



# T100

## Modelling SABSA® with ArchiMate®

Release 1.0

A SABSA Tools & Techniques Paper published by The SABSA Press™, an imprint of The SABSA Institute™

October 2019

Copyright © 2019, The SABSA Institute C.I.C. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the copyright owners unless it is presented in its current form as published by The SABSA Institute.

Document Title: Modelling SABSA® with ArchiMate®. (A SABSA Tools & Techniques Paper) 2019

Document Number: TSI T100

Published by The SABSA Press, (a trading name of The SABSA Institute C.I.C.) October 2019.

Comments relating to the material contained in this document may be submitted to:

The SABSA Institute C.I.C, 126 Stapley Road, Hove, BN3 7FG, UK

Registered in England and Wales, No. 08439587

Or by electronic mail to:

[sabsapress@sabsa.org](mailto:sabsapress@sabsa.org)

## Trademarks

SABSA® is a registered trademark of The SABSA Institute. Other trademarks owned by The SABSA Institute are labelled with a TM mark on their first occurrence in the text.

ArchiMate®, TOGAF® ADM® & 'Boundaryless Information Flow®' are registered trademarks of The Open Group.

All other brands, company, and product names are used for identification purposes only and may be trademarks that are the sole property of their respective owners.

## This Document

This SABSA Tools & Techniques Paper describes how security architecture concepts can be expressed using ArchiMate 3.0, the latest version of The Open Group's widely adopted Enterprise Architecture (EA) modelling language.

It has been developed and approved by The SABSA Institute C.I.C. Board of Trustees.

## Acknowledgements

Author: Steven Bradley (SCP).

Contributors and Reviewers: John Czaplewski, Editor-in-Chief, The SABSA Institute. John Sherwood, Chief Architect; Maurice Smit, Deputy Chief Architect; The SABSA Institute

## Contents

1	Introduction.....	4
1.1	Background.....	4
1.2	What is ArchiMate?.....	6
1.3	Purpose.....	7
1.4	References.....	8
2	The Rationale for SABSA-ArchiMate Alignment .....	9
2.1	The Benefits of Modelling .....	9
2.2	Why ArchiMate needs a security architecture perspective.....	10
2.3	What Benefits can Modelling bring to SABSA.....	11
3	An Introduction to the ArchiMate Language .....	13
3.1	Core Elements .....	13
3.2	Core Relationships .....	13
3.2.1	Definition and Notation of Relationships.....	14
3.2.2	Derived Relationships.....	14
3.2.3	The Core Layers and their Elements.....	15
3.3	Extension Layers and Elements .....	18
3.3.1	Strategy Layer Extension.....	18
3.3.2	The Motivation Extension .....	18
3.3.3	Implementation and Migration Extension .....	19
3.3.4	Miscellaneous Elements .....	19
3.4	The ArchiMate 3.0 Framework .....	20
4	Aligning the SABSA and ArchiMate Frameworks .....	22
4.1	An Overview of the Task.....	22
4.2	Risk & Security Modelling in ArchiMate 3.....	23
5	A Security Overlay for ArchiMate 3 .....	27
5.1	The Security Overlay of the Motivation Metamodel.....	27
5.1.1	Value.....	27
5.1.2	Value Chains & Value Streams .....	28
5.1.3	SABSA Business Attributes .....	31

5.1.4	Meaning .....	36
5.1.5	Impact, Threat, Vulnerability & Risk Analysis .....	37
5.1.6	Control Objective and Control (Requirements) .....	38
5.1.7	Modelling Control Profiles .....	38
5.1.8	Trust .....	40
5.2	Modelling the SABSA® Contextual Security Architecture .....	40
5.2.1	The Business Goals Model .....	41
5.2.2	Security Extension of Business Elements .....	42
5.3	Modelling the SABSA® Conceptual Security Architecture .....	47
5.3.1	Domain Framework Model .....	48
5.3.2	Risk Management Model .....	49
5.3.3	Security & Risk Governance Model .....	52
5.4	Modelling the SABSA® Logical Security Architecture .....	58
5.4.1	Application Service .....	58
5.4.2	Account .....	60
5.4.3	Application Role .....	60
5.4.4	Application Interface .....	60
5.4.5	Application Component .....	61
5.4.6	Application Function .....	61
5.4.7	Software Defect .....	61
5.4.8	Security Event .....	61
5.4.9	Data Object .....	62
5.4.10	Security Configuration .....	62
5.5	Modelling the SABSA® Physical Security Architecture .....	63
5.5.1	Technology Service .....	63
5.5.2	Technology Interface .....	64
5.5.3	Device and Node .....	64
5.5.4	System Software .....	65
5.5.5	Artefact .....	65

5.5.6 Defect.....66

6 Conclusion.....68

# 1 Introduction

This White Paper describes how security architecture concepts can be expressed using ArchiMate 3.0, the latest version of The Open Group's widely adopted Enterprise Architecture (EA) modelling language. This enables an integrated approach to producing SABSA artefacts created with standard EA notations and tooling.

A roadmap to enterprise architecture with integrated security, through the alignment of SABSA concepts with Zachman, TOGAF ADM and other established EA frameworks, has been formally established for several years. In practice however, the lack of native support for security concepts in everyday EA modelling notations and tools (and therefore processes) means that security architects remain at a disadvantage to their architectural peers.

Devising a means to express security concepts in ArchiMate would fill a deficiency that is impeding the realisation of a truly holistic architecture methodology that can serve the needs of all architecture practitioners with a concern for security, which should be of course all architects, not just security architects.

## Next Steps

This White Paper seeks to advance the cause for the inclusion of security concepts into EA modelling in ArchiMate. It is published as a pragmatic and immediately useful guide for security modellers. Nevertheless, it describes only the first steps, early experiences and current state of thinking in an immature field.

By stimulating review comments and feedback from subject matter experts in the SABSA and EA modelling communities, the ideas discussed in this paper may be developed into a settled consensus of what the core security elements, relationships, properties and viewpoints should be. Eventually, this work could evolve into 'ArchiMateSE': a security extension of some future version of the ArchiMate standard. To these ends, a Working Group is being established on The SABSA Institute Portal to facilitate participation and to which contributions are welcome.

## 1.1 Background

In the not too distant past, information security was a collection of disparate disciplines (people, processes and technologies) that were detached from the main body of enterprise architecture. Yet it is self-evident that security architecture does not exist in isolation. Its purpose is to protect information and capabilities that exist in the Enterprise Architecture and it produces artefacts that guide the development of Enterprise Architecture.

Thankfully this demarcation is no longer the norm, having been challenged, first conceptually by the philosophy developed in the 'Blue Book'<sup>1</sup> and the subsequent integration and alignment of the SABSA

---

<sup>1</sup> Ref.1: Enterprise Security Architecture: A Business-Driven Approach. John Sherwood, Andy Clark, and David Lynas. Boca Raton, FL: Taylor & Francis Group, LLC. 2005.

approach with other EA frameworks<sup>2</sup> including TOGAF, ITIL, Zachman and DoDAF. SABSA was formally aligned and integrated with TOGAF via the 2011 Joint White Paper [Ref.3] and is now included in the Open Group Guide to Security Architecture in the TOGAF 9.2 Library [Ref. 12].

The Open Group / SABSA collaboration aligns and integrates the key concepts of the two frameworks. It recognises the need for a truly holistic approach. Nevertheless, there are practical impediments to adopting an integrated methodology, namely the lack of security support in popular EA modelling languages and their associated modelling tools.

Modelling is as essential to security analysis as it is to any other design and engineering discipline. Models amplify<sup>2</sup> the efficiency and effectiveness of practitioners and provide a valuable means of establishing a common understanding with stakeholders: a pre-requisite for quality. And yet the analyst's toolkit (Threat Models, Trust Models, Control Traceability, etc.) are too often a collection of disparate artefacts, detached from central EA models, which are consequently costly to create and maintain in synchronisation and coherence with the main body of system documentation.

Ultimately, real world systems have one, de-facto architecture: what ISO 42010<sup>3</sup> calls the 'System of Interest' (SOI). It makes sense to strive towards a single model<sup>4</sup> of that SOI, capable of describing, validating, querying and analysing all its pertinent functional and non-functional aspects, including the security perspective.

---

<sup>2</sup> The SABSA Institute: [SABSA Executive Summary](#)

<sup>3</sup> [ISO/IEC/IEEE 42010:2011](#) - Systems and Software Engineering -- Architecture Description

<sup>4</sup> or a series of interconnected models forming a complete Architectural Description (AD).

## 1.2 What is ArchiMate?

ArchiMate is visual EA modelling language published as an open<sup>5</sup> standard by The Open Group. It supports the description, analysis and development of Enterprise Business & IT Systems in much the same way that the Unified Modelling Language (UML) describes software design or Business Process Modelling Language (BPML) describes business processes.

The ArchiMate Core Framework resolves enterprise architecture into three core layers: Business, Logical (applications & data), and Technical Infrastructure.

In each layer, three aspects are considered: *active elements* (actors, roles, applications) that perform *behaviour* (services realised by processes & functions) on *passive elements* (information, data) via a constrained set of relations. A simple example is shown in Figure 1.

Subsequent releases have supplemented this core with additional layers and perspectives that now include: Strategy, Motivation, Physical and 'Implementation & Migration'. A brief primer on the notation is presented in Section 3: 'An Introduction to the ArchiMate Language'.

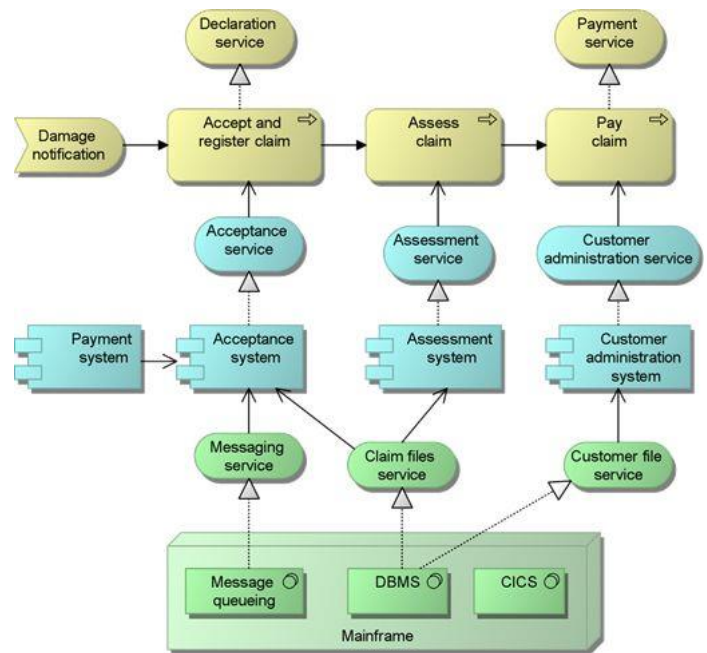


Figure 1: A small ArchiMate Diagram

The next release (version 3.1, November 2019) will mark a decade of the language's maturation<sup>6</sup>. Its popularity has been achieved through its essential usefulness (developed through practitioners' experience with earlier versions), its concise but expressive notation (easy to learn, yet capable of communicating complex patterns with precision and clarity); the availability of certified, compliant modelling tools to suit all budgets (including freeware) and the portability of models (courtesy of a standardised ArchiMate Exchange Format).

<sup>5</sup> The Open Group subscribes to the FRAND definition of 'open' i.e. that it is licensed under 'fair, reasonable and non discriminatory' terms.

<sup>6</sup> ArchiMate was originally developed in the Netherlands by a project team from the Telematica Instituut in cooperation with several Dutch partners from government, industry and academia. In 2008, stewardship was transferred to the ArchiMate Forum within The Open Group. The ArchiMate® 1.0 standard was first published as a formal technical standard in 2009.



Currently there are more than 7500 certified ArchiMate practitioners worldwide, over 40 thousand ArchiMate resources have been downloaded from The Open Group's site. In the past year Archi<sup>7</sup>, an accomplished free modelling tool, is being downloaded up to 3000 times per month.

Despite this progress, the least developed aspect of the notation is its lack of specific support for security.

### 1.3 Purpose

This White Paper explores how the ArchiMate language can be used to express security concepts, enabling the creation of the artefacts used in SABSA, using only the inherent extensibility capabilities of the existing notation. The aim is to establish a unified EA/ESA modelling environment capable of supporting in practice, the holistic architecture methodology envisioned in theory.

This analysis explores the following questions:

- What is the rationale for using ArchiMate to model security concerns in general and SABSA in particular?
  - What motivates the goal of expressing security in a modelling language?
  - What does a model-driven approach offer to SABSA practitioners and other security stakeholders?
- Which kinds of security artefact are required from an ArchiMate model?
- What are the gaps and limitations of the current language specification?
- Which new elements, relationships, properties and viewpoints are needed to create these artefacts?

