, then it triggers the *Handle Claim* business process as a whole and the internal business process *Register* is receiving the damage by register the occurred damage of the insurer.

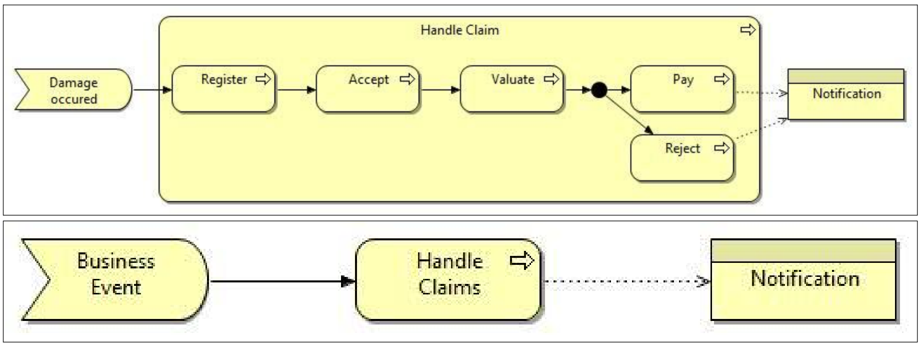


Fig. 4.2. Example of a business process for handle claims.

Concepts		*Places (P)*	*Transitions (T)*	*Arcs (P,T) (F)*
Handle Claim	=	$\boldsymbol{p_0}$	t_0, t_1	$\left(\left((p_1, t_1), (t_1, p_0)\right), \left((p_0, t_0), (t_0, p_7)\right)\right)$
Damage occurred	=	$\boldsymbol{p_1}$	t_1	$\left((p_1, t_1), (t_1, p_2)\right)$
Register	=	$\boldsymbol{p_2}$	t_1, t_2	$\left(\left((p_1, t_1), (t_1, p_2)\right), \left((p_2, t_2), (t_2, p_3)\right)\right)$
Accept	=	$\boldsymbol{p_3}$	t_2, t_3	$\left(\left((p_2, t_2), (t_2, p_3)\right), \left((p_3, t_3), (t_3, p_4)\right)\right)$
Valuate	=	p_3 $\boldsymbol{p_4}$ p_5 p_6	t_3 t_{J1} t_{J2}	$\left(\begin{matrix} ((p_3, t_3), (t_3, p_4)), \\ \left(\begin{matrix} \left(p_4, (t_{J1}, t_{J2})\right), \\ \left((t_{J1}, p_5), (t_{J2}, p_6)\right) \end{matrix}\right) \end{matrix}\right)$
Pay	=	$\boldsymbol{p_5}, p_7$	t_4	$\left((p_5, t_4), (t_4, p_7)\right)$
Reject	=	$\boldsymbol{p_6}, p_7$	t_5	$\left((p_6, t_5), (t_5, p_7)\right)$
Notification	=	p_5 p_6 $\boldsymbol{p_7}$	t_4, t_5	$\left(\left((p_5, t_4), (t_4, p_7)\right), \left((p_6, t_5), (t_5, p_7)\right)\right)$

Junction (SPLIT)	=	p_4, p_5, p_6	t_{J1}, t_{J2}	$\Big(\big(p_4, (t_{J1}, t_{J2})\big), \big((t_{J1}, p_5), (t_{J2}, p_6)\big)\Big)$

Markings of a net

The set of all initial place markings of a net:

$M_{(p0)}$= $M_{(p1)}, M_{(p6)}$

$M_{(p1)}$= the damage that can be claimed according to the insurance policy.

$M_{(p2)}$= detailed damage info, the insurance policy data of the insurer.

$M_{(p3)}$= preparing data for further processing.

$M_{(p4)}$= requirements with respect to compensation in the insurance policy.

$M_{(p5)}$= the amount is reimbursed via the bank account of the insurer.

$M_{(p6)}$= requirements of damage coverage according to the insurance policy.

$M_{(p7)}$= a letter that states the motivation whether the damage is being compensated or the insurer needs to compensate the damage by himself as a consequence of his insufficient damage coverage insurance policy.

ArchiSurance Model in Petri Net terms
for handle claim concept
$\Big(\big((p_1, t_1), (t_1, p_0)\big), \big((p_0, t_0), (t_0, p_7)\big)\Big)$
for internal concepts within the handle claim concept
$\left(\begin{matrix} \big((p_1, t_1), (t_1, p_2)\big), \big((p_2, t_2), (t_2, p_3)\big), \big((p_3, t_3), (t_3, p_4)\big), \left(\begin{matrix} \big(p_4, (t_{J1}, t_{J2})\big), \\ \big((t_{J1}, p_5), (t_{J2}, p_6)\big) \end{matrix}\right), \\ \big((p_5, t_4), (t_4, p_7)\big), \big((p_6, t_5), (t_5, p_7)\big) \end{matrix}\right)$

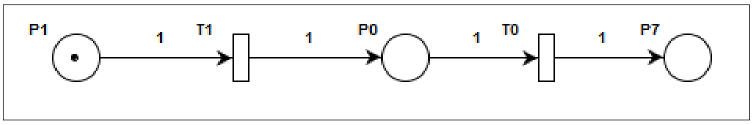


Fig. 4.3. The Petri Net graph of the business process for handle claims.

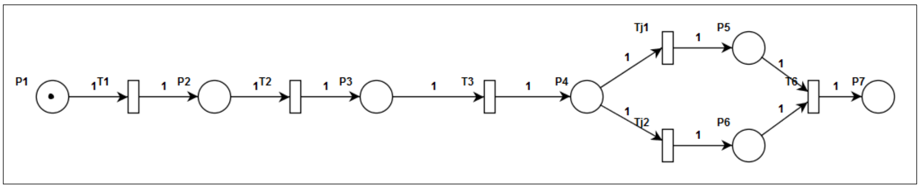


Fig. 4.4. The Petri Net graph extended business process for handle claims.

4.4.2 Some Concepts from BPMN to ArchiMate

This section reflects the opposite way of the previous section and takes some BPMN specific concepts that is equivalent to the business process concept in ArchiMate. Now, the activity (*task, process, subprocess*) concept is compared. Due to time limitations, this section only provides the comparison, but should be done in the same way as described in section 4.4.1. The ArchiSurance model in the previous section is equivalent with the BPMN Diagram (see **Fig. 4.5 and 4.6**).

Tasks		
Direct Mapping	$\pi_B(\mathcal{T})$	$\big((p_x, t_{task}), (t_{task}, p_y)\big)$
	$\equiv$	$\equiv$
Indirect Mapping	$\pi_A(\tau_{B,A}(\mathcal{T}))$	$\big((p_x, t_{ba}), (t_{ba}, p_y)\big)$

Sub-Processes	
$\pi_B(\mathcal{S})$	$\Big(\Big(\big((p_x, t_{SPcall}), (t_{SPcall}, p_s)\big) \ldots \big((p_e, t_{SPreturn}), (t_{SPreturn}, p_y)\big)\Big)\Big)$
$\equiv$	$\equiv$
$\pi_A(\tau_{B,A}(\mathcal{S}))$	$\big((p_x, t_{bp}), (t_{bp}, p_{x1}) \ldots (p_{x2}, t_{bp}), (t_{bp}, p_y)\big)$

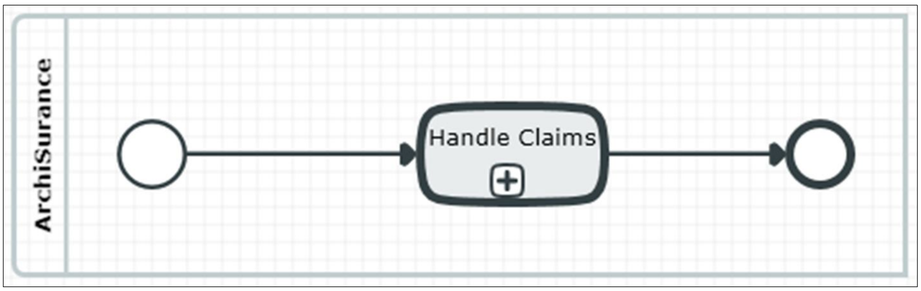


Fig. 4.5. The BPMN Diagram of the business process for handle claims.

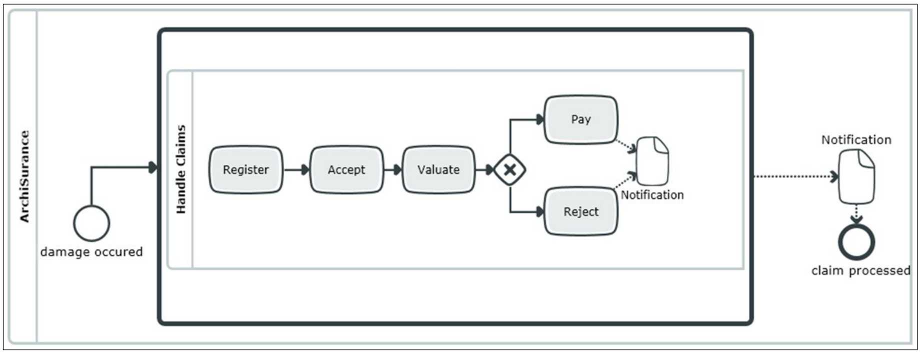


Fig. 4.6. The BPMN Diagram of the handle claims subprocess.

4.5 Expressiveness

Most modeling languages are designed to model or describe a specific architectural domain such as applications e.g. UML or business e.g. BPMN. The expressiveness of such modeling languages is related to its concepts [Prop 05]. ArchiMate is designed to describe the high-level architecture of the enterprises architecture, while BPMN focuses on the detailed description of business processes. ArchiMate concepts are thus of general nature, while concepts of BPMN are more designed for detailed process modeling.

4.5.1 Frameworks (AM – BPMN)

ArchiMate focuses on the architecture of the enterprise which encompasses *business*, *application* and *technology* domains, whereas BPMN concentrate on process modeling within the business domain.