The analysis leads to two main focus areas:

- What can currently be achieved within the constraints of ArchiMate 3 and available tools?
- What might further be achieved via a security-specific extension to the core language and what would such an extension look like?

---

<sup>7</sup> Ref.9: The Archi Modelling Tool

## 1.4 References

Table 1: References

Ref.	Title	Source	Date
1	Enterprise Security Architecture: A Business-Driven Approach	Sherwood et al.	2005
2	What is the benefit of a model-based design of embedded software systems in the car industry?	Manfred Broy et al. Technical University Munich	2011
3	TOGAF® and SABSA® Integration	The Open Group & SABSA Institute	Oct 2011
4	How to Model Enterprise Risk Management and Security with the ArchiMate® Language	The Open Group	Mar 2017 Updated Nov 2017 <sup>8</sup>
5	The ArchiMate® 3.0.1 Specification	The Open Group	Aug 2017
6	ArchiMate 3.0 Reference Cards	The Open Group	Aug 2017
7	Mastering ArchiMate – Edition III	Gerben Wierda R&A Enterprise Architecture	Oct 2017
8	Digital Identity Guidelines	NIST Special Publication 800-63-3	Dec 2017
9	The Archi Modelling Tool	<a href="https://www.archimatetool.com">https://www.archimatetool.com</a>	
10	R101: SABSA Matrices 2018	The SABSA Institute Academic Board	June 2018
11	Holistic Enterprise Development	Eero Hosiaisloma Frameworks, Methods, Tools & Modelling	Oct 2018
12	Integrating Risk and Security within a TOGAF® Enterprise Architecture	The Open Group Library	March 2019

<sup>8</sup> Updated for alignment with the ArchiMate 3.0.1 release.

## 2 The Rationale for SABSA-ArchiMate Alignment

Before immersing in a discussion of the “what” and “how” of security modelling, it’s worth taking a step back to reflect on the “why”. What do we hope to achieve by augmenting ArchiMate’s security modelling capability?

### 2.1 The Benefits of Modelling

Models are created in virtually every discipline of science and engineering to represent complex ideas or systems. They are used in various ways: to document, query, analyse, explain, explore, validate, plan, teach or predict the structure and behaviour of a system via an abstraction that obviates the need for interaction with the real system.

Interaction with a model can often happen earlier and be faster, cheaper, more flexible, agile and safer than doing so with a real system. A study<sup>9</sup> of the benefits of model-based engineering estimated an average 27% cost saving and 36% time saving (rising to 40% and 50% respectively if the model includes testing). It also found a tendency to produce “*better architected solutions*”: evaluation of a model promotes understanding, reveals problems that can be addressed early and allows the selection of solution building blocks (products and components) to be deferred until a more complete and well-defined set of requirements has been elicited. The characteristics of modelling that produce these benefits include:

- **Simplification**

Models focus attention on what really matters by abstracting above irrelevant or inconsequential details. The simplified view eliminates unnecessary complexity and makes what remains more comprehensible.

- **Efficiency**

Human visual processing is the most powerful of our faculties for making sense of the surrounding world. Numerous studies<sup>10</sup> have shown that visual information is absorbed and analysed from words and pictures with far less cognitive effort than from words alone. From an author’s perspective also, an ability to generate documentation from a model results in artefacts that are consistent, easily refreshed, up-to-date and generally maintainable at higher quality and with less effort.

- **Better Collaboration**

Because a good visual language is concise, expressive yet (semi)-formal (i.e. has a precise meaning), it provides a more effective communication medium than a verbose textual description. This advantage is especially beneficial across international projects with multi-lingual teams.

---

<sup>9</sup> Ref. 2: Benefits of Model-based Development of Embedded Software Systems in Automobiles: Broy, Kirstan TU Munich

<sup>10</sup> Including: Levie & Lentz, 1982; Levin, Anglin, & Carney, 1987

- Reuse

Experienced modellers express recurring ideas using established notational patterns. Consistent use of such design patterns facilitates comprehension through recognition: implicitly conveying contextual information such as problem analysis, design trade-offs or emphasis on resulting system characteristics such as flexibility, extensibility, separation of concerns, policy compliance, etc.

- Stakeholder Buy-In

The ability to create and maintain viewpoints addressing specific stakeholder concerns and perspectives (with minimal additional effort) helps provide assurance for stakeholders about the system or project, manage their expectations and help maintain their buy-in.

- Quality of Documentation

Documentation quality is likely to be enhanced on two counts:

- Concise, expressive, comprehensible notation enables a more efficient, effective review process enabling greater involvement and feedback from stakeholders who otherwise may not devote enough time or cognitive effort to the (recurrent) review of a heavily textual document;
- By generating the documentation set from the model, artefacts are referentially consistent, more easily kept up to date, better targeted to specific stakeholders and available in multiple formats / media types (traditional document, intranet etc.) at a fraction of the effort of manual authorship.

- Standardisation

Standardised modelling promotes sharing beyond the immediate in-house or project team. It makes it possible to benefit from a wide pool of literature, blogs, tools, trainings, certifications, etc., as well as facilitating the exchange of artefacts with suppliers, customers and communities of practice.

## 2.2 Why ArchiMate needs a security architecture perspective

ArchiMate provides all the above-mentioned benefits to a wide variety of architects and designers and is specifically designed to support the TOGAF Architecture Development Methodology (ADM). But just as an under-developed business-driven security perspective in the original TOGAF specification proved a deficiency in its vision for a holistic methodology (subsequently addressed by Refs.3 & 12), the architectural notation that supports TOGAF artefacts still lacks substantial support for security concerns.

At present, most security artefacts must be produced outside the EA modelling environment: separate models, separate notations, separate tools and processes, resulting in parallel representations of the SOI that EA and ESA teams must strive to reconcile and maintain in synchronisation. This dissonance is to the detriment of the overall security posture of the organisation and requires a means of redress.

The Open Group's stated mission is '*boundaryless information flow*': global interoperable access requiring relevant information to permeate through boundaries to support business processes. The SABSA framework enhances this vision with a complete and coherent security methodology that produces a comprehensive set

of security artefacts. SABSA offers techniques such as domain modelling and trust modelling that support the analysis of business relationships for architectural synthesis.

By providing a means to express these security artefacts, ArchiMate could (and should) become a pivot for bringing these parallel but intrinsically linked worlds closer together.

### 2.3 What Benefits can Modelling bring to SABSA

SABSA is a methodology for developing risk-driven enterprise information security and information assurance architectures and for delivering security infrastructure solutions that support critical business initiatives. Perhaps SABSA's most striking achievement is its completeness: every conceivable security service and activity that could be encountered has a logical place and purpose in the SABSA Matrices, with traceability to parent requirements in the layer above and cascading requirements and control measures to the layer below.

As elegant and comprehensive as this framework is, performing the methodology generates a volume of complex, interconnected and interdependent information, presenting significant information management and documentation challenges. Each cell in the Architecture Matrix produces artefacts, a collection of information that must be consistent with its peers in the same layer, the requirements and assumptions of the layer above, traceable to consequences in the layers below and linked to relevant cells in the Management Matrix.

Each artefact must be version-controlled and subject to a review cycle. Review feedback may introduce or change requirements, forcing revisions that potentially impact multiple related documents. The application of these changes produces more revisions, triggering further review cycles – and so the wave continues.

IT departments are adopting Agile: rapid design iterations, shortened review cycles and immediate stakeholder feedback. This only compounds the pressures described above. What might be the consequences for security?

Doing SABSA well requires a formidable information and document management capability to keep all artefacts at a consistent version with mutual referential integrity at the required turnaround rate.

Unfortunately, a traditional approach that relies on maintaining dozens of independent artefacts (documents, spreadsheets, diagrams etc.) in coherence requires effort that scales exponentially with the number of items.

A model-driven approach has the potential to transform this information management problem. If the various artefacts can be generated as views of a single underlying model, the impact of a change in one area is immediately reflected globally. Traceability chains can be re-calculated. Stakeholder review becomes streamlined (concise and comprehensible) and when all is correct, diagrams and documentation can be revalidated and regenerated with the assurance that they are mutually consistent by virtue of being derived from a single source of truth. Security is such an essential aspect of modern enterprises that the other benefits of simplification, efficiency, collaboration, reuse and buy-in should not be overlooked.

Arguably, the lack of suitable tooling has held back the adoption of SABSA as the security architecture framework of choice for some organisations, yet despite this the method has gained wide global popularity. The development of sophisticated tooling that is possible using ArchiMate should lower this barrier to adoption.

Standardisation of an ArchiMate security extension is still a distant prospect. In the meantime, this paper explores what can be achieved within the constraints of ArchiMate v3.0.1<sup>11</sup>. The ideas offered here can hopefully be refined and developed into a settled consensus of the SABSA practitioner community.

---

<sup>11</sup> At the time of writing, version 3.0.1 is the latest version for which the full specification is published

## 3 An Introduction to the ArchiMate Language

The paper assumes no familiarity on the part of the Reader with the ArchiMate language. This section presents a brief introduction to the structure and grammar of the notation to help familiarise readers to the level needed to evaluate the arguments that will be put forward, Tables and figures in this section are reproduced from References 4 and 5 in Table 1.

### 3.1 Core Elements

ArchiMate's core elements belong to one of the following aspects:

- *Active Structure Elements*, which are entities capable of performing behaviour, such as the human and technical Actors found in security models;
- *Behaviour Elements*, which are units of activity performed by Active Structure Elements. (Events are specific types of Behaviour Elements that denote a state change);
- *Passive Structure Elements*, upon which behaviour is performed such as business information, data and its physical realisation in files or databases.

This Active->Behaviour->Passive syntax provides the basic grammar of the core language rather like the *Subject-Verb-Object* construct of a natural language. The Active & Behavioural types are further specialised to enable service-oriented architectural viewpoints:

- *External Behaviour Elements* (Services): units of functionality exposed to the external environment. Services provide a 'Service Definition' while concealing the details of their internal implementation;
- *Internal Behaviour Elements* (Processes & Functions) describing how services are realised internally;
- *External Active Structure Elements* (i.e. Interfaces) are the points of access where systems expose one or more services to their environment;
- *Internal Active Structure Elements* (Actors, Roles, Applications & Devices) which offer the interfaces and perform the internal behaviour offered through services.

### 3.2 Core Relationships

ArchiMate elements relate to each other through a core set of relationships, grouped into four categories:

- *Structural relationships* model the static construction or composition of elements: *realization*, *assignment*, *aggregation* and *composition*;
- *Dependency relationships* model how elements support other elements: *serving*, *access* and *influence*;
- *Dynamic relationships* model information and control flows between elements: *flow* and *trigger*;
- *Other relationships* not classified into any of the above categories: *specialization* and *association*.




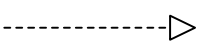
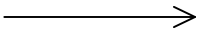

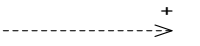
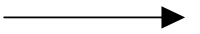



The language does not allow elements and relationships to be combined arbitrarily: each relationship is constrained to connect a predefined set of valid source and target elements, usually in a specific direction.

Relationships are directional and constrained to connecting exactly two endpoints, though this latter rule can be overcome by using ‘junction connectors’ that allow, for example, the Boolean combination of event triggers.

## 3.2.1 Definition and Notation of Relationships

The definition and notation for each of these relationships is shown in Table 2.

Table 2: ArchiMate Relationships

Structural Relationships		Notation
Composition	Indicates that an element consists of one or more other elements.	
Aggregation	Indicates that an element groups several other elements.	
Assignment	Expresses the allocation of responsibility, performance of behaviour or execution.	
Realisation	Indicates that an entity plays a critical role in the creation, achievement, sustenance or operation of a more abstract entity.	
Dependency Relationships		Notation
Serving	Models that an element provides its functionality to another element.	
Access	Models the ability of behaviour and active structure elements to observe or act upon passive structure elements.	
Influence	Models that an element affects the implementation or achievement of some motivation element.	
Dynamic Relationships		Notation
Triggering	Describes a temporal or causal relationship between elements	
Flow	Data transfer from one element to another	
Other Relationships		Notation
Specialisation	Indicates that an element is a kind of another element.	
Association	Models an unspecified relationship, or one that is not represented by an existing ArchiMate relationship.	
Junction	Used to connect relationships of the same type.	● And ○ Or

To keep the number of relationships in the vocabulary manageable, some are overloaded, i.e. although their usage remains conceptually similar, their exact meaning adapts to the context of the source and destination elements. This philosophy will be extended when adapting relationships for security contexts.