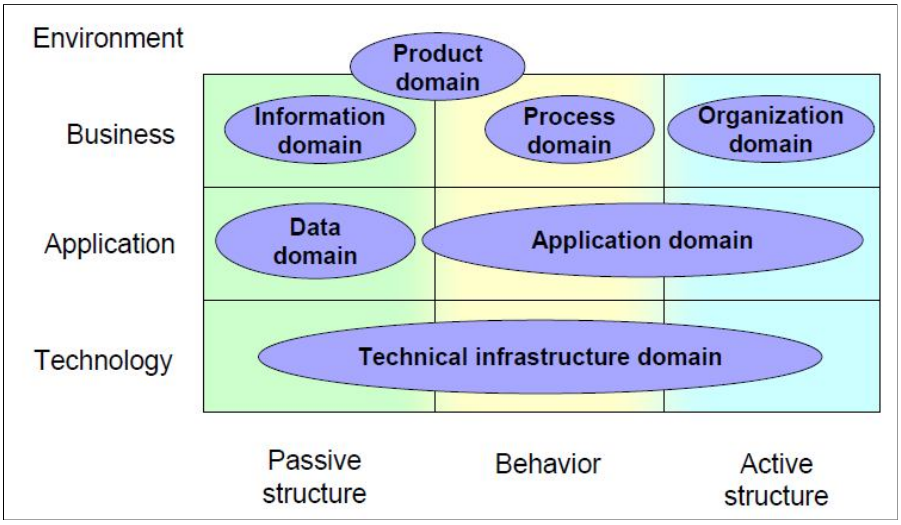


Fig. 4.4. ArchiMate's framework covering integrated architectural domains.

An architectural framework of the ArchiMate language (see **Fig. 4.4**) defines the structure of the concepts and its mutual relationship into the business, application and technology layer, in which the passive, behaviour and active aspects are an essential part. One of the driving forces to develop the ArchiMate language is to define any global structure *within* each domain and to define the relevant relations *between* the domains. Concepts defined in the ArchiMate language are intended for the integration of business, application, and technology domains; the detailed concepts that are used for modeling specific domains, such as UML for modeling software architecture and BPMN that is used for business process modeling, which can be closely related to ArchiMate concepts. In the ArchiMate's framework, BPMN covers only the process domain (see **Fig. 4.5**). BPMN is restricted to the business layer with a strong emphasis on the behavioral aspect. BPMN's main purpose is to provide a uniform notation for modeling business processes in terms of activities and their relationships. The language specific concepts: *pool, lane* and *artifact* do not cover the structural and informational aspects associated with the business processes. Thus, the BPMN concepts offer limited possibilities with respect to both the informational and structural aspects, due to its business specific detailed concepts.

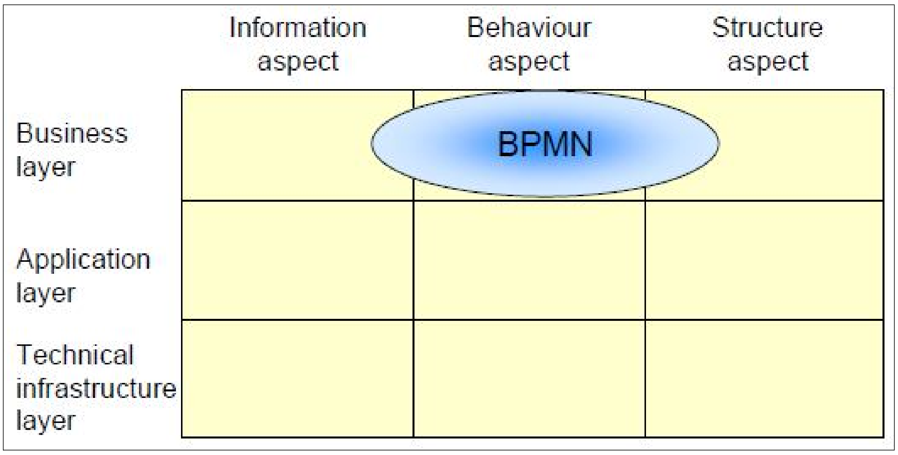


Fig. 4.5. BPMN's framework covers the process domain.

4.5.2 Business Processes

BPMN can be used for modeling detailed business processes as it is specifically designed for process modeling. ArchiMate models the global structure, i.e. high level architecture *within* and *between* architectural domains. The expressiveness of concepts from these languages are evaluated with respect to modeling business processes. To this end, only the business layer of the ArchiMate language is abstracted, since these layer-specific concepts are closely related to BPMN basic concepts.

ArchiMate contains a considerable number of concepts and relationships [Jonk 04]. Which relationships may be modeled between two concepts is precisely defined in ArchiMate, but in practice it is difficult to find the right choice within the permissible purposes. The question is what relationships between concepts in most cases should be modeled. This section describes the most common situations in which relationships should be modeled. Not all relationships between ArchiMate concepts discussed, only the relationships that are most commonly used.

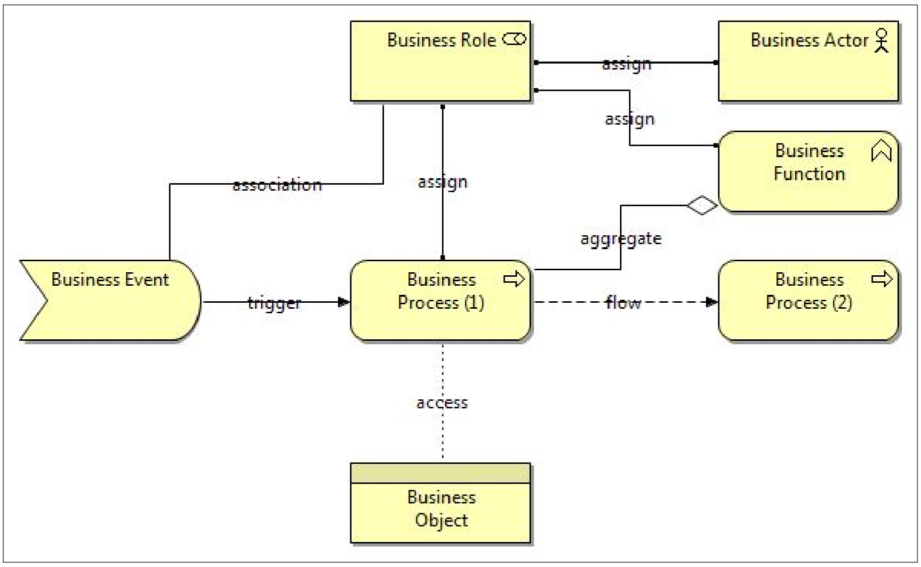


Fig. 4.6. Expressiveness of ArchiMate business process concept.

Business process has the following relationship with its associated concepts (see **Fig. 4.6**):

- ✓ A business process exchange data with other business processes via the flow relation;
- ✓ A business process is triggered by / or triggers a business event, a business function or other business processes;
- ✓ A business process is assigned to a business role;
- ✓ A business process is part of a business function;
- ✓ A business process has access to a business object; a business process creates, reads, edit or destroy a business object.

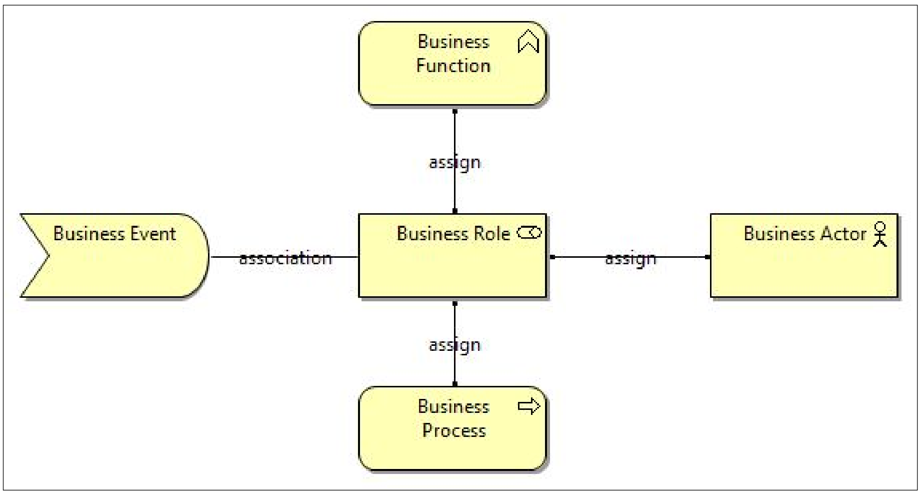


Fig. 4.7. Relationship of the business role concept.

Business role has the following relationship with its associated concepts (see **Fig. 4.7**):

- ✓ A business role can be assigned to a business actor;
- ✓ A business role can be assigned to a business process, a business function or associated with a business event;

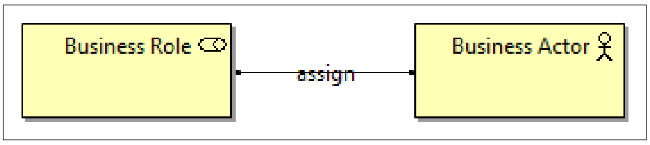


Fig. 4.8. Relation between business role and business actor concept.