## 3.2.2 Derived Relationships

Structural and dependency relationships have a transitive property in the ArchiMate language that allows them to be traversed as a chain along their directionality as *Derived Relationships*.

Generally speaking, this means that if an Element A is linked to an Element B via one of these relationships ( $A \rightarrow B$ ), and B is also linked to an Element C via another such relationship ( $B \rightarrow C$ ), then the existence of a relationship from A to C ( $A \rightarrow C$ ) can be implied by derivation. To determine what this derived relationship is, the specification assigns a relative strength to each relationship according to the scale shown in Table 3. The derived relationship is defined to be the weakest of constituent relationships present in the chain.



Table 3: Relative Strength in Derived Relationships

Relationship Strength (←strongest to weakest→)						
composition	aggregation	assignment	realisation	serving	access	influence

## 3.2.3 The Core Layers and their Elements

ArchiMate defines three **core** architectural layers (aligned with TOGAF) into which these elements are placed:

- The *Business Layer* describes the products and services available to external customers of the domain being modelled. These services are realized by business processes performed by business actors/roles upon business information;
- The *Application Layer* provides the Business Layer with IT services that are realized by software components that perform operations on logical data objects;
- The *Technology Layer* provides the infrastructure services (data processing, storage and communications) necessary to run applications. These services are realized by hardware, system software and data artefacts such as files, directories and databases.

The definition and notation for the full set of Business Layer elements is shown in Table 4.

Table 4: Business Layer Elements



















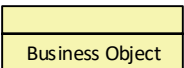
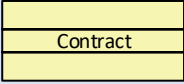
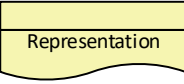
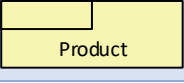
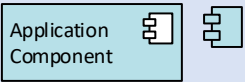
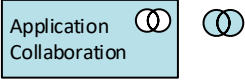
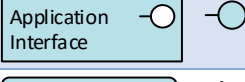
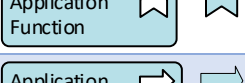
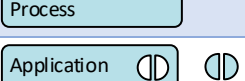
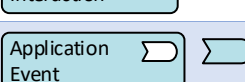
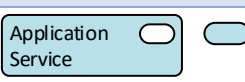
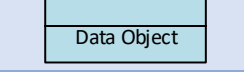

Element	Definition	Notation
Business Actor	A business entity that can perform behaviour.	Business Actor  
Business Role	The responsibility for performing specific behaviour, to which an actor can be assigned, or the part an actor plays in a particular action or event.	Business Role  
Business Collaboration	An aggregate of two or more business internal active structure elements that work together to perform collective behaviour.	Business Collaboration  
Business Interface	A point of access where a business service is made available to the environment.	Business Interface  
Business Process	A sequence of business behaviours that achieves a specific outcome such as a defined set of products or business services.	Business Process  
Business Function	A collection of business behaviour based on a chosen set of criteria (typically required business resources and/or competences).	Business Function  
Business Interaction	A unit of collective business behaviour performed by (a collaboration of) two or more business roles.	Business Interaction  
Business Event	A business behaviour element that denotes an organisational state change. It may originate from and be resolved inside or outside the organisation.	Business Event  
Business Service	An explicitly defined exposed business behaviour.	Business Service  
Business Object	A concept used within a particular business domain.	

Table 4: Business Layer Elements

Element	Definition	Notation
Contract	A formal or informal specification of an agreement between a provider and a consumer that specifies the rights and obligations associated with a product.	
Representation	A perceptible form of the information carried by a business object.	
Product	A coherent collection of services and/or passive structure elements, accompanied by a contract/set of agreements, which is offered as a whole to (internal or external) customers.	

The definition and notation for Application Layer elements is shown in Table 5.

Table 5: Application Layer Elements

Element	Definition	Notation
Application Component	An encapsulation of application functionality aligned to implementation structure, which is modular and replaceable. It encapsulates its behaviour and data, exposes services, and makes them available through interfaces.	
Application Collaboration	An aggregate of two or more application components that work together to perform collective application behaviour.	
Application Interface	A point of access where application services are made available to a user, another application component, or a node.	
Application Function	Automated behaviour that can be performed by an application component.	
Application Process	A sequence of application behaviours that achieves a specific outcome.	
Application Interaction	A unit of collective application behaviour performed by (a collaboration of) two or more application components	
Application Event	An application behaviour element that denotes a state change.	
Application Service	An explicitly defined exposed application behaviour.	
Data Object	Data structured for automated processing.	

Similarly, the definition and notation for Technology Layer elements is shown in Table 6.

Table 6: Technology Layer Elements

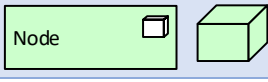
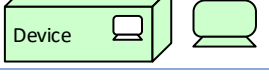
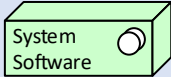

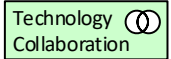

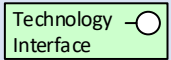

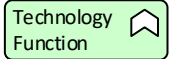

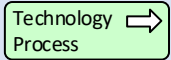

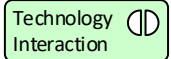

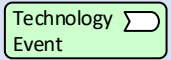

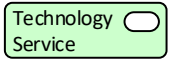

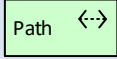

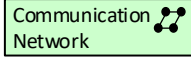

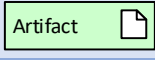

Element	Definition	Notation
Node	A computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources.	
Device	A physical IT resource upon which system software and artefacts may be stored or deployed for execution.	

Table 6: Technology Layer Elements

Element	Definition	Notation
System Software	Software that provides or contributes to an environment for storing, executing, and using software or data deployed within it.	 
Technology Collaboration	An aggregate of two or more nodes that work together to perform collective technology behaviour.	 
Technology Interface	A point of access where technology services offered by a node can be accessed.	 
Technology Function	A collection of technology behaviour that can be performed by a node.	 
Technology Process	A sequence of technology behaviours that achieves a specific outcome.	 
Technology Interaction	A unit of collective technology behaviour performed by (a collaboration of) two or more nodes.	 
Technology Event	A technology behaviour element that denotes a state change.	 
Technology Service	An explicitly defined exposed technology behaviour.	 
Path	A link between two or more nodes, through which these nodes can exchange data or material.	 
Communication Network	A set of structures and behaviours that connects computer systems or other electronic devices for transmission, routing, and reception of data or data-based communications such as voice and video.	 
Artefact	A piece of data that is used or produced in a software development process, or by deployment and operation of a system.	 

## 3.2.3.1 Use of Colour

An observant Reader may notice that the Business, Application and Technology elements are shown in different colours (yellow, blue & green respectively). It is important to note however that, while a number of colour-schemes have been adopted by convention, the ArchiMate specification itself is colour-agnostic, i.e. colour is defined as carrying no semantics, leaving the modeller free to use colour for visual emphasis.

The colours adopted in this section mirror those in the ArchiMate Reference Card [Ref. .6].

ArchiMate 3.0 introduced a Physical extension to the language to better represent technology nodes embedded in real-world objects (IoT, automation, raw materials distribution channels etc.) rather than Data Centre equipment. The Physical Layer, shown in Table 7, is an extension of the Technology. It defines only new Structural elements, reusing the same behaviour elements as the Technology Layer.

Table 7: Physical Layer Elements

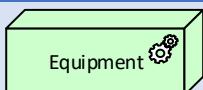
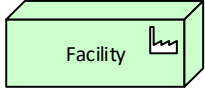
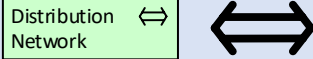
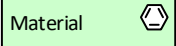
Element	Definition	Notation
Equipment	One or more physical machines, tools, or instruments that can create, use, store, move or transform materials.	

Table 7: Physical Layer Elements

Element	Definition	Notation
Facility	A physical structure or environment.	
Distribution Network	A physical network used to transport materials or energy.	
Material	Tangible physical matter or physical elements.	

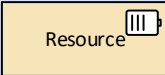
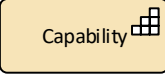
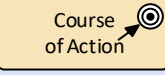
### 3.3 Extension Layers and Elements

In addition to these core elements, the ArchiMate specification defines other elements that are relevant in Enterprise Architecture practice.

#### 3.3.1 Strategy Layer Extension

The Strategy elements (Capability, Resource & Course of Action) are used to model business strategy and capability-based planning. The notation is shown in Table 8.

Table 8: Strategy Layer Elements

Element	Definition	Notation
Resource	An asset owned or controlled by an individual or organisation.	
Capability	An ability that an active structure element, such as an organisation, person or system, possesses.	
Course of Action	An approach or plan for configuring some capabilities and resources of the enterprise, undertaken to achieve a goal.	

#### 3.3.2 The Motivation Extension

Motivation elements form an independent aspect that models the motivation for enterprise design and operation and can be placed in any layer. The set of Motivation elements is shown in Table 9.

Table 9: Motivation Extension Element

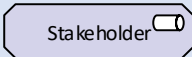

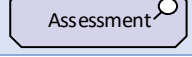

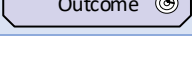
Element	Definition	Notation
Stakeholder	The role of an individual, team, or organization (or classes thereof) that represents their interests in the outcome of the architecture.	
Driver	An external or internal condition that motivates an organization to define its goals and implement the changes necessary to achieve them.	
Assessment	The result of an analysis of the state of affairs of the enterprise with respect to some driver.	
Goal	A high-level statement of intent, direction, or desired end state for an organization and its stakeholders.	
Outcome	An end result that has been achieved.	

Table 9: Motivation Extension Element

Element	Definition	Notation
Principle	A qualitative statement of intent that should be met by the architecture.	
Requirement	A statement of need that must be met by the architecture.	
Constraint	A factor that prevents or obstructs the realization of goals.	
Meaning	The knowledge or expertise present in, or the interpretation given to, a core element in a particular context.	
Value	The relative worth, utility, or importance of a core element or an outcome.	

### 3.3.3 Implementation and Migration Extension

The Implementation and Migration elements model the implementation of all aspects of Enterprise Architectures, as well as the migration between generations of implemented architectures. They include representations of Work Package, Deliverable, Plateau and Gap as shown in Table 10.

Table 10: Implementation & Migration Elements

Element	Definition	Notation
Work Package	A series of actions identified and designed to achieve specific results within specified time and resource constraints.	
Deliverable	A precisely defined outcome of a work package.	
Implementation Event	A behaviour element that denotes a state change related to implementation or migration.	
Plateau	A relatively stable state of the architecture that exists during a limited period of time.	
Gap	A statement of difference between two plateaus	

### 3.3.4 Miscellaneous Elements

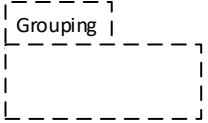
The Miscellaneous elements<sup>12</sup> shown in Table 11 can be used to annotate any layer.

Table 11: Miscellaneous Elements

Element	Definition	Notation
Location	A place or position where structural elements can be located or behaviour can be performed.	

<sup>12</sup> **Note:** ArchiMate tools generally support additional visual elements to annotate diagrams (e.g. comment boxes, visual groupings) that are not part of the formal specification

Table 11: Miscellaneous Elements

Element	Definition	Notation
Grouping	The grouping element aggregates or composes concepts that belong together based on some common characteristic.	

## 3.4 The ArchiMate 3.0 Framework

Figure 2: The ArchiMate 3.0 Framework (Figure 2) arranges the elements described previously to show the overall structure of the ArchiMate framework. Generally speaking, elements in any layer may only be connected to elements in the same layer or those in a layer directly above or below<sup>13</sup>.

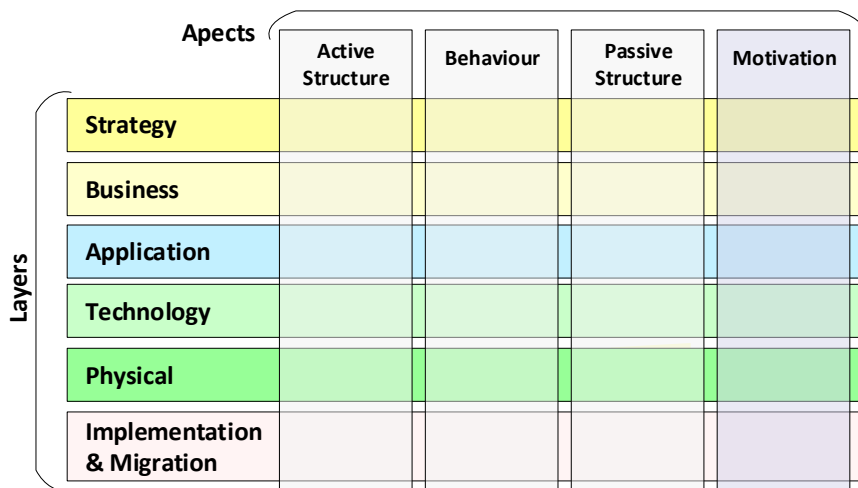


Figure 2: The ArchiMate 3.0 Framework

To complete this Section, Figure 3 reproduces the mapping of ArchiMate to the TOGAF ADM.