Business actor has the following relationship with its associated concepts (see **Fig. 4.9)**:

- ✓ A business actor is assigned to a business role;

Business object has the following relationship with its associated concepts (see **Fig. 4.10**):

- ✓ A business object is created, read, edited, or removed by a business process or business function via the access relation.
- ✓ A business object can have specializations;
- ✓ A business object can refer to other objects (aggregation relation)
- ✓ A business object may contain other objects (composition relation)

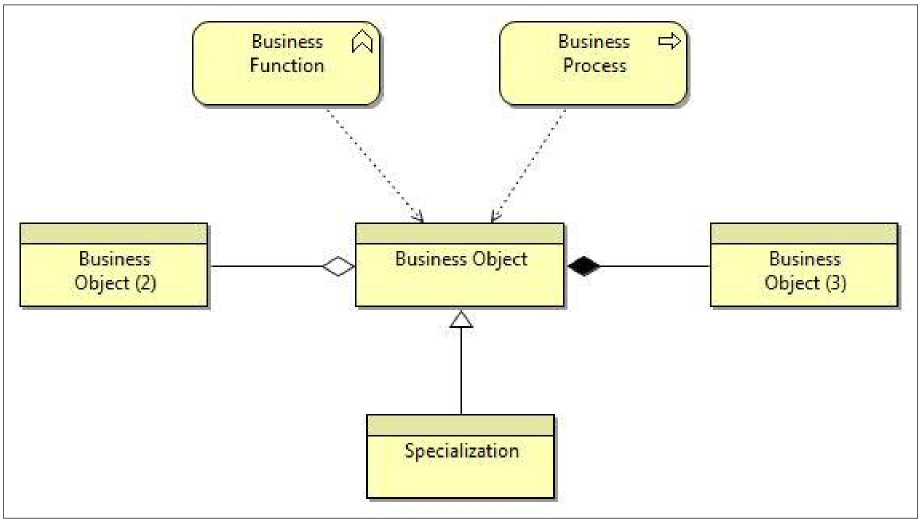


Fig. 4.10. Relationship of the business object concept.

NOTE: For BPMN, only *description of parts of the relevant concepts* with respect to the expressiveness are given.

Task has the following relationship with its associated concepts:

- ✓ A task is a specialization of the generic activity concept that represents the *smallest peace of work* in an organization that needs to be performed;
- ✓ A task may associate ***pools*** (including ***lanes***) for denoting sources (i.e. participants) that are responsible for the tasks to be performed;
- ✓ A task triggers/may be triggered by other ***processes***, ***tasks*** or ***subprocesses*** by using the ***sequence flow*** relation;
- ✓ Tasks are atomic or non-atomic activities.

Subprocess has the following relationship with its associated concepts:

- ✓ A subprocess is also a specialization of an activity concept and represents the detailed internal activities in terms of ***tasks, processes***, and ***subprocesses***;
- ✓ Subprocess itself may associate participants by the use of ***pools*** (including ***lanes***);
- ✓ A subprocess triggers/may be triggered by other ***tasks***, ***processes***, or ***subprocesses*** by using the ***sequence flow*** relation;
- ✓ Subprocesses are atomic or non-atomic activities.

Pools & Lanes has the following relationship with its associated concepts:

- ✓ A swimlane concept is a generic term for assigning (human or systems) resources to activities that are responsible for performing the work flow;
- ✓ Pools may comprise ***lanes*** and pools mutually exchange messages by the ***message flow relations***;
- ✓ Pools can contain ***tasks***, ***processes*** and ***subprocesses***;
- ✓ Lanes are subparted of pools and can contain ***tasks***, ***processes*** and ***subprocesses***.

Data objects has the following relationship with its associated concepts:

- ✓ Data objects represents the information that are accessed by activities via ***message flow relations***.
- ✓ Data objects may exchange from *message* ***start events***, ***tasks*** and ***(sub)processes***.

Chapter 5

Discussions

This section discusses the resulting Petri Nets semantics originated from ArchiMate and BPMN concepts. Some clues are discussed to enhance the semantic modeling approach that are of relevant value for analyzing the behaviour of nets [Bern 02] in future work.

Semantics of concepts in business processes:

The ArchiMate business process can be seen as a generalization of the BPMN language, while BPMN is designed for a detailed business process with specific business concepts. Some concepts cannot be translated such as *business actor*, *business roles* in ArchiMate and the *pools and lanes* in BPMN. Such incomplete translation of ArchiMate/BPMN business concepts means in essence loss of information, but that is the price to be paid. Also the grouping mechanism in both languages needs to be compensated to some extent.

These issues are abstracted in Petri Net terms as the focus lies on ***event processing*** [Zang 08],[Shen 04] by means of *activities and processes* (i.e. the actual work). Besides, the transformation of relations is a difficult one, because a message flow in BPMN means an exchange of messages between *pools*, while in ArchiMate a flow relation in graphical sense does not differentiate from other relations. Since Petri Net does not differentiate relationships, a BPMN message flow is interpreted as an event process. Thus, as long the business concepts are reflecting behavior of activities, such constructs can be modeled.

A formal definition of the mapping of BPMN/ArchiMate to Petri Net (let's say *P'* for places, *T'* for transitions and *F'* for flow relations) defined in terms of ArchiMate and BPMN formal semantics, should be provided that contributes to a better understanding.

Analyzing Petri Nets (HLPNG's) to determine its behaviour properties:

For future work the semantics of ArchiMate/BPMN models in Petri Net terms should be analyzed [Khom 07] by the following properties to obtain useful information (by tools) about the behaviour of such models being created in ArchiMate and BPMN within the *business* domain and transformed to nets:

- <u>*Reachability*</u>
  For a directed (acyclic) graph (DAG), the ***reachability*** can be calculated to detect whether certain states cannot be reached with respect to place markings. The reachability set of a net is the set of all markings reachable from initial markings. This might be very helpful to find erroneous states in the net with having many places and transitions.

- <u>*Liveness*</u>
  Liveness of nets is one of the property in Petri Net to detect whether transitions can be 'fired' in the sense that transitions are activated. A transition is *deadlocked* if it can never fire and a transition is *live* if it can never deadlock. By doing this, the relevant transitions behaviour of the net can be traced back to ArchiMate/BPMN models.

- <u>*Boundedness*</u>
  The ***boundedness*** of a net concerns the distribution of tokens with regards to markings of nets. A Petri Net with initial markings (i.e. the set of all place markings) is **safe** if places always hold at most 1-token. A marked net is **(k-)bounded** if places never hold more than *k* tokens. A marked net is **conservative** if the number of tokens is constant. When taken this property in account, the development of the 'work' can be seen during its performance and might facilitate the decision making processes.

- <u>*Number of places, transitions, and arcs*</u>
  A desirable feature is to calculate how many *places*, *transitions* and *arcs* are used to construct the Petri Net model resulting from equivalent ArchiMate/BPMN models. It is interesting to evaluate how empirical data is related to the semantic of the ArchiMate/BPMN models.

Chapter 6

Conclusions

This thesis proposes a mapping from a relevant set of core ArchiMate and BPMN concepts to Petri Net in order to test the hypothesis in how these concepts are related with respect to their semantics. Based on the results of the modelling approach the hypothesis are evaluated.

Hypotheses

1. $\pi_A(\mathcal{C}_A) \equiv \pi_B(\tau_{A,B}(\mathcal{C}_A))$
2. $\pi_B(\mathcal{C}_B) \equiv \pi_A(\tau_{B,A}(\mathcal{C}_B))$

The results of the comparing method (see **Sect. 4.4**) can be divided in to the following argumentation. This concerns the mapping concepts *indirectly* given an specific concept in ArchiMate/BPMN that leads to **elementary** or **composite concepts** in BPMN/ArchiMate.

- ***Elementary concepts***
  Business Activity *concept in ArchiMate and the BPMN* ***Task*** *concepts are matching. These concepts are at both sides elementary, i.e.,* $\tau_{A,B}(\mathcal{BP}^{\mathcal{A}})$ *and* $\tau_{B,A}(\mathcal{T})$*, because the semantics are identical and the HLPNG are reflecting each other. Thus, it can be concluded that when taking a* ***business activity*** *concept in ArchiMate that is directly mapped into Petri Nets, the semantics of* $\pi_A(\mathcal{C}_A)$ *is identical to the indirect mapping of a business process concept* $\pi_B(\tau_{A,B}(\mathcal{C}_A)$ *from the perspective of ArchiMate via BPMN as intermediary language.*

- ***Composites concepts***
  The business process concept in ArchiMate is mapped to BPMN resulting in ***composite concepts*** *of respectively* ***(activity) task*** *and* ***subprocess*** *concepts. This arises the question how one concept (in this case the business process concept) in ArchiMate is related to equivalent concept from the BPMN perspective and vice versa.*

If composite concepts are not found in the direct mapping of ArchiMate & BPMN, the following should be performed as such that:

Let f be a function of composite concepts C_1 and C_i (enhance $\tau_{A,B}$)

Hypotheses 1

$$\boldsymbol{\pi_A(C) \equiv \pi_B\left(\tau_{A,B}(\mathcal{C}_A)\right)} \textbf{\textit{ would be }} \boldsymbol{\pi_A(\mathcal{BP}) \equiv \pi_B\left(f\left(\pi_B(\mathcal{T}), \pi_B(\mathcal{S})\right)\right)} \textbf{\textit{ and}}$$

$$\boldsymbol{\pi_A(\mathcal{BP}) \equiv \pi_B\left(\tau_{B,A}\left(f_{B,A}\left(\tau_{B,A}(\mathcal{T}), \tau_{B,A}(\mathcal{S})\right)\right)\right) = \mathcal{BP}}$$

Hypotheses 2

This applies for the opposite way *(enhance $\tau_{B,A}$)*

$$\pi_{\boldsymbol{B}}(C) \equiv \pi_{\boldsymbol{A}}\left(\tau_{B,A}(\mathcal{C}_{\boldsymbol{B}})\right) \textbf{\textit{ would be }} \pi_{\boldsymbol{B}}(\boldsymbol{C}) \equiv \pi_{\boldsymbol{A}}\left(\boldsymbol{f}\left(\boldsymbol{\pi_A}(C_1), \boldsymbol{\pi_A}(C_i)\right)\right)$$

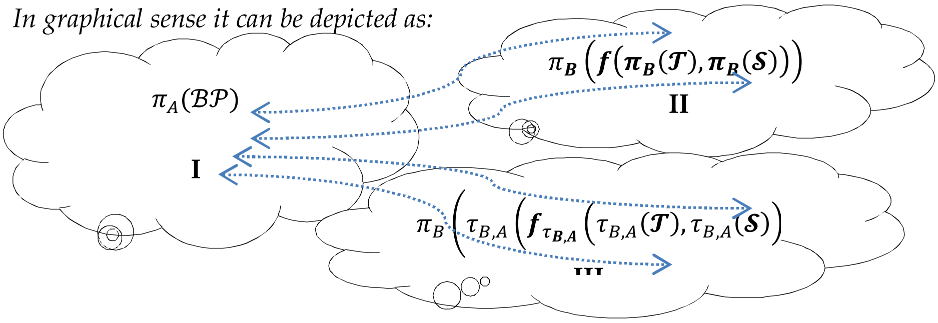
$$\boldsymbol{\pi_B}(C) \equiv \boldsymbol{\pi_A}\left(\boldsymbol{\tau}_{A,B}\left(\boldsymbol{f}_{\boldsymbol{\tau_{A,B}}}\left(\boldsymbol{\tau}_{A,B}(\mathcal{C}_1), \boldsymbol{\tau}_{A,B}(\mathcal{C}_i)\right)\right)\right) = C$$

The above method should be as such that **II(I)** reflects **I**, which means that constructs in **II(I)** should be at some way traced back in the constructs of **I**.