Since the mapping between TOGAF and SABSA has been defined [Refs.3 12], any mapping between ArchiMate and SABSA must be consistent with this alignment.

<sup>13</sup> This statement refers to best practice when creating the underlying ArchiMate model.  
In diagrams, visual shortcuts are possible and valid when implemented through the use of derived relations.

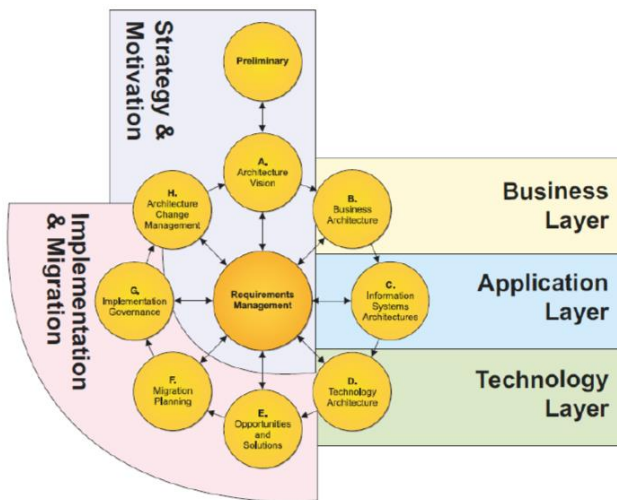
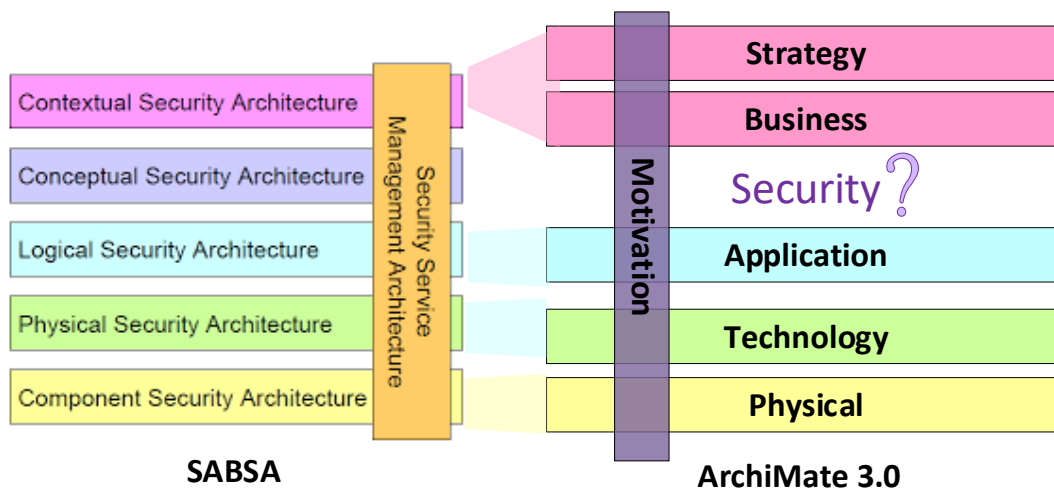


Figure 3: TOGAF - ArchiMate Overlay

## 4 Aligning the SABSA and ArchiMate Frameworks

### 4.1 An Overview of the Task

From a high-level perspective, the task of aligning SABSA and ArchiMate frameworks is illustrated in Figure 4.



**Figure 4: Comparison of the SABSA and ArchiMate Frameworks**

As a first approximation and allowing for different interpretations of the term ‘Physical’, the layers of the SABSA and ArchiMate frameworks align neatly. The most glaring exception however is that, except for the Motivation aspect (which fulfils with the purpose of SABSA’s ‘Why’ column), ArchiMate has no equivalent of SABSA’s Conceptual Security Architecture layer in which to model the Security Architect’s view. One aim of this paper will be discussing the practical options for addressing this gap within the constraints of the ArchiMate specification and with an eye on modelling-tool capabilities.

A less obvious deficiency is that, although the ArchiMate Core (Business, Logical, Technology) has direct SABSA equivalents, the elements in these layers lack ‘essential’ security properties: a ‘Business Object’ ought reasonably to carry a data classification; a ‘Business Role’ marked if providing privileged access. Although ArchiMate supports the embellishment of elements and relationships using properties, remarkably few are defined in the standard. Much of this paper will be concerned with identifying relevant security properties.

The ArchiMate Physical layer was added in v3.0 to support for material world entities (Equipment, Materials, Facilities, Distribution Networks). Its support for physical devices aligns with SABSA’s Component layer.

Finally, although SABSA’s Security Service Management architecture is not addressed specifically in the specification, it can be expressed in practical models. The recommendation of ‘Mastering ArchiMate’ [Ref..7] is to factor the Architectural Description (AD) into three architectural planes: a primary architecture that shows how the EA supports business processes at run-time; a secondary architecture that illustrates how the



primary EA is created, maintained and operated, and a tertiary architecture that deals with ownership and governance.

In this paradigm, artefacts for the SABSA Management Matrix would be modelled using standard notation in the secondary and tertiary architectures.

### 4.2 Risk & Security Modelling in ArchiMate 3

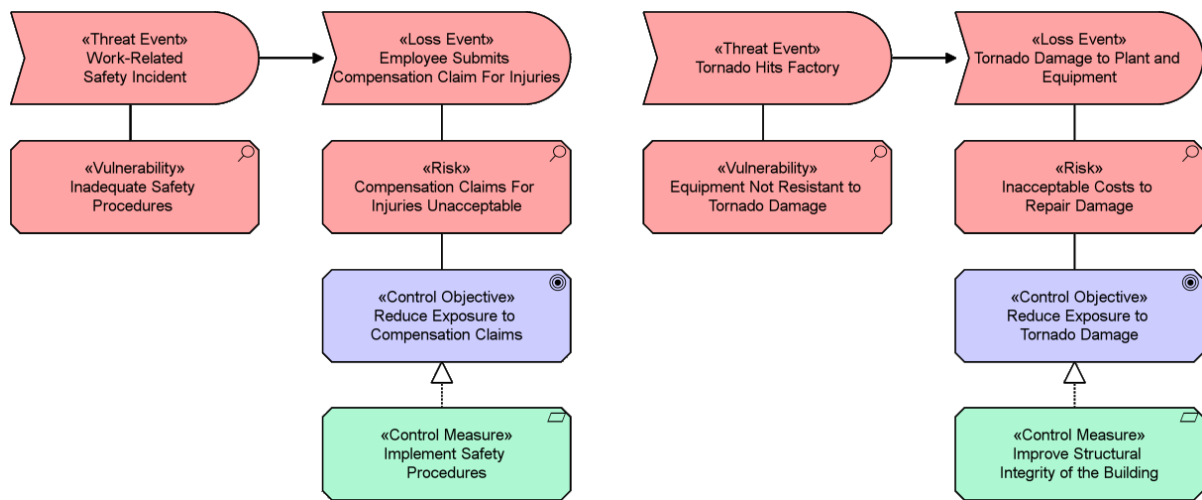
The ArchiMate 3.0.1 specification describes itself as a “visual modelling language with a set of default iconography for describing, analysing and communicating many concerns of Enterprise Architectures as they change over time”. This definition makes no special provision for security but nor does it exclude it from the “many concerns” of Enterprise Architecture. Although it makes no mention of security in its formal definition, the specification contains several examples with a security theme<sup>14</sup>.

- Information domains are cited as an appropriate use of the **Grouping** element, described as “a set of users, their information objects and a security policy”.
- **Driver** is defined as “an external or internal condition that motivates an organization to define its goals and implement the changes necessary to achieve them” with security concerns cited as an example. Note that ArchiMate offers **Driver** for use as either a positive or negative motivation, echoing SABSA’s dual view of risk as an enabler (exploit an opportunity) or requiring a defensive posture (mitigation).
- In describing ArchiMate’s capacity for extension of the core language by specialisation, an example is given of a notional Risk & Security overlay in which:
  - **Assessment** is specialised into Risk Assessment and Vulnerability Assessment;
  - **Business Event** is specialised into Threat Event and Loss Event;
  - **Goal** is specialised into Control Objective;
  - **Principle** is specialised into Business Policy;
  - **Requirement** is specialised into Control Measure;
  - **Grouping** is specialised into Domain.
- It also illustrates the specialization<sup>15</sup> of Business and Motivation elements to model the derivation of Control Measures from threat & vulnerability analysis, via risk analysis and control objectives.

---

<sup>14</sup> ArchiMate 3.0.1 Specification, section 15.2: Specialisation of Events & Relationships.

<sup>15</sup> The specification adopts the UML stereotype notation of using angled brackets to denote specialized elements.



**Figure 5: The Risk-Modelling example in the ArchiMate 3.0 Specification**

Examples like these cannot, of course, be taken as either normative nor complete: they illustrate only how the language *could* be used rather than how it *should* be used. Nevertheless, the example provides some insight into how the ArchiMate authors envisaged risk and security concerns being expressed.

The examples confirm that security modelling should stem from the specialisation of Motivation elements: with clear guidance on four elements (Assessment, Principle, Goal, and Requirement) and a recommended use for one other (Driver). The semantics of these elements are a natural fit for their specialisation into security. It also proposes Grouping for Trust Domains and specialised Events for the materialisation of threats and losses.

We might reasonably consider whether all security events are specialisations of Business Event: ArchiMate 3.0 introduces Application layer events (suitable for modelling input validation errors, time-outs) and Technical layer events (malware detection, data leak alert, back-up failure).

The example uses association to link Assessment to Event (a perceived vulnerability to the materialisation of a threat, a perceived risk to an actual loss event). Association is the only relationship that the specification permits from Assessment to Event (the forward direction). But what are these examples really trying to convey?

A classical security framework might postulate that a Threat Agent (Actor, Event) exploits a vulnerability (Assessment) to materialise a hypothetical risk (Assessment) into a concrete exploit (Outcome, Event). From this, the forward association might be termed ‘*exploited by*’ (vulnerability to damage can be exploited by a tornado), causing an Outcome (Damage to Plant & Equipment) that “materialises” the risk of major repair costs.

In the reverse direction (Event to Assessment), both association and influences are permitted, but the choice of association is reasonable despite not offering any visual indicator of directionality (the Facility’s vulnerability to storm damage is an innate property, over which the occurrence (or not) of tornados has no influence).

Association is used between Goal (Control Objective) and Assessment (Risk). The specification also allows the influences relationship in both directions, but only the forward semantics of “Control Objective influences Risk” make sense. The bi-directional navigability of association is useful for two-way traceability between Risks and Control Objectives.

The example uses realisation to model the achievement of a Control Objective through Control Measures. This relationship is structural (strong) and makes sense semantically, especially when a single Control Measure is sufficient. Complications arise when multiple measures are necessary to achieve the required objective: here the default semantics of realisation imply that each measure is an OR’ed alternative (as in different software implementations of the same service definition). Control Measures often need AND semantics: a set of requirements that are all necessary to achieve a chain of end-to-end protection or defence in depth.

The specification allows two other relationships: influence and association. Influence can be used in either direction; association is inherently bi-directional<sup>16</sup>. Let us consider whether either of these would be a better fit.

Association is ArchiMate’s ‘*relationship of last resort*’: made available for linking elements without any precise meaning beyond “*an unspecified relationship or one that is not represented by another ArchiMate relationship.*” The disadvantages of using association would be:

- inability to distinguish this specific use from other uses of ‘*the relationship of last resort*’;
- loss of visual directionality i.e. that Control Measures contribute to a Control Objective but that the reverse is not true;
- Since v3.0, *association* is no longer derivable – in other words, asking a standard tool to navigate from model elements that contribute to the achievement of a specific Control Objective may no longer be supported, unless provided as functionality outside the specification.

In practice, Control Measures and Control Objectives can conflict, in that some Measures impact some Objectives negatively while reinforcing others positively. The *influence* relationship is the only one capable of expressing this within the standard notation and therefore merits a place.