If composites concepts are found in $\pi_A(\mathcal{C}_A)$ or $\pi_B(\mathcal{C}_B)$ concepts

If the above mentioned cannot be enhanced, it can be concluded that from the perspective of ArchiMate/BPMN the concept seems not as elementary as in first sight.

In graphical sense it can be depicted as:



$\pi_A(\mathcal{BP})$
I
$\pi_B\left(f\left(\pi_B(\mathcal{T}), \pi_B(\mathcal{S})\right)\right)$
II
$\pi_B\left(\tau_{B,A}\left(f_{\tau_{B,A}}\left(\tau_{B,A}(\mathcal{T}), \tau_{B,A}(\mathcal{S})\right)\right)\right)$
III

The equation $\pi_A(\mathcal{C}_A) \equiv \pi_B(\tau_{A,B}(\mathcal{C}_A)) \vDash \pi_B\left(\tau_{A,B}(\mathcal{C}_A)\right) \leftrightarrow \pi_A(\mathcal{C}_A)$ holds for a **(ArchiMate) business activity / (BPMN) task** as well as the equation $\pi_B(\mathcal{C}_B) \equiv \pi_A(\tau_{B,A}(\mathcal{C}_B)) \vDash \pi_A\left(\tau_{B,A}(\mathcal{C}_B)\right) \leftrightarrow \pi_B(\mathcal{C}_B)$. For the business concept **business process** in ArchiMate, the direct mapping of a business concept (*hypothesis 1*) is semantically identical to the indirectly mapping $\pi_A(\mathcal{C}_A) \equiv \boldsymbol{\pi_B\left(\tau_{A,B}(\mathcal{C}_A)\right)}$. This lies in the composite concepts that can be found in **I**. If function $\boldsymbol{f}$ is mapped to ArchiMate, it results in a ***business process*** concept corresponding to the hypothesis. But when mapping back to ArchiMate, hypothesis 2 would result in elementary concept in ArchiMate, namely the ***business process or activity concept*** if considers $\boldsymbol{\pi_B(\mathcal{T})}$ and $\boldsymbol{\pi_B(\mathcal{S})}$ separately. We can conclude that the semantics of the ArchiMate business/BPMN language concepts are the same. This makes integration of models possible.

References

[Aals 11] Aalst, W.M.P. van der, Lohmann, N., La Rosa, M., Ensuring correctness during process configuration via partner synthesis, *Information Systems*, volume 37, Issue 6, pp. 574-592, September 2011.

[Aloi 12] Aloini, D., Dulmin, R., Mininno, V., Modelling and assessing ERP project risks: A Petri Net approach, *European journal of operational research*, volume 220, issue 2, pp. 484 -495, February 2012.

[Bern 02] Bernardo, M., Busi, N., Ribaudo, M., Integrating TwoTowers and GreatSPN through a compact net semantics, vol. 50, issue 2-3, pp. 153-187, 2002.

[Bill 03] Billington, J., Christensen, S., Hee, K. van, Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M., The Petri Net Markup Language: Concepts, Technology and Tools, *Lecture notes in computer science ISSN 0302-9743*, vol. 2679, pp. 483-506, 2003.

[Boer 05] De Boer, F.S., Bonsangue, M.M., Groenewegen, L.P.J., Stam, A.W., Stevens, S., van der Torre, L., *Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration (IRI -2005)*, Change Impact Analysis of Enterprise Architecture, pp. 177-181, 2005.

[Boer 06] de Boer, F.S., Bonsangue, M.M., Jacob, J.F., Stam, A., van der Torre, L., Using XML Transformation for Enterprise Architecture, *Lecture notes in computer science* [ISSN 0302-9743], vol. 4313, pp. 42 - 56, 2006.

[Buur 04] van Buuren, R., Jonkers, H., Iacob, M., Stratings, P. Composition of Relations in Enterprise Architecture Models, *Lecture Notes in Computer Science (LNCS)*, vol. 3256, pp. 39-53, 2004.

[Chin 12] Chinosi, M., Trombetta, A., BPMN: An introduction to the standard, *Computer standards & interfaces: an international journal*, vol. 34, pp. 124-134, 2012.

[Chri 10] Christiansen, D.R., Carbone, M., Hildebrandt, T., Formal Semantics and Implementation of BPMN 2.0 Inclusive Gateways, *Proceeding WS-FM'10: Proceedings of the 7th international conference on Web services and formal methods*, pp. 146-160, 2010.

[Dijk 08] Dijkman, R.M., Dumas, M., Ouyang, C., Formal semantics and analysis of business process models in BPMN, *Journal Information and Software Technology archive*, vol. 50 Issue 12, pp. 1281-1294, November, 2008.

[Gree 11] D. Greefhorst and H.A. Proper. A Practical Approach to the Formulation and Use of Architecture Principles. In *Proceedings of the 6th Trends in Enterprise Architecture Research (TEAR) workshop, held in conjunction with the 15th International EDOC Enterprise Computing Conference*, pages 330-339, Helsinki, Finland, September 2011.

[Gust 09] Gustafsson, P., Höök, D., Franke, U., Johnson P., Modeling the IT Impact on Organizational Structure, *IEEE International Enterprise Distributed Object Computing Conference (edoc 2009)*, pp. 14-23, 2009.

[Halp 96] Halpin, T., Business rules and Object-Role modeling, *Database Programming and Design*, vol. 9 no. 10, pp. 66-72, 1996.

[Halp 98] Halpin, T., Object-Role Modeling (ORM/NIAM), *Handbook on Architectures of Information Systems (Ch.4)*, Springer, Heidelberg, 1998.

[Hopp 04] Hoppenbrouwers, S.J.B.A., Bleek, A.I., Proper, H.A., *Modeling Linguistically Complex Business Domains*, Technical Report NIII-R0405, Nijmegen Institute for Information and Computing Sciences, University of Nijmegen, The Netherlands, EU, 2004.

[Hopp 05] Hoppenbrouwers S.J.B.A., Proper, H.A., van der Weide, Th.P. (eds.): Conceptual Modelling ER-2005, A Fundamental View on the Process of Conceptual Modeling, *Lecture Notes in Computer Science (LNCS)*, vol. 3716, pp. 128-143, 2005.

[Jonk 03] Jonkers, H, van Buuren, R., Arbab, F., de Boer. F., Bonsangue M., Bosma, H., ter Doest, H., Groenewegen, L., Scholten J.G., Hoppenbrouwers S., Iacob, M., Janssen W., Lankhorst, M., van Leeuwen, D., Proper, H.A., Stam, A., van der Torre, L., Veldhuijzen van Zanten, G., Towards a Language for Coherent Enterprise Architecture Descriptions, *Proceedings of the 7th International Conference on Enterprise Distributed Object Computing*, pp. 28-37, 2003.

[John 07] Johnson, P., Lagerström, R., Närman, P., Simonsson, M., Enterprise architecture analysis with extended influence diagrams, *Journal of Information Systems Frontier*, vol. 09, pp. 163-180, 2007.

[Jonk 11] H. Jonkers, M.M. Lankhorst, D.A.C. Quartel, H.A. Proper, and M.-E. Iacob. ArchiMate for Integrated Modelling Throughout the Architecture Development and Implementation Cycle. In *Proceedings of the 13th IEEE Conference on Commerce and Enterprise Computing (CEC2011)*, pages 294-301, Luxembourg-Kirchberg, Luxembourgu, 2011.

[Jonk 04] Jonkers, H., Lankhorst, M., van Buuren, R., Hoppenbrouwers S., Bonsangue M., van der Torre, L., Concepts for Modelling Enterprise Architectures, *International Journal of Cooperative Information Systems*, 2004.

[Kind 06] Kindler, E., (2006). Concepts, Status, and Future Directions. In E. Schnieder (ed.): Entwurf Komplexer Automatisierungssysteme, EKA 2006, 9. pp. 35-55, May 2006.

[Khom 07] Khomenko, V., Koutny, M., Verification of bounded Petri nets using integer programming, Formal methods in system design, vol. 30, Issue 2, pp. 143 - 176, 2007.

[Kosa 07] Kosanke, K., Vernadat, F., Zelm, M., CIMOSA: enterprise engineering and integration, *Computers In Industry*, vol. 40, pp. 83-97, 1999.

[Land 09] Op 't Land, M., Proper, H.A., Waage, M., Cloo, J., Steghuis, C., *Enterprise Architecture: Creating Value by Informed Governance*, Springer, Berlin, 2009.

[Lank 05] Lankhorst, M.M., van Buuren, R., van Leeuwen, D., Jonkers, H., ter Doest, H., Enterprise architecture modelling—the issue of integration, *Advanced Engineering Informatics: Enterprise Modelling and System Support*, vol. 18 (Issue 04), pp. 205-216, January 2005.