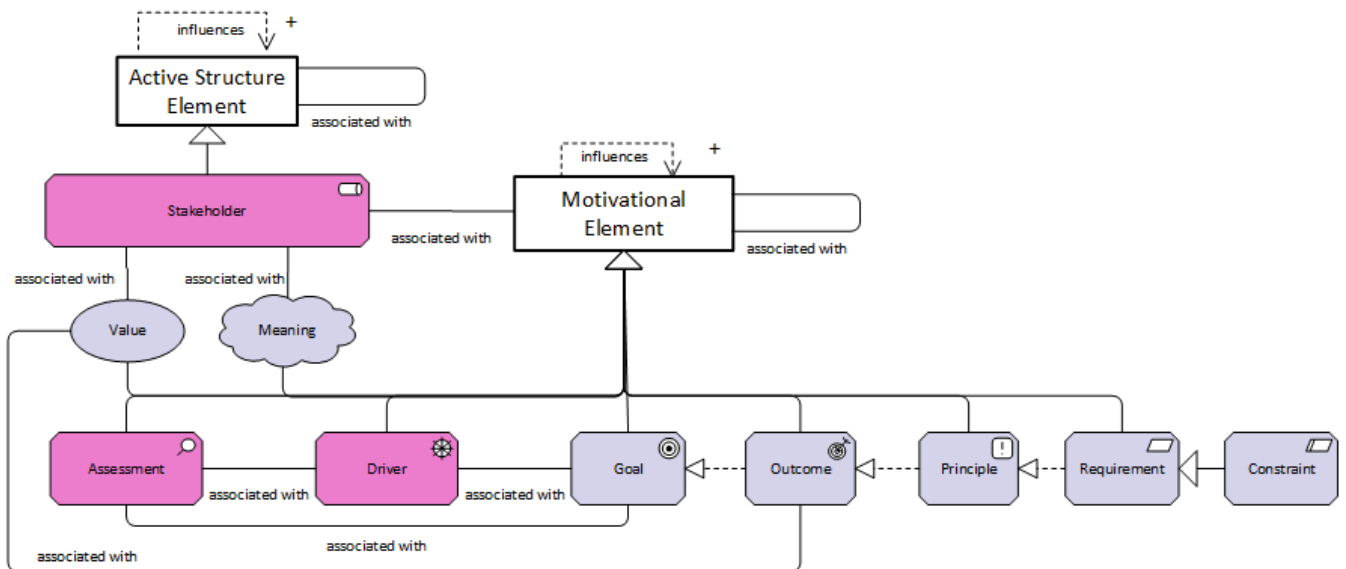
*Influence* is, since, v3.0 derivable. The downside of *influence* is its relative weakness and vagueness: it lacks a solid assurance that implementing a set of Measures ensures that the Control Objective is achieved.

Realisation then appears to be the best choice for the structural relationship between Objectives and Controls, with influence reserved for ‘cross-coupling’. A solution to expressing AND / OR semantics is discussed on Page 38.

---

<sup>16</sup> Although the specification defines association as non-directional, the Exchange format always identifies a source and target element on all relationships. In effect, association is visually & semantically bi-directional but, in the model, it is not.

Finally, the specification suggests *Driver* as the appropriate element to model security concerns but offers no advice about how this should be connected to other elements. For guidance, the ArchiMate meta-model of Motivation elements is reproduced in Figure 6.



**Figure 6: ArchiMate Motivation Meta-Model**

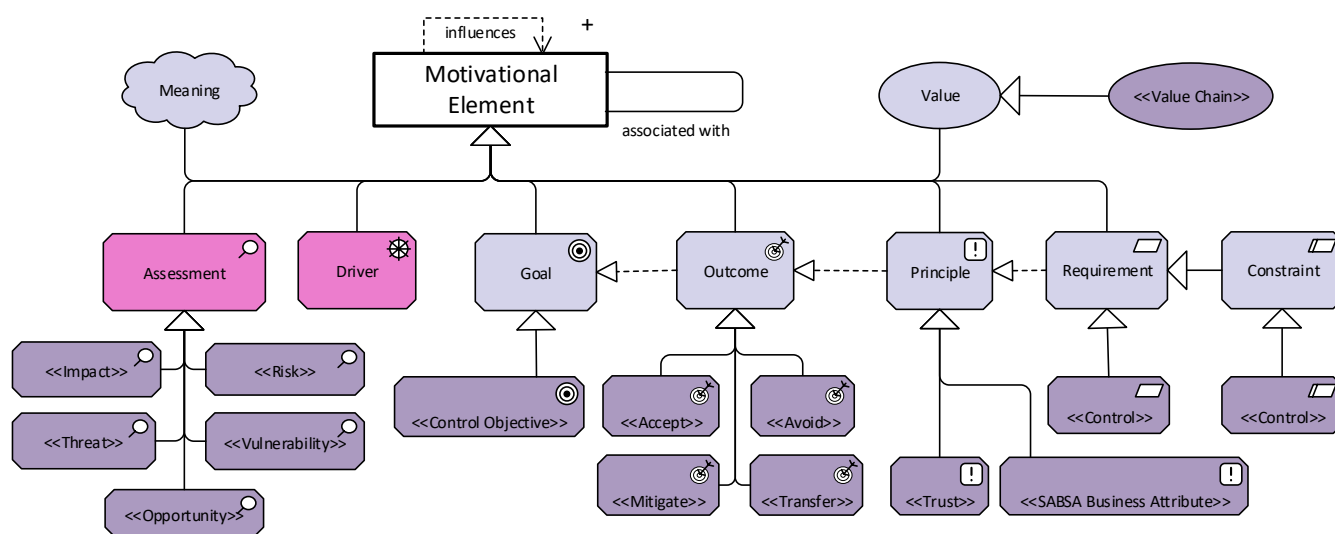
The metamodel confirms that *Driver* (as a specialisation of the abstract Motivation element), may be associated with a *Stakeholder* and may *associate with*, *influence* or be *influenced* by any other Motivation element.

## 5 A Security Overlay for ArchiMate 3

This section of the paper discusses a security overlay for the ArchiMate core layers and Motivation model, capable of supporting a model-based framework for SABSA.

### 5.1 The Security Overlay of the Motivation Metamodel

Building on the previous section, Figure 7 shows the full security overlay for the Motivation Metamodel that formalises the non-normative examples into a specification that better supports the security perspective.



**Figure 7: Security Enhanced Motivation Metamodel**

This section will describe how these elements can be used to express security motivations: the identification and valuation of assets, analysis of threats & vulnerabilities, evaluation & treatment of risks and the identification of Control Objectives and Controls.

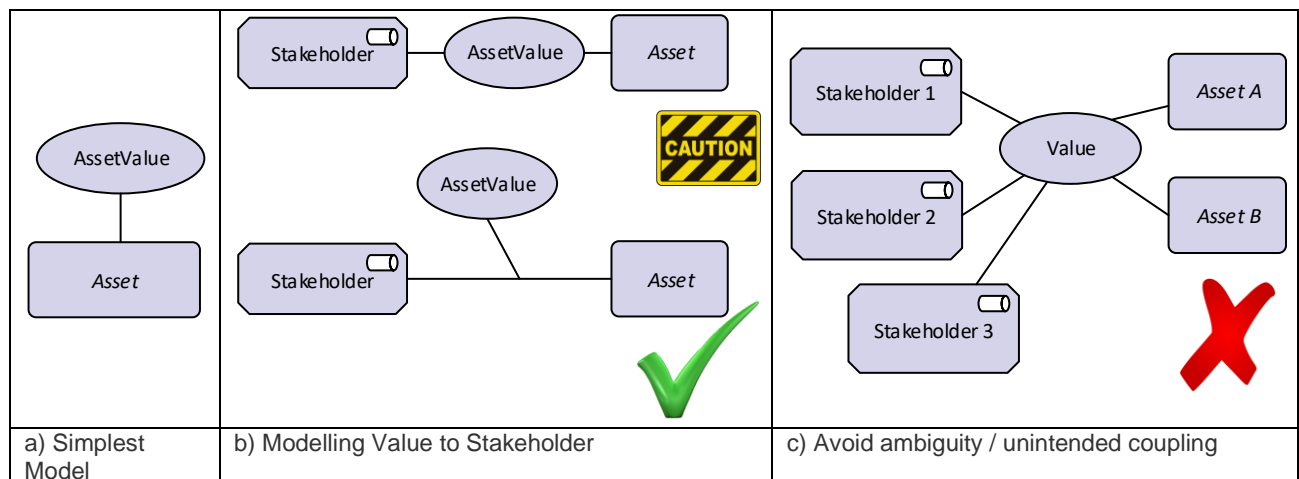
#### 5.1.1 Value

The latest revision of the SABSA Architecture Matrix [Ref. 10] presents a shift of emphasis from the identification of Assets to the elevation of Business Value as the highest expression of worth. Business Value may be measured in several terms (financial, legal, brand, social, economic, health & safety) or any combination of these (and others). This evolution brings it closer to ArchiMate, which has no element to represent an abstract Asset but instead offers the ability to associate a **Value** element with any part of a model.

Because security is concerned with enhancing value and the protection of assets, the Value element is likely to be used extensively in a security model. A model containing many Values will encounter the issue of providing sufficient unique names for them, but this can be overcome through adoption of a naming convention which reflects the associated Element name. Examples of Value modelling are shown in Figure 8.

Figure 8a shows the simple association of a Value with a generic element, thereby marking it as an asset.

If the Modeller wishes to represent the Stakeholder to whom the Value is important, use of the tertiary relationship (Figure 8b lower) is recommended over the in-line association (Figure 8b upper). The former pattern scales better in situations where a Stakeholder appreciates the same Value in multiple assets, or a Value is appreciated by multiple Stakeholders. It avoids the risk of ambiguity / unintended coupling that could otherwise occur in the underlying model even if not explicitly drawn, as in Figure 8c.



**Figure 8: Modelling Assets using Value**

A good textual description of the nature and scale of the Value is essential for a complete model but to enable automated analysis, it is important to try to capture key characteristics of the element in a more structured way. In ArchiMate, this is achieved using custom properties. The following properties are proposed for the Value overlay.

*Table 12: Proposed Value Property Overlay*

Value	Properties	Schema
	The properties of the Value element should include a number of key-value pairs that describe the value type and quantifier. The set of keys could be drawn from a standardised set with local configuration setting them as required or optional. The type and range of quantifiers	<div>Value</div> <div>financial : Integer</div> <div>reputational: {LOW, MEDIUM, HIGH}</div> <div>legal: {LOW, MEDIUM, HIGH}</div>

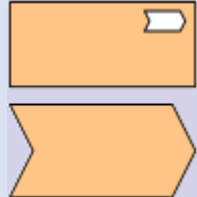
### 5.1.2 Value Chains & Value Streams

Value Chains and Value Streams can be denoted by ValueStream: a Strategy element expected in ArchiMate 3.1. The provisional definitions and notation are shown. in Table 13Table 13.

*Table 13: The anticipated Value Chain Element*

Element	Definition	Notation
Value Chain	A high-level view of an organisation: how it works, how it delivers value, how its functions are organised within an operating model.	

Table 13: The anticipated Value Chain Element

Element	Definition	Notation
Value Stream	A representation of an end-to-end collection of value-adding activities that create an overall result for a customer, stakeholder or end-user.	

According to TOGAF 9.2, Value Chains and Value Streams show why an organisation needs business capabilities. Capabilities describe what the organisation needs for a particular value stage to be successful. A preview example of the relationship between Value Chains and Capabilities has been published on the *‘Frameworks, Methods, Tools & Modelling’* blog [Ref. 11] and reproduced in Figure 9.