[Lank 09a] Lankhorst, M.M., Proper, H.A., Jonkers, H., The Architecture of the ArchiMate Language, *Lecture Notes in Business Information Processing*, vol. 29, pp. 367-380, June 2009.

[Lank 09b] Lankhorst M. et al., *Enterprise Architecture at Work: Modelling: Communication and Analysis*, Springer, Berlin, 2009.

[Lind 11] D.J.T. Van der Linden, S.J.B.A. Hoppenbrouwers, A. Lartseva, and H.A. Proper. Towards an Investigation of the Conceptual Landscape of Enterprise Architecture. In T. Halpin, editor, *BPMDS 2011 and EMMSAD 2011 proceedings, pp. 526-535. Springer, Heidelberg*, number 81 in LNBIP, pages 526-535. Springer, Berlin, Germany, 2011.

[Odeh 03] Odeh, M., Kamm, R., Bridging the gap between business models and system models, *Information and Software Technology*, volume 45, Issue 15, pp. 1053-1060, December 2003.

[Over 07] Overbeek, S.J., van Bommel, P., Proper, H.A. (Erik)., Rijsenbrij, D.B.B., Visualizing Formalisms with ORM Models, *In Proceedings of the 2007 OTM confederated international conference on the move to meaningful internet systems LNCS 4805*, vol. (Part I), pp. 709-718, 2007.

[Paig 00] Paigea, R.F., Ostroffa, J.S., Brooke, P.J., Principles for modeling language design, *Information and Software Technology*, volume 42, Issue 10, pp. 665–675, March 2000.

[Prop 05] Proper, H.A., Verrijn-Stuart, A., Hoppenbrouwers, S., Towards Utility-based Selection of Architecture-Modelling Concepts, *Proceedings of the Second Asia-Pacific Conference on Conceptual Modelling APCCM*, vol. 42, pp. 25-36, 2005.

[Raed 07] Raedts, I., Petković, M., Usenko, Y.S., van der Werf, J.M., Groote, J.F., Somers, L., Transformation of BPMN models for Behaviour Analysis, *In Proceeding of: Modeling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS 2007)*, pp. 126-137, June 2007.

[Shen 04] Shen, H., Wall, B., Zaremba, M., Chen, Y., Browne, J., Integration of business modelling methods for enterprise information system analysis and user requirements gathering, *Computers in Industry*, volume 54, Issue 3, pp. 307-323, August 2004.

[Soar 08] Dos Santos Soares, M., Jos Vrancken, A Meta-modeling Approach to transform UML 2.0 Sequence Diagrams to Petri Nets, *Proceeding SE '08 Proceedings of the IASTED International Conference on Software Engineering*, pp. 159-164, 2008.

[Tuli 09] P. Tulinayo, S.J.B.A Hoppenbrouwers, P. Bommel Van Bommel, and H.A. Proper. Integrating System Dynamics with Object-Role Modeling and Petri Nets. In J. Mendling, S. Rinderle-Ma, and W. Esswein, editors, *Enterprise Modelling and information systems Architectures*, pages 41-54, Ulm, Germany, September 10-11 2009.

[Wang 05] Wang, C-B., Chen, T-Y., Chen, Y-M., H.C., Chu, Design of a Meta Model for integrating enterprise systems, *Computers in Industry*, vol. 56, Issue 3, pp. 305-322, 2005.

[Webe 03] Weber, M., Kindler, E., The Petri Net Markup Language, *In Proceeding of: Lecture Notes in Computer Science ISSN 0302-9743*, vol. 2472, pp. 124-144, 2003.

[Whit 04a] White, S.A., Process Modeling Notations and Workflow Patterns, IBM Corp., United States, January 2004.

[Whit 04b] White, S.A., Introduction to BPMN, www.bptrends.com, IBM Corporation, July 2004.

[Whit 05a] White, S.A., Using BPMN to model a BPEL process, www.bptrends.com, IBM Corporation, July 2005.

[Whit 05b] White, S.A., mapping BPMN to BPEL example, www.bptrends.com, IBM Corporation, February 2005.

[Wier 04] Wiering M.J., Bonsangue M.M., van Buuren R., Groenewegen L.P.J., Jonkers H., Lankhorst M.M., Investigating the mapping of an Enterprise Description Language into UML 2.0, *Electronic Notes in Theoretical Computer Science 101 (ENTCS)*, pp. 155–179, 2004.

[Yu 07] Yu, X., Zhang, Y., Zhang, T., Wang, L., Hu, J., Zhao, J., Li, Xuandong, A model-driven development framework for enterprise Web services, *Information Systems Frontier*, vol. 09, pp. 391-409, July 2007.

[Zang 07] Zang, C., Fan, Y., Liu, R., Architecture, implementation and application of complex event processing in enterprise information systems based on RFID, *Information Systems Frontiers*, vol. 10, Issue 5, pp. 543 - 553, 2008.

Reports / Documentation

[01] Documentation, International Standard ISO/IEC 15909 version 4.7.1, High-level Petri Nets Concepts, Definitions and Graphical Notation Final Draft, October 28, 2000.

[02] Business Process Model Notation (BPMN) Specification 2.0, http://www.omg.org/spec/BPMN/2.0, January 2011.

[03] Mapping ArchiMate and standards ArchiMate Deliverable ArchiMate Deliverable 2.2.3b, 2004.

[04] Concepts for Architectural Descriptions ArchiMate Deliverable 2.2.1 v4.1, 2007.

Glossary

A

Activity - an activity is a generic term for work that company performs in a process. An activity can be atomic or non-atomic (compound). The types of activities that are a part of a process model are: sub-process and task, which are rounded rectangles. Activities are used in standard processes.
Application collaboration - a configuration of two or more application components that cooperate to jointly perform application interactions.
Application component - a modular, deployable, and replaceable part of a system that encapsulates its contents and exposes its functionality through a set of interfaces.
Application function - a coherent unit of internal behaviour of an application component.
Application interaction - a unit of behaviour performed by a collaboration of two or more application components.
Application interface - declares how a component can connect with its environment.
Application service - an externally visible unit of functionality, provided by one or more components, exposed through well-defined interfaces, and meaningful to the environment.
Arc annotation - an expression that may involve constants, variables and operators used to annotate an arc of a net. The expression must evaluate to a multiset over the type of the arc's associated place.
Artifact - a physical piece of information that is used or produced in a software development process, or by deployment and operation of a system.
Association - an association is used to link information and artifacts with BPMN graphical elements. Text annotations and other artifacts can be associated with the graphical elements. An arrowhead on the association indicates a direction of flow (e.g., data), when appropriate.

B

Business actor - an organizational entity capable of (actively) performing behaviour.
Business collaboration - a (possibly temporary) configuration of two or more business roles resulting in specific collective behavior (interactions) in a particular context).
Business event - something that happens (internally or externally) and may influence business behaviour (business processes, functions, interactions).

Business function - a unit of internal behaviour that groups behaviour according to (for example) required skills, knowledge, resources, etc.

Business interaction - a unit of behaviour performed in collaboration by two or more business roles.

Business interface - declares how a business role can connect with its environment.

Business object - a unit of information relevant from a business perspective.

Business process - a unit of internal behaviour or collection of causally-related units of internal behaviour intended to produce a defined set of products and services.

Business role - a named specific behavior of a business actor participating in a particular context.

Business service - the externally visible ('logical') functionality, which is meaningful to the environment and is realized by business behaviour (business process, business function or business interaction).

C

Communication path - a logical link between two or more nodes, through which these nodes can exchange information.

Contract - a formal or informal specification of agreement that specifies the rights and obligations associated with a product.

D

Data Object (ArchiMate) - a coherent, self-contained piece of information suitable for automated processing.

Data Object (BPMN) - provide information about what activities require to be performed and/or what they produce. Data Objects can represent a singular object or a collection of objects.

Declaration - a set of statements which define the sets, constants, parameter values, typed variables and functions required for defining the inscriptions on a High-level Petri Net Graph.

Device - a physical computational resource upon which artifacts may be deployed for execution.

E

Enabling (a transition) - a transition is enabled in a particular mode and net marking, when the following conditions are met: The marking of each input place of the transition satisfies the demand imposed on it by its arc annotation evaluated for the particular transition mode. The demand is

satisfied when the place's marking contains (at least) the multiset of tokens indicated by the evaluated arc annotation.
Enabling Tokens - the multiset of values obtained when an input arc annotation is evaluated for a particular binding to variables.
End Event - indicates where a process will end.
Enterprise Architecture - a coherent whole of principles, methods, and models that are used in the design and realization of an enterprise's organizational structure, business processes, information system and infrastructure.
Event - something that "happens" during the course of a process. These events affect the flow of the model and usually have a cause (*trigger*) or an impact (*result*). Events are circles with open centers to allow internal markers to differentiate different *triggers* or *results*. There are three types of events, based on when they affect the flow: **start event**, **intermediate event**, and **end event**.

G

Gateway - used to control the divergence and convergence of sequence flows in a process. Thus, it will determine branching, forking, merging, and joining of paths. Internal markers will indicate the type of behavior control.

H

High-level Petri Net Graph - a net graph and its associated annotations comprising Place Types, Arc Annotations, Transition Conditions, and their corresponding definitions in a set of Declarations, and an Initial Marking of the net.

I

Infrastructure interface - a point of access where the infrastructural services offered by a node can be accessed by other nodes or by application components.
Infrastructure service - externally visible unit of functionality, provided by one or more nodes, exposed through well-defined interfaces, and meaningful to the environment.
Initial Marking (of the net) - the set of initial place markings
Initial Marking of a place - a special marking of a place
Input Arc (of a transition) - an arc directed from a place to the transition.
Input Place (of a transition) - a place connected to the transition by an input arc.

Intermediate Event - occurs between a start event and an end event. They will affect the flow of the process, but will not start or (directly) terminate the process.