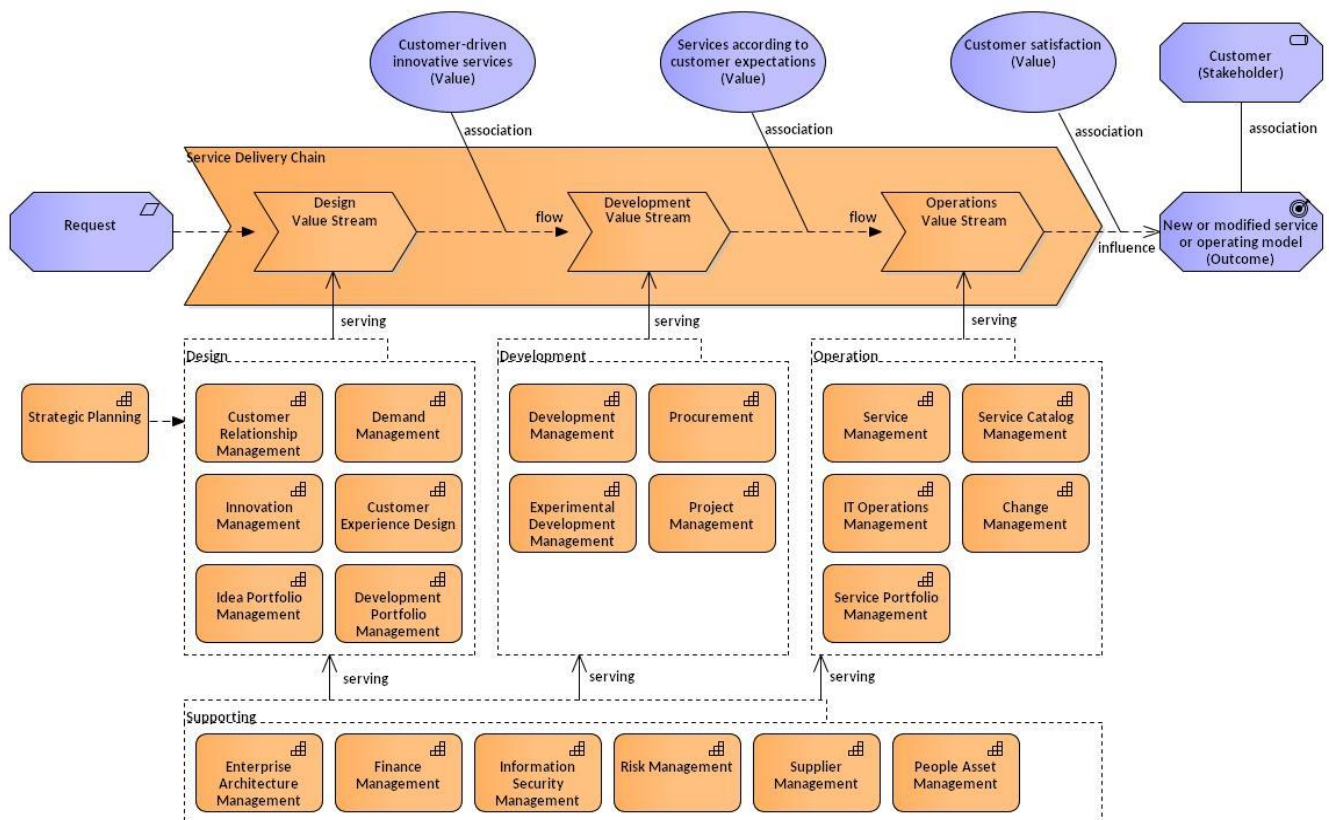


Figure 9: Value Chain Modelling

Value Chains are the focus of the SABSA Business Meta-Process architecture. The key attribute of a Value Chain is its *margin*: the difference between the cost of performing the activity (materials & processing costs) and the value realised from the activity's output. The analysis of Value Chains must enable an organisation to identify which stages contribute to business value and which perform inefficiently or are uneconomic. The security perspective is concerned with enhancing / protecting the value of the former while containing the costs of the latter. Unlike Value, it is predominantly a financial measure.



Using the Security Overlay, the composition of a Value Chain can be extended to any part of an EA model as shown in Figure 10.

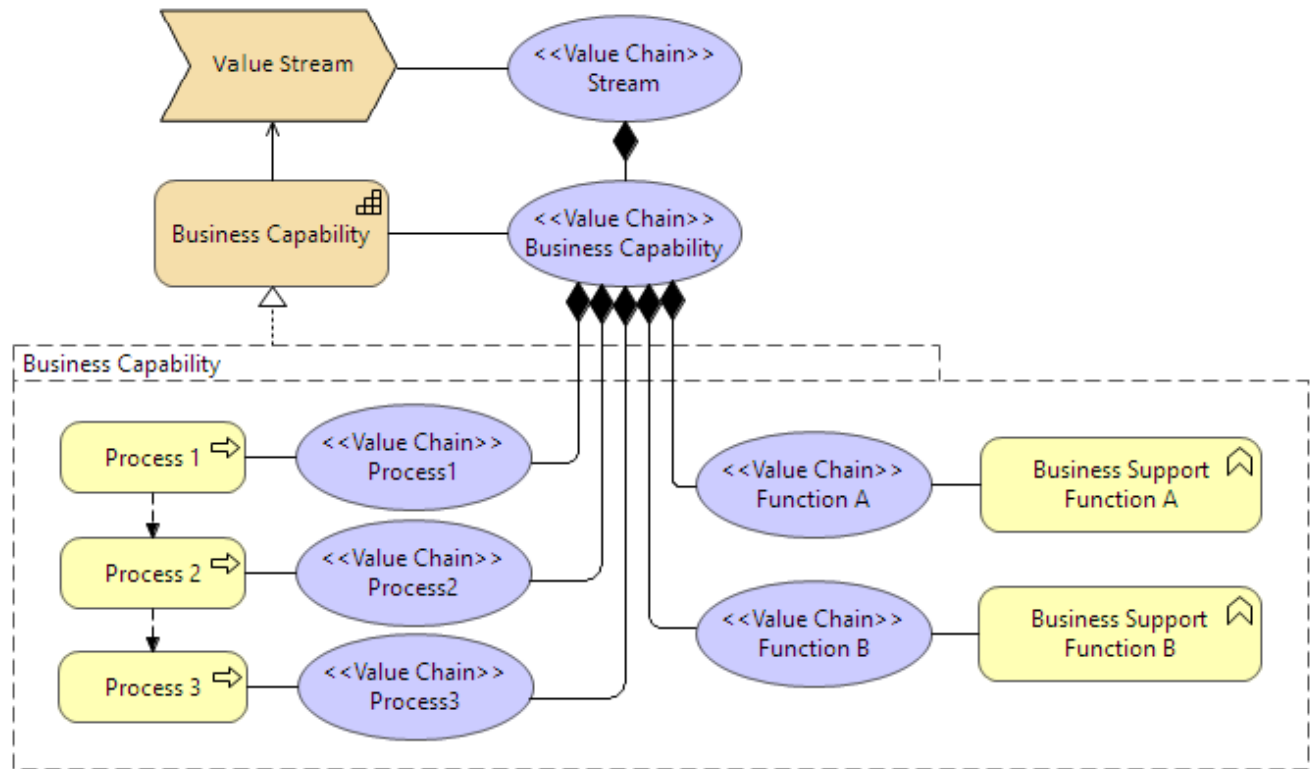



Figure 10: Value Composition of Streams and Chains

By externalising the properties of a Value Chain and Value Streams, it becomes possible to decompose and trace the contribution of supporting Business Capabilities and sub-chains associated with high-level business processes and support functions, which may in turn be decomposed into sub-processes and sub-functions to arbitrary depth.

The properties proposed for the Value Chain stereotype enable the margin to be calculated as the sum of its constituent sub-chains.

Table 14: Value Chain Properties

Value Chain	Properties	Schema						
	<p>The Value Chain element should be able to express the costs of production (input materials, process handling and consumption of support services) and the manifest worth in the final output.</p> <p>The margin can also be expressed as a derived value.</p>	<table><tr><td>Value Chain</td></tr><tr><td>materialCost : Integer</td></tr><tr><td>processingCost: Integer</td></tr><tr><td>secondaryCost: Integer</td></tr><tr><td>finalWorth: Integer</td></tr><tr><td>margin: Integer (Derived)</td></tr></table>	Value Chain	materialCost : Integer	processingCost: Integer	secondaryCost: Integer	finalWorth: Integer	margin: Integer (Derived)
Value Chain								
materialCost : Integer								
processingCost: Integer								
secondaryCost: Integer								
finalWorth: Integer								
margin: Integer (Derived)								

The ArchiMate 3.1 specification, when published, is likely to support the internalisation of these properties within the ValueStream element itself, but it remains to be seen whether the composition relationship to Value elements will be a legal construct.

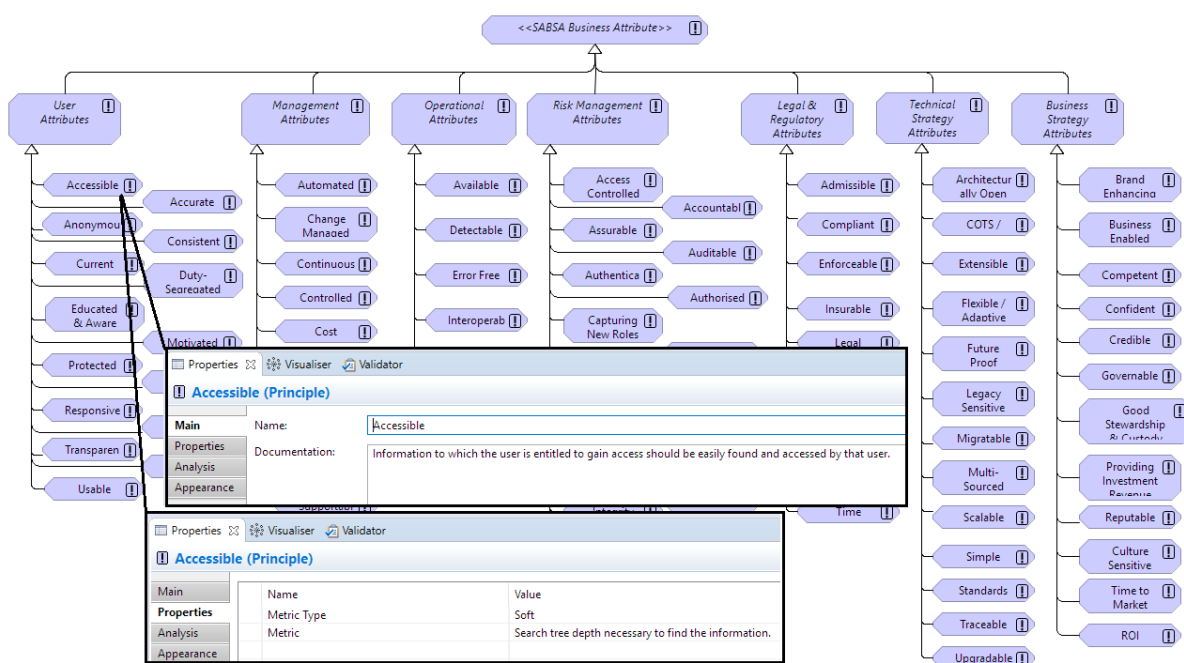


### 5.1.3 SABSA Business Attributes

Business Attributes represent the ideal, essential qualities desired by the Stakeholders of a system which must be protected and / or enhanced in the material world implementation if the enterprise is to fulfil its mission. They are essential to the SABSA approach but have no direct equivalent in ArchiMate.

At closer examination however, the definition of ArchiMate's Principle element as "*an intended property of a system ... a general property that applies to any system in a certain context ... motivated by some goal or driver*" is semantically compatible for modelling SABSA Business Attributes in the security overlay.

Figure 11 recreates the SABSA Business Attributes catalogue as a taxonomy of Principle elements.



**Figure 11: SABSA Business Attributes represented in ArchiMate**

Points to note about this representation:

- i. At the root element of this taxonomy, SABSA Business Attribute has been modelled as an abstract stereotype (indicated by *<<Italics within Double Chevrons>>*). This stereotyping is intended to distinguish SABSA Business Attribute and its concrete sub-types from conventional uses of Principle in EA, such as “Buy before Build” or “Cloud First”;
- ii. The taxonomy has been structured using six abstract sub-domains of the root. This is a design choice that may not be strictly necessary and could have been implemented differently – e.g. a visual grouping that would have retained the taxonomical structure while retaining a flat internal model, but ultimately, it is implemented this way to enable manageability and constraint. It offers potential benefits in practical modelling and in various analysis scenarios such as:
  - The domains act as a namespace that allows the construction of qualified names. This not only enables attributes to be overloaded, with appropriate definition & metrics, in different contexts but

also appear on the same diagram (more on this on [page 34](#)). For example, *Usable* might be re-used to represent similar system characteristics in a User, Management or Operations context;

- To encourage consistency & correctness in modelling, the formal structure can be used to ensure that Legal or Compliance stakeholders may only be associated with Legal & Compliance attributes;

Note also that the Attribute element has been modelled with properties that convey a description and an appropriate metric of either hard/soft type. These have been populated with values published in [Ref. 1](#).

### 5.1.3.1 Structural Placement of Business Attributes

In the SABSA approach, business-layer Attributes are refined and resolved at lower layers into contributing Attributes with sub-system scope but must ultimately be addressed by Control Objectives (**Goals**) achieved through the realisation of Controls (**Requirements & Constraints**).

This is consistent with the [dictionary definition](#) of ‘principle’ as: *‘something more abstract than policy and objectives and meant to govern both’*.

In an abstraction hierarchy, Principles are universal, singleton concepts at the apex, with Policies, Control Objectives and Control Measures layered beneath as shown in the Figure 12 (right).

Unfortunately, this concept of a principle is not universal: the widely-adopted COSO Enterprise Risk Management framework places its ‘Principles’ between Objectives and Controls as the example shown in Figure 13 (right).