L

Lane - a lane is a sub-partition within a process, sometimes within a pool, and will extend the entire length of the process, either vertically or horizontally. Lanes are used to organize and categorize activities.

M

Marking (of a net) - the set of the place markings for all places of the net.
Marking of a place - a multiset of tokens associated with ('residing in') the place.
Meaning - the knowledge or expertise present in (the representation of) a business object, given a particular context. A message flow is used to show the flow of messages between two participants that are prepared to send and receive them.
Message - a message is used to depict the contents of a communication between two participants
Mode - a value taken from the transition's type. When considering a High-level Petri Net Graph, a mode may be derived from an assignment of values to the transition's variables that satisfies the transition condition.

N

Net - a general term used to describe all classes of Petri nets.
Network - a physical communication medium between two or more devices.
Node - a logical computational resource upon which artifacts may be deployed for execution.
Node (of a net) - a vertex of a net graph (i.e., a place or a transition).

O

Output Arc (of a transition) - an arc directed from the transition to a place.
Output Place (of a transition) - a place connected to the transition by an output arc.

P

Place - a node of a net, taken from the place kind, normally represented by an ellipse in the net graph. A place is typed.

Place Type - a non-empty set of data items associated with a place.
Pool - the graphical representation of a *Participant*. It also acts as a “swimlane” and a graphical container for partitioning a set of Activities from other Pools, usually in the context of B2B situations. A pool *may* have internal details, in the form of the Process that will be executed. Or a pool *may* have no internal details, i.e., it can be a "black box."
Product - a coherent collection of services accompanied by a *contract*/set of agreements, which is offered as a whole to (internal or external) customers.

R

Representation - the perceptible form of the information carried by a business object.

S

Service - a service is defined as a unit of functionality that some entity (e.g. system, organization or department) makes available to its environment, and which has some value for certain entities in the environment (typically the 'service users').
Sequence Flow - a sequence flow is used to show the order that Activities will be performed in a Process.
Start Event - indicates where a particular process will start.
Sub-Process - is a compound activity that is included within a. It is compound in that it can be broken down into a finer level of detail (a process) through a set of sub-activities.
System software - a software environment for specific types of application components and data objects that are deployed on it in the form of artifacts.

T

Task - a task is an atomic activity that is included within a process. A Task is used when the work in the process is not broken down to a finer level of process detail.
Token - a data item associated with a place and chosen from the place’s type
Transition - a node of a net, taken from the transition kind, and represented by a rectangle in the net graph.
Transition condition - a Boolean expression (one that evaluates to true or false) associated with a transition.
Transition mode - a pair comprising the transition and a mode.
Transition occurrence (Transition rule) - if a transition is enabled in a mode, it may occur in that mode. On the occurrence of the transition, the following actions occur indivisibly:

- For each input place of the transition: the enabling tokens of the input arc with respect to that mode are subtracted from the input place's marking, and
- For each output place of the transition: the multiset of tokens of the evaluated output arc expression is added to the marking of the output place.

NOTE: A place may be both an input place and an output place of the same transition.

Transition Variables - all the variables that occur in the expressions associated with the transition. These are the transition condition, and the annotations of arcs surrounding the transition.
Type - a set.

V

Value - that which makes some party appreciate a product or service.

Appendix A. ArchiMate Metamodel

A summary of the ArchiMate concepts and their relationships is shown in **Fig. A.1** guided by the determined permitted relationships in Appendix C.

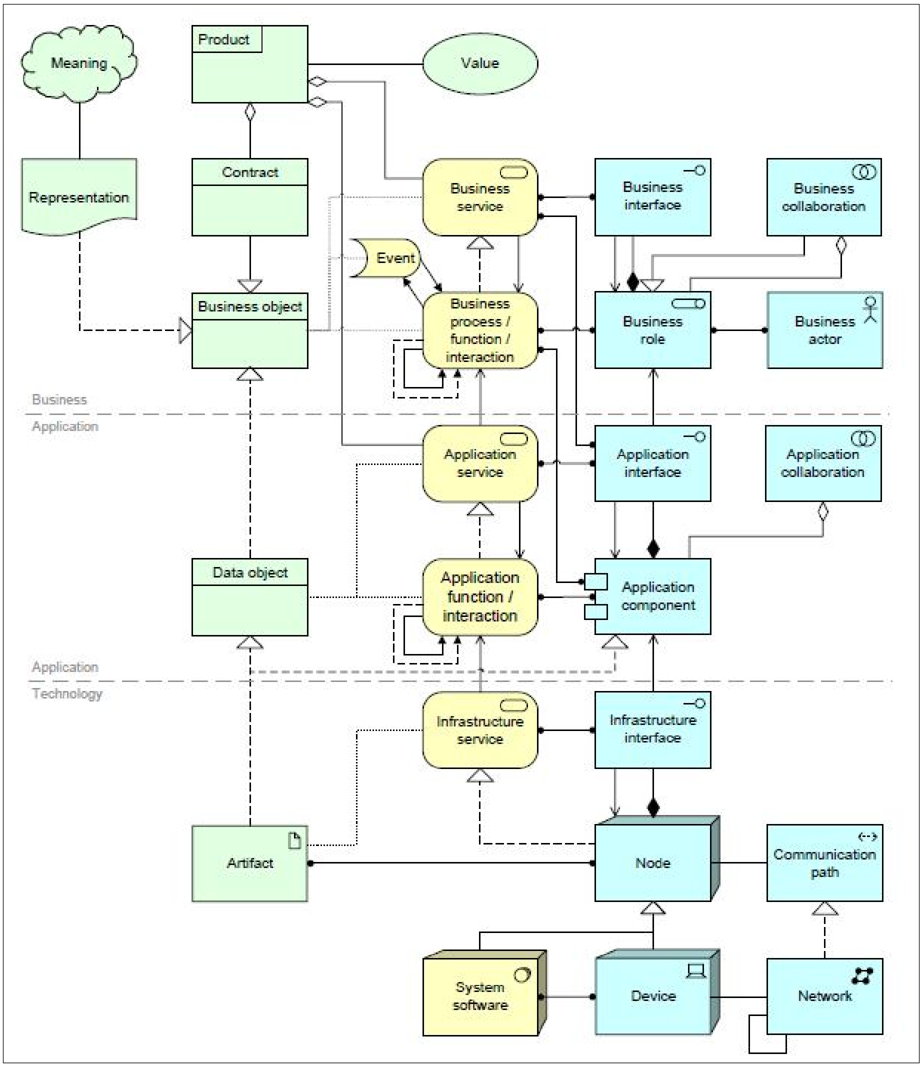


Fig. A.1. Metamodel of the ArchiMate language.

Appendix B. ArchiMate Graphical Notation

The symbols of the ArchiMate language are shown in **Fig. B.1.**

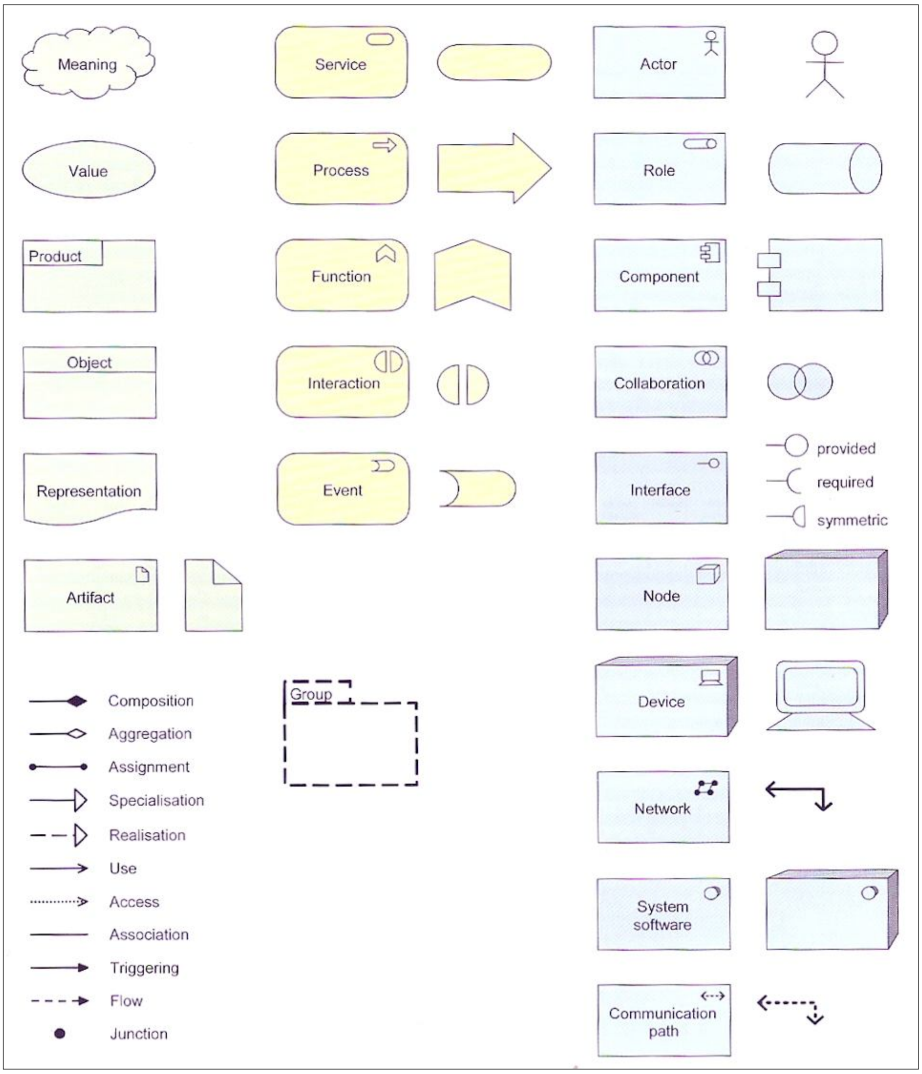


Fig. B.1. *Symbols of the ArchiMate language*

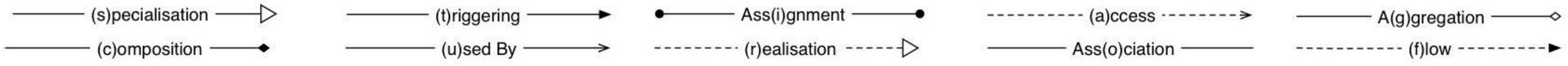
Appendix C. ArchiMate Relations

The table below lists all permitted relationships between elements of the ArchiMate language, which is based on the ArchiMate metamodel in Appendix A.