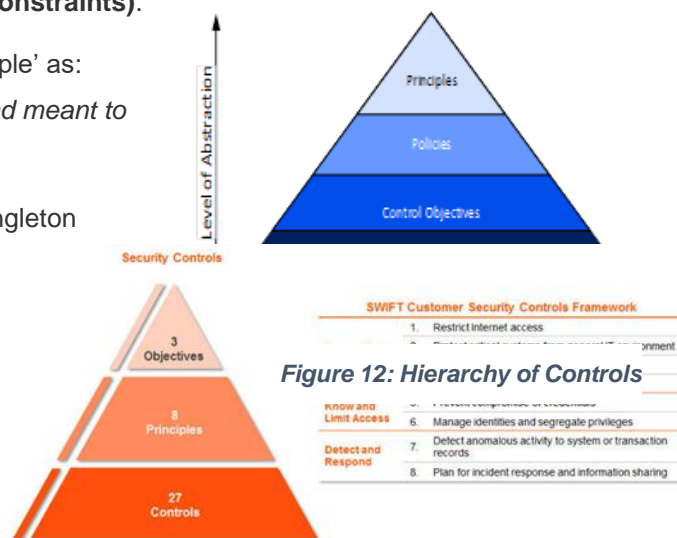


Figure 12: Hierarchy of Controls

Figure 13: The COSO concept of Principle in the Control Hierarchy

ArchiMate’s Motivation Metamodel has adopted

the COSO paradigm<sup>17</sup>: i.e. Principles are used as maxims to guide Designers *towards* Outcomes and Goals (Figure 14) rather than what is required for SABSA: an abstract, idealised quality that is to be enhanced or protected through the achievement of concrete, contextualised Control Objectives.

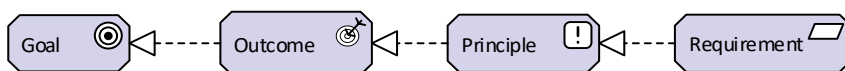


Figure 14: Principle in the ArchiMate Motivation Hierarchy

This presents a few problems for SABSA usage. Take for example the situation shown in Figure 15.

<sup>17</sup> Notwithstanding the example from the ArchiMate specification which proposes the Principle element to represent policy.

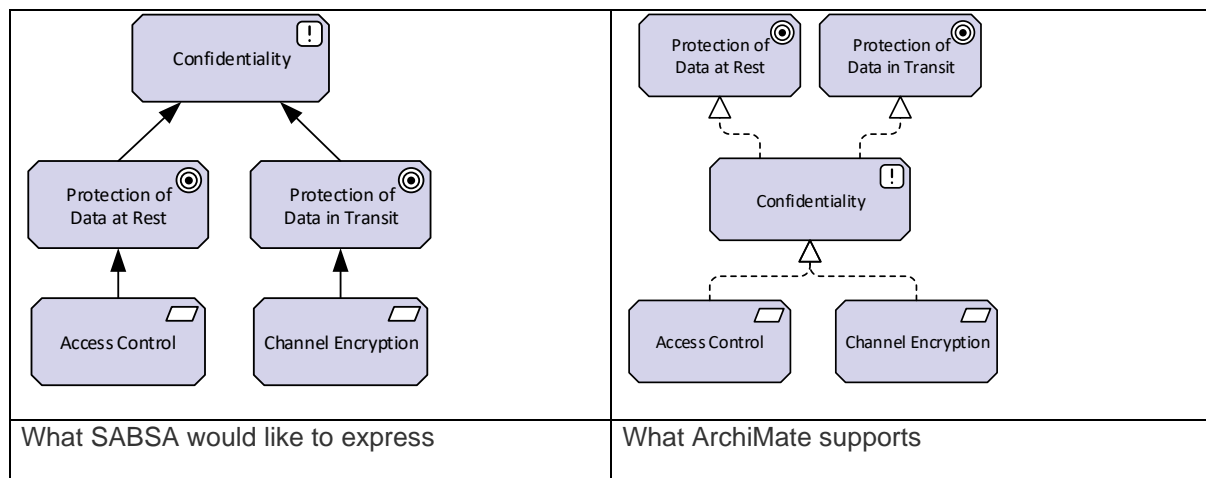


Figure 15: Highlighting the Control Hierarchy Mismatch (i)

The SABSA view (left) tries to express an abstract Attribute (*Confidentiality*) being enhanced by specific Control Objectives (*Protection of Data at Rest* & *Protection of Data at in Transit*) at various points in the model. These are achieved by hard, verifiable Requirements (*Access Control* & *Channel Encryption*) respectively.

ArchiMate's Metamodel (right) would place *Confidentiality* in the centre of the structure and by doing so, forces the realisation paths to merge. The resulting model reads as if '*Protection of Data at Rest*' can be realised through '*Channel Encryption*' and '*Protection of Data at in Transit*' can be realised through '*Access Control*'.

This is both unintentional and incorrect. It is also difficult to disentangle because, even if the two structures are *drawn* as distinct views, they remain merged in the underlying model. Any tool supporting model navigation or analysis capability will be unable to recover the modeller's intent.

A possible workaround to this problem is the creation of parallel stacks, each with its own copy of '*Confidentiality*' (Figure 16). Note that the copies must be given distinct names because, as ArchiMate elements represent 'classes' rather than 'objects', reputable tools do not allow the same element to appear more than once in the same diagram.

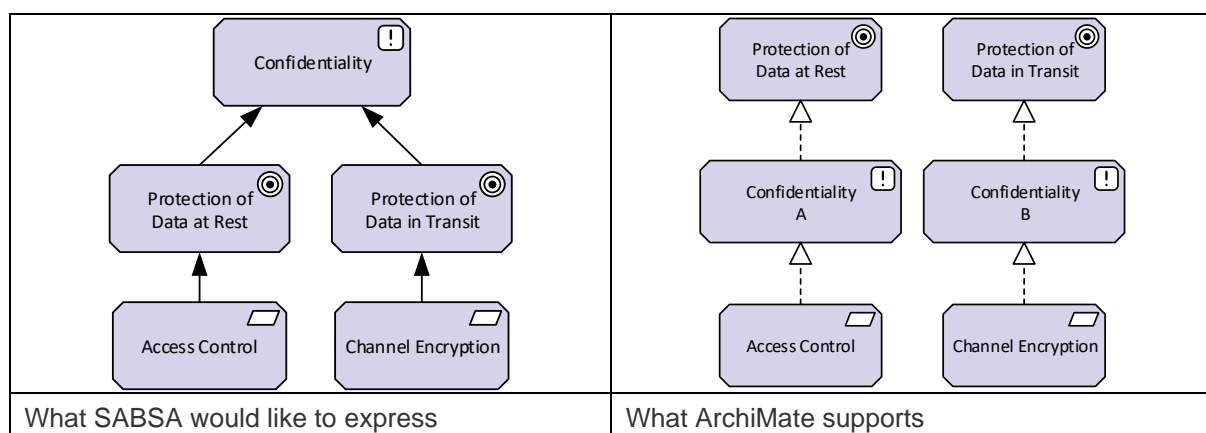


Figure 16: Highlighting the Control Hierarchy Mismatch (ii)

Although this approach achieves the required segregation and is entirely in line with the specification, the creation of distinct *Confidentiality* elements for each stack is inelegant. If a good model is meant to reflect reality, this is bad modelling: the concept of different ‘flavours’ of confidentiality is not natural in the real world. A better solution (Figure 17) eschews the standard structure but remains legal and reflects the original intent.

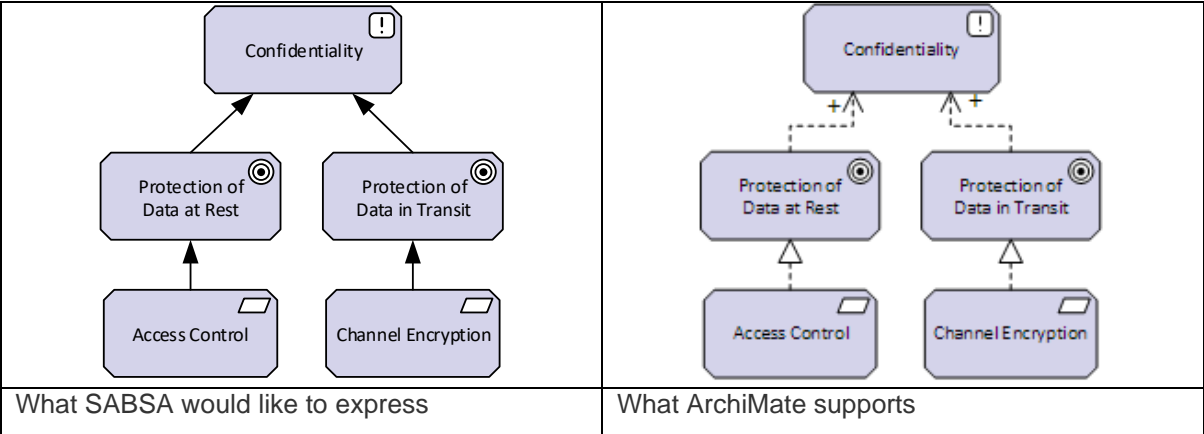


Figure 17: Achieving the desired Hierarchy

Via a convention in which SABSA Attributes are only ever **influenced** by Control Objectives, the desired hierarchy is achieved within the language’s grammar. The choice aligns well with the ArchiMate definition of **influences** as “a traceable motivational path” where the “motivation element is achieved to a certain degree. That is to say, assurance of SABSA Attributes is *enhanced* by Control Objectives rather than *guaranteed*.

5.1.3.2 Traceability of Business Attributes

Another essential requirement of SABSA Attribute modelling is the ability to trace their refinement through the architectural layers. This is also implemented using influence relationships as shown in Figure 18

**Error! Reference source not found..**

An issue that is encountered when creating moderately-sized attribute hierarchies concerns element reuse.

Consider a situation where ‘*Compliant*’ is an appropriate SABSA Attribute to apply in two places in the model: e.g. to indicate GDPR privacy compliance on personal data in the Business layer and FIPS compliance on a Hardware Security Module in the Technology layer. The Modeller encounters the following problems:

- Modelling tools generally forbid the same element to appear multiple times on the same drawing;
- Using the element once but with associations to both targets is both possible and ‘legal’ but:
  - the Attribute’s properties, e.g. the definition or metric, may be incompatible with both situations;
  - the reused Attribute becomes a knot of merged relationships, warping the structure of the hierarchy;
- Creating artificial ‘flavours’ of global, abstract concepts such as Attributes is inelegant design practice.

Because SABSA Attributes are global singletons, reuse within a model runs the risk of unintended coupling. Nevertheless, if its definition is compatible and it can be shared without warping the layer hierarchy, a Attribute can be reused, accommodating different metrics using an externalised ‘Meaning’ element.

In all other cases, the best solution is to create distinct instances in the Attribute taxonomy (Figure 11). In this example, this would mean creating ‘Compliant’ in the Technical Strategy domain for use with the HSM as well as the existing one in the ‘Legal & Compliance’ domain. Once they are distinct elements (same short name, different qualified names), both Attributes can appear on the same diagram.

The influence relationship is suited here to convey the same sense of intangibility that justified its use with Control Objectives. There is a subtle difference, however. Attributes are abstract concepts: singletons, in a taxonomy that can be reused across models, projects and enterprises. When connected by ‘concrete’ relationships in a model, they also become connected in the abstract: universally and forever.

It’s perhaps an esoteric point but it is important to note that, in reality, SABSA Attributes are just concepts with no intrinsic correlation. The relationship only arises as an “echo” of a real correlation in the context of the specific material system being modelled.

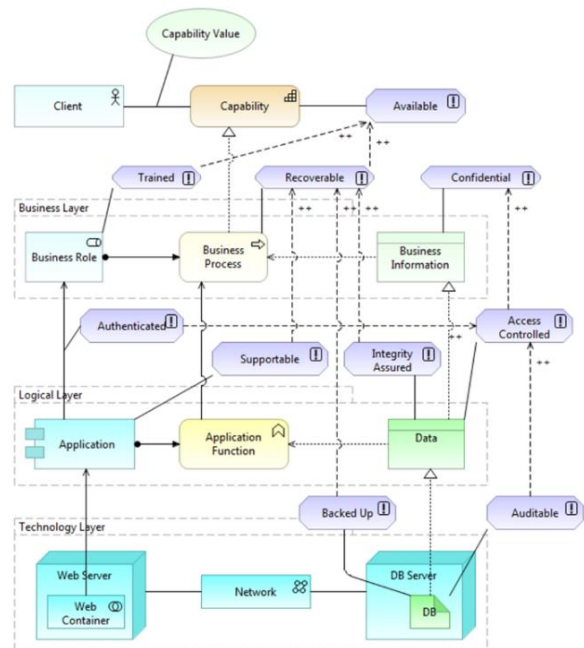


Figure 18: Attribute Traceability across Layers

### 5.1.4 Meaning

ArchiMate's Meaning element has not been extended in the overlay other than by the addition of optional properties. The element plays an essential role in the re-use of Attributes and the local customisation of metrics.