Table C.1. Permitted relations between ArchiMate concepts.

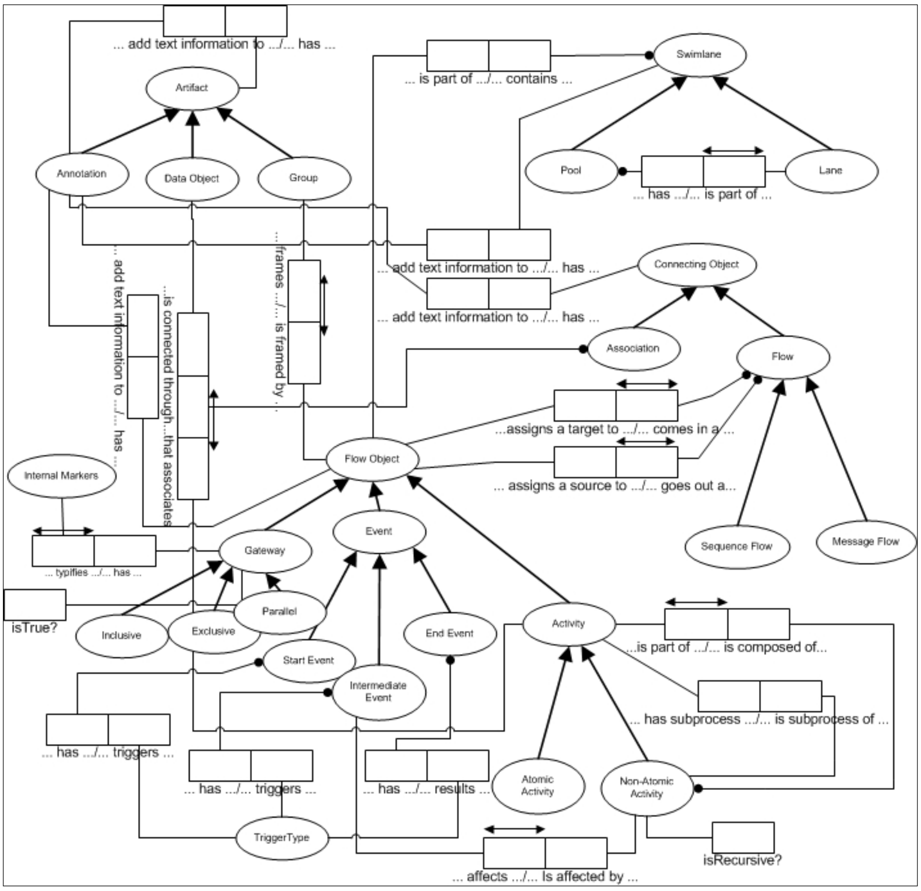
	Junction	Business Activity	Business Event	Busines Interaction	Business Process	Business Actor	Business Interface	Business Colabo-ration	Business Role	Business Function	Contract	Product	Business Service	Value	Business Object	Repres-entation	Meaning	Application Collaboration	Application Component	Application Function	Application Interaction	Application Interface	Application Service	Data Object	Artifact	Communitation Path	Device	Node	Infra-structure Interface	Network	Infra-structure Service	System Software
Junction	ft	ft	ft	ft	ft	ft	ft	ft	ft	ft			ft					ft	ft	ft	ft	ft	ft				ft	ft	ft		ft	ft
Business Activity	ft	fostu	fotu	fotu	fotu	ou	ou	ou	ou	fotu	ao	oru	oru	o	ao	ao	o	ou	ou	ou	ou	ou	oru	ao	o	o	o	o	o	o	o	o
Business Event	ft	fot	cfgost	fot	fot	o	o	o	o	fot	ao	o	o	o	ao	ao	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Busines Interaction	ft	fotu	fotu	cfgostu	fotu	ou	ou	ou	ou	fotu	ao	oru	oru	o	ao	ao	o	ou	ou	ou	ou	ou	oru	ao	o	o	o	o	o	o	o	o
Business Process	ft	cfgotu	cfgotu	fotu	cfgostu	ou	ou	ou	ou	cfgotu	ao	oru	oru	o	ao	ao	o	ou	ou	ou	ou	ou	oru	ao	o	o	o	o	o	o	o	o
Business Actor	ft	iou	iou	ou	iou	cfgostu	fiotu	cfgiostu	cfgiostu	fiou	ao	oru	ioru	o	ao	ao	o	fotu	fotu	ou	ou	fotu	oru	ao	o	o	o	o	o	o	o	o
Business Interface	ft	ou	ou	ou	ou	fotu	cfgostu	fotu	fotu	ou	ao	ou	iou	o	ao	ao	o	fotu	fotu	ou	ou	fotu	ou	ao	o	o	o	o	o	o	o	o
Business Colaboration	ft	iou	iou	iou	iou	cfgostu	cfgiotu	cfgiostu	cfgiostu	fiou	ao	oru	ioru	o	ao	ao	o	fotu	fotu	ou	ou	fotu	oru	ao	o	o	o	o	o	o	o	o
Business Role	ft	iou	iou	ou	iou	cfgostu	cfgiotu	cfgiostu	cfgiostu	fiou	ao	oru	ioru	o	ao	ao	o	fotu	fotu	ou	ou	fotu	oru	ao	o	o	o	o	o	o	o	o
Business Function	ft	cfgotu	cfgotu	fotu	cfgotu	fou	ou	fou	fou	cfgostu	ao	oru	oru	o	ao	ao	o	ou	ou	ou	ou	ou	oru	ao	o	o	o	o	o	o	o	o
Contract		o	o	o	o	o	o	o	o	o	cgos	o	o	o	cgos	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Product		ou	ou	ou	ou	ou	ou	ou	ou	ou	ago	cgosu	gou	o	ao	ao	o	ou	ou	ou	ou	ou	gou	ao	o	o	o	o	o	o	o	o
Business Service	ft	ou	ou	ou	ou	ou	ou	ou	ou	ou	ao	ou	cfgostu	o	ao	ao	o	ou	ou	ou	ou	ou	fotu	ao	o	o	o	o	o	o	o	o
Value	o	o	o	o	o	o	o	o	o	o	o	o	c	gos	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Business Object		o	o	o	o	o	o	o	o	o	cgos	o	o	o	cgos	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Representation		o	o	o	o	o	o	o	o	o	or	o	o	o	or	cgos	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Meaning		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	cgos	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Application Collaboration	ft	iou	iou	iou	iou	fotu	fotu	fotu	fotu	iou	ao	oru	ioru	o	ao	ao	o	cfgostu	cfgostu	iou	iou	cfgotu	ioru	ao	o	o	o	o	o	o	o	o
Application Component	ft	iou	iou	ou	iou	fotu	fotu	fotu	fotu	iou	ao	oru	ioru	o	ao	ao	o	cfgostu	cfgostu	iou	ou	cfgotu	ioru	ao	o	o	o	o	o	o	o	o
Application Function	ft	ou	ou	ou	ou	ou	ou	ou	ou	ou	ao	ou	ou	o	ao	ao	o	ou	ou	cfgostu	fotu	ou	oru	ao	o	o	o	o	o	o	o	o
Application Interaction	ft	ou	ou	ou	ou	ou	ou	ou	ou	ou	ao	ou	ou	o	ao	ao	o	ou	ou	fotu	cfgostu	ou	oru	ao	o	o	o	o	o	o	o	o
Application Interface	ft	ou	ou	ou	ou	fotu	fotu	fotu	fotu	ou	ao	ou	iou	o	ao	ao	o	fotu	fotu	ou	ou	cfgostu	iou	ao	o	o	o	o	o	o	o	o
Application Service	ft	ou	ou	ou	ou	ou	ou	ou	ou	ou	ao	ou	fotu	o	ao	ao	o	ou	ou	ou	ou	ou	cfgostu	ao	o	o	o	o	o	o	o	o
Data Object		o	o	o	o	o	o	o	o	o	or	o	o	o	or	o	o	o	o	o	o	o	o	cgos	o	o	o	o	o	o	o	o
Artifact		o	o	o	o	o	o	o	o	o	aor	or	o	o	aor	ao	o	oru	oru	oru	oru	oru	oru	aor	cgos	o	o	o	o	o	o	o
Communitation Path		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	cgos	o	o	o	o	o	o
Device	ft	ou	ou	ou	ou	ou	ou	ou	ou	ou	ao	ou	ou	o	ao	ao	o	ou	ou	ou	ou	ou	ou	ao	aiou	ioru	cfgostu	cfgostu	cfgotu	iou	ioru	cfgiostu
Node	ft	ou	ou	ou	ou	ou	ou	ou	ou	ou	ao	ou	ou	o	ao	ao	o	ou	ou	ou	ou	ou	ou	ao	aiou	ioru	cfgostu	cfgostu	cfgotu	iou	ioru	cfgiostu
Infrastructure Interface	ft	ou	ou	ou	ou	ou	ou	ou	ou	ou	ao	ou	ou	o	ao	ao	o	ou	ou	ou	ou	ou	ou	ao	aou	ou	fotu	fotu	cfgostu	ou	iou	fotu
Network		o	o	o	o	o	o	o	o	o	o	o	o	o	o	ao	o	o	o	o	o	o	o	o	o	o	o	o	o	cgos	o	o
Infrastructure Service	ft	ou	ou	ou	ou	ou	ou	ou	ou	ou	ao	ou	ou	o	ao	ao	o	ou	ou	ou	ou	ou	ou	ao	aou	ou	ou	ou	ou	ou	cfgostu	ou
System Software	ft	ou	ou	ou	ou	ou	ou	ou	ou	ou	ao	ou	ou	o	ao	ao	o	ou	ou	ou	ou	ou	ou	ao	aiou	ioru	cfgostu	cfgostu	cfgotu	iou	ioru	cfgiostu

EA
@
Work



Appendix D. BPMN Metamodel

A summary of the BPMN concepts and their relationships is shown in **Fig. D.1** guided by the descriptions in Chapter 3 (see **Sect. 3.2**).