As described in the earlier section, the SABSA Business Attribute taxonomy proposes default definitions and hard/soft metrics via user-defined properties. While these properties can and should be customised for each new context, because each Business Attribute is modelled as a Singleton, the definition and metrics will apply globally, in every context where the Attribute is referenced.

As we have seen, this presents a problem where the Attribute must be applied at multiple points in the same model. Consider for example, whether there is a single *Confidentiality* definition or metric that would be applicable to the two Control Objectives and four Controls of Figure 19.

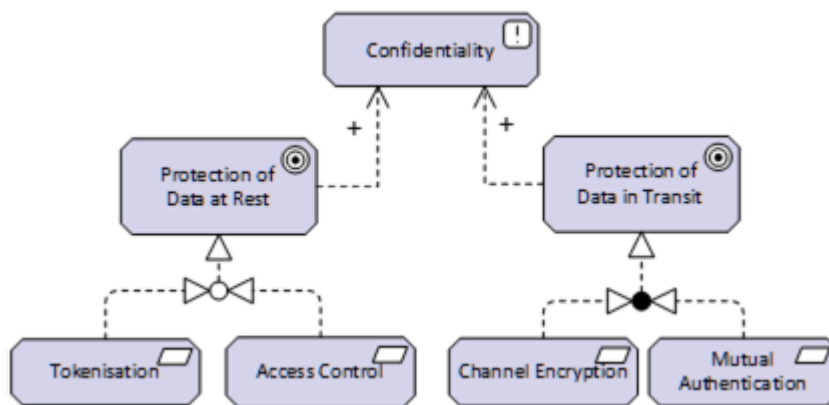


Figure 19: Applying Attribute Metrics to Multiple Controls

To overcome this, ArchiMate's **Meaning** element can be enlisted to interpret an Attribute in a particular context. So, the meaning of *Confidentiality* in the context of *Tokenisation* (i.e. its definition and metrics) can be different when applied to *Access Control* or any of the other requirements as shown in Figure 20.

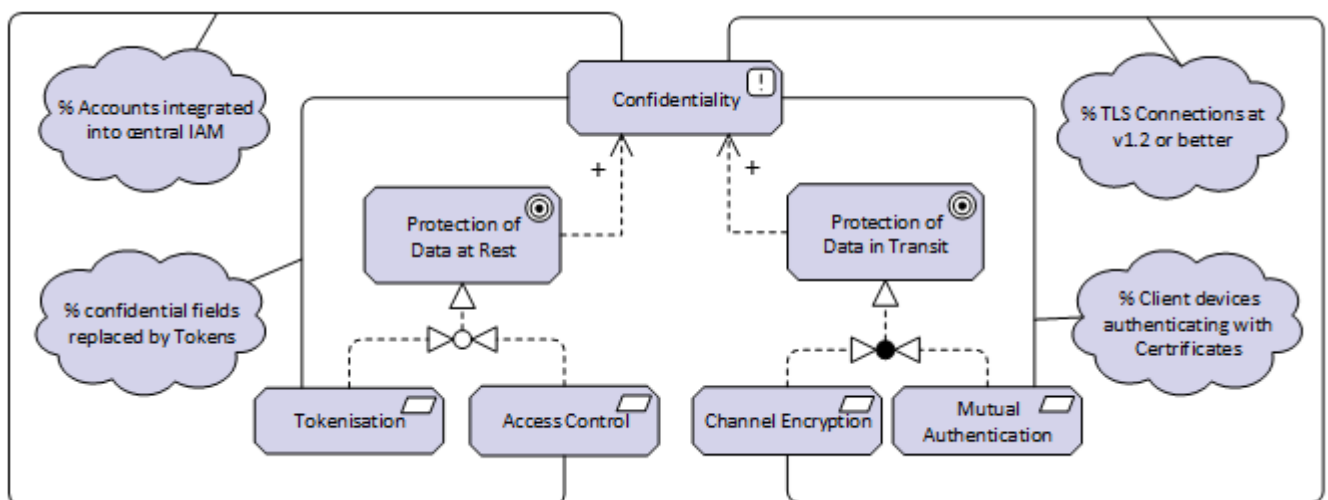


Figure 20: Use of Meaning to define Distinct Metrics

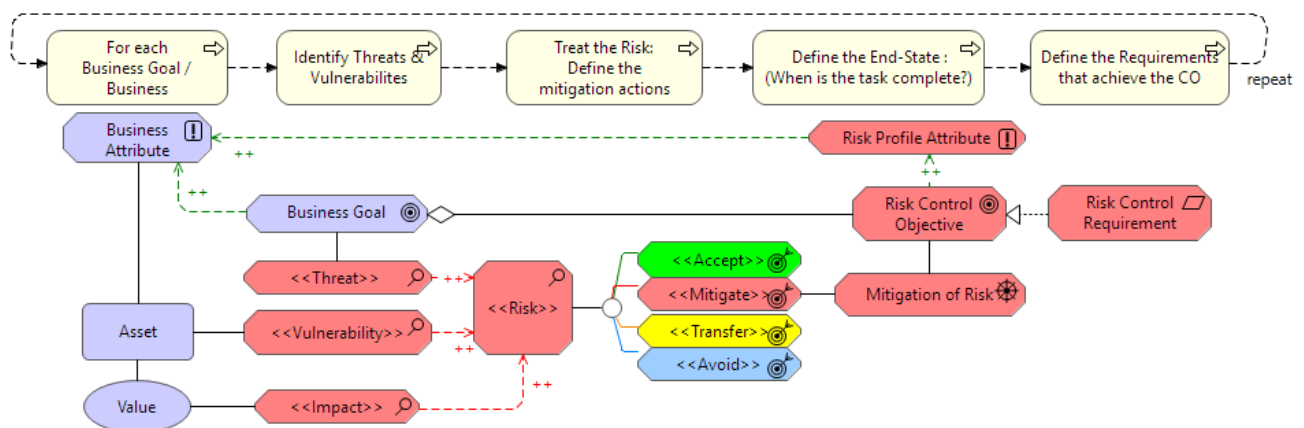
### 5.1.5 Impact, Threat, Vulnerability & Risk Analysis

Having identified and modelled the system's Value and the SABSA Attributes that require protection, this section discusses the modelling of a risk-driven approach.

The extended Motivation Metamodel of Figure 7 adopts a convention of reserving of the native **Assessment** element for positive analysis (strengths, opportunities, etc.) and **Driver** for the resulting positive motivations (leverage a strength, exploit an opportunity).

To distinguish adverse conditions (impacts, threats, vulnerabilities or risks), ArchiMateSE proposes new stereotyped elements, to represent the desired **specialisation** relationships in the metamodel.

The process of applying these new elements would be through a risk analysis process in the secondary architecture (Security Management Processes). It would look something like that shown in Figure 21.



**Figure 21: Business Risk Analysis Process**

The first step identifies the assets of significant value and the SABSA Attributes in which that value is enshrined. For each Attribute, a set of concrete Business Goals are identified that are necessary to enhance or protect it. Taking each Business Goal in turn, the risk analysis process performs:

- a Threat Analysis to identifies threats to the Business Goal;
- a Vulnerability Analysis to identify weaknesses in Assets that could expose its Attribute to this threat;
- an Impact analysis to assess the loss of Asset value, should the threat to the vulnerability materialise.

The risk assessment considers the potency of the threat, the exposure of the vulnerability and the value of the Asset to determine the nature and scale of the risk. An influences relation is proposed here to reflect directionality and qualitative nature of the analysis.

In the risk treatment phase, the Owner of the Business Goal must decide whether to accept, mitigate, transfer or avoid the risk, (modelled as stereotyped **Outcomes** of the Treatment process). The latter two options most likely result in some redesign of the Contextual model so the need to model them explicitly may be transient.



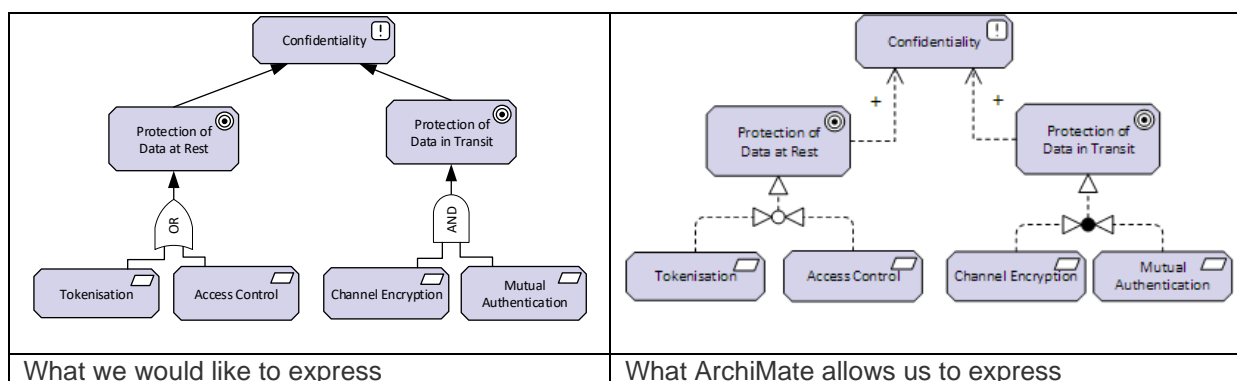
A <<Mitigate>> outcome produces a **Driver** for change. Ideally a **triggers** relationship would be available here but as the specification does not allow this relation from **Outcome** to **Driver**, **association** must suffice.

A decision to mitigate a risk results in *Control Objectives* that can be modelled structurally as sub-goals of the main Business Goal using aggregation. Use of a structural relationship allows derivation (i.e. traceability) and aggregation supports the many-to-many nature of the Business Goal – Risk Control Objective relationship<sup>18</sup>.

## 5.1.6 Control Objective and Control (Requirements)

This section discusses the relationship by which Controls<sup>19</sup> achieve a Control Objective. The statement “a *Control Objective is realised by a Control*” seems a good semantic fit but in practice, a single Control (**Requirement**) is seldom enough to model *End-to-End security* or *Defence-in-Depth*.

For the security overlay, a means of expressing more complex arrangements is needed. Consider the slightly extended example of Figure 22 (left). We wish to express that the ‘*Protection of Data at Rest*’ objective can be achieved through ‘*Access Control*’ or ‘*Tokenisation*’ and that the ‘*Protection of Data at in Transit*’ objective must be implemented with both ‘*Channel Encryption*’ and ‘*Mutual Authentication*’.



**Figure 22: Achieving Complex Requirements**

A convenient solution exists using via ArchiMate's AND and OR Junctions Figure 22 (right), which since version 3 can be used to join all relations of the same type i.e. is no longer restricted to just **trigger & flow**.

## 5.1.7 Modelling Control Profiles

The need to select a Control of an appropriate strength or combine Controls for ‘Defence-in-Depth’ or ‘End-to-End Protection’ requires support for control profiling.

As a starting point, NIST 800-53 classifies its controls into 5 main functions: Identify, Protect, Detect, Respond & Recover. To this, the paper includes the category of ‘Deter’ to cover sanctions expressed in

<sup>18</sup> Of the other available relations, *composition* would imply that a Risk Control Objective was bound to a single Business Goal. *Realisation* might be reasonable but is weaker than *aggregation* and could be interpreted as each Control Objective was alone sufficient to deliver the Business Goal, which is seldom the case.

<sup>19</sup> Strictly speaking, we identify control requirements here— not the mechanisms that actually implement these controls.



policies & contractual clauses. The *Identify* set includes Asset Management, Business Environment, Governance, Risk Assessment and Risk Management Strategy – and has been marked optional since these processes are described in the secondary & tertiary architectures and therefore rarely realised by mechanisms in the primary, run-time architecture.

The resulting properties schema is that proposed in Table 15.

Table 15: Control Profile Properties

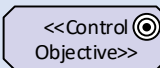
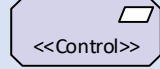
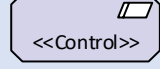
Element	Properties	Schema
 <<Control Objective>>	The Control Objective defines a protection profile using the required level for each of the five profile properties This profile can be compared with the aggregate values achieved by combining the set of controls whose individual protection capabilities are expressed using the same properties.	Control Objective / Control identify: {N/A, WEAK, MODERATE, STRONG}: optional deter: {N/A, WEAK, MODERATE, STRONG} protect: {N/A, WEAK, MODERATE, STRONG} detect: {N/A, WEAK, MODERATE, STRONG} respond: {N/A, WEAK, MODERATE