... add text information to .../... has ...
Artifact
Annotation
Data Object
Group
... is part of .../... contains ...
Swimlane
Pool
Lane
... has .../... is part of ...
... add text information to .../... has ...
... add text information to .../... has ...
Connecting Object
Association
Flow
... add text information to .../... has ...
...is connected through...that associates
... frames .../... is framed by ...
...assigns a target to .../... comes in a ...
... assigns a source to .../... goes out a...
Flow Object
Internal Markers
... typifies .../... has ...
Gateway
Event
Sequence Flow
Message Flow
isTrue?
Inclusive
Exclusive
Parallel
Start Event
Intermediate Event
End Event
Activity
...is part of .../... is composed of...
... has subprocess .../... is subprocess of ...
... has .../... triggers ...
... has .../... triggers ...
... has .../... results ...
Atomic Activity
Non-Atomic Activity
TriggerType
... affects .../... Is affected by ...
isRecursive?

D.1. Metamodel of the BPMN language.

Appendix E. BPMN Graphical Notation

The elements of the basic BPMN modeling language are shown in **Fig. E.1.**

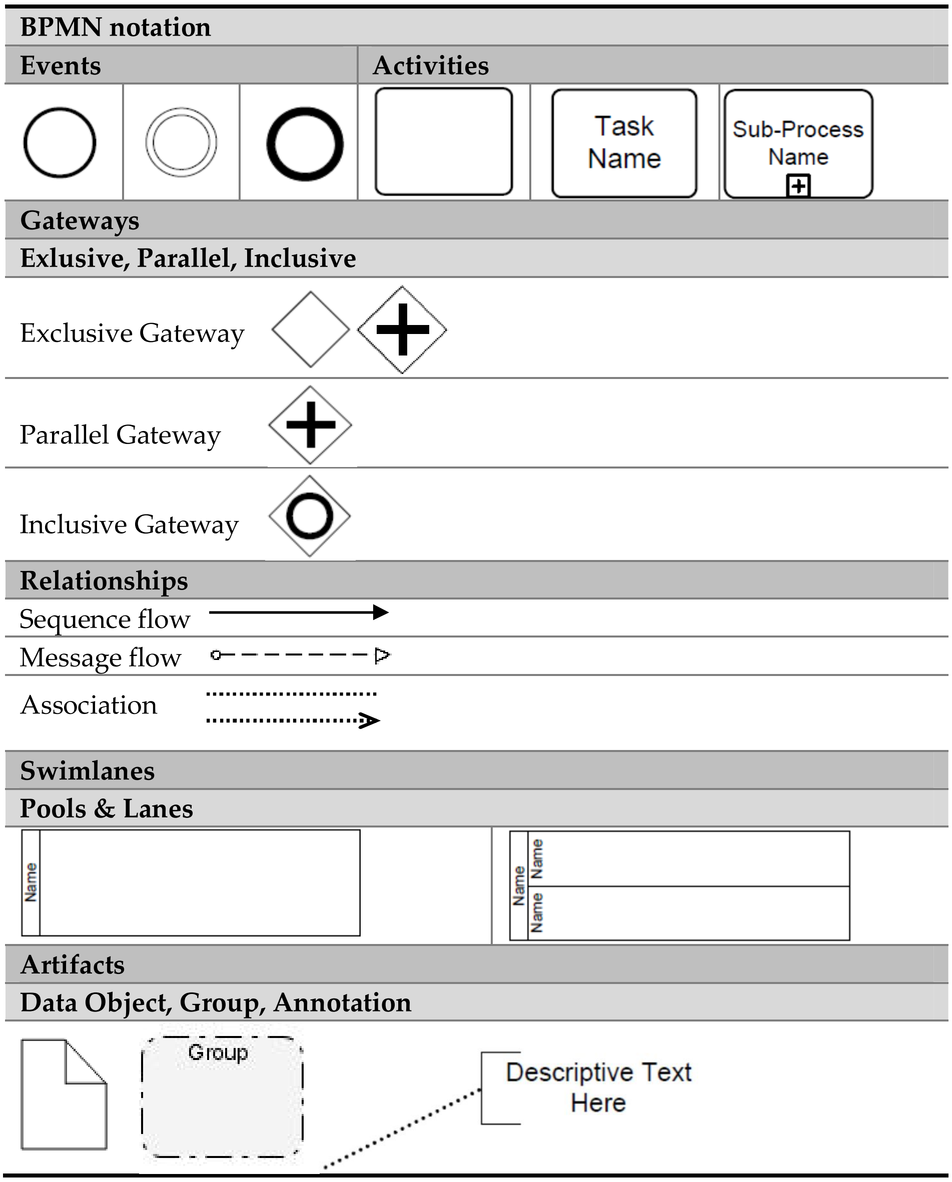


Fig. E.1. Symbols of the BPMN language.

Appendix F. Petri Net Metamodel

A summary of the Petri Net structure and their relationships is shown in **Fig. F.1** guided by the descriptions in Chapter 3 (see **Sect. 3.3**).

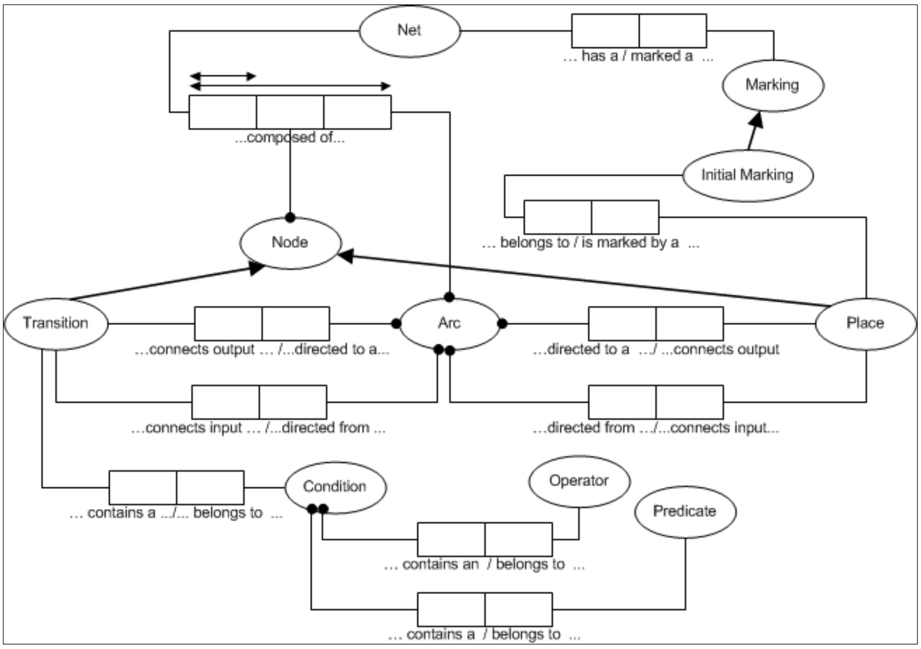

Net
… has a / marked a …
Marking
…composed of…
Initial Marking
… belongs to / is marked by a …
Node
Transition
Arc
Place
…connects output … /…directed to a…
…directed to a …/ …connects output
…connects input … /…directed from …
…directed from …/…connects input…
Condition
Operator
Predicate
… contains a …/… belongs to …
… contains an / belongs to …
… contains a / belongs to …

Fig. F.1 Petri Net metamodel.

Constraints that needs to be added (see **Fig. F.1**):

1. Node **1…***
2. Arc **1…***
3. Predicate **1** (to Condition)
4. Operator **1** (to Condition)
5. Place **1…*** (Arc)
6. Transition **1…*** (Arc)

Appendix G. Petri Nets Graphical Notation

Petri Net graphical notations are summarized in Fig. G.1.

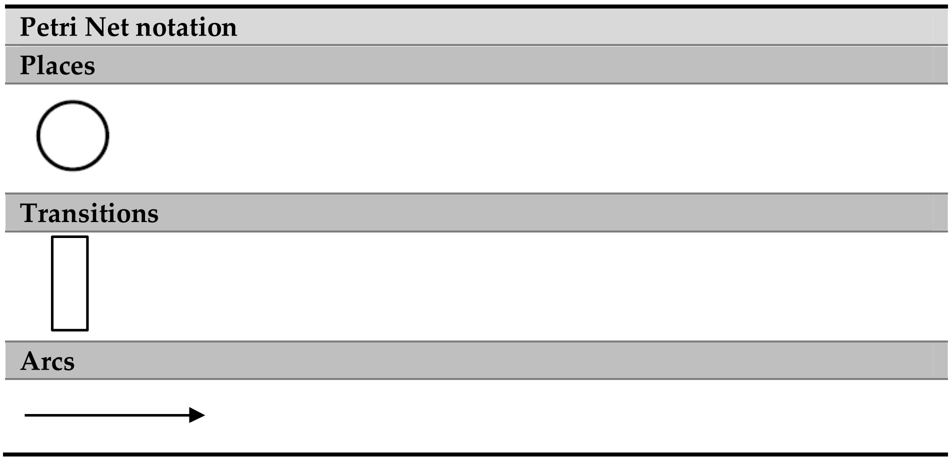


Fig. G.1 Petri Net graphical symbols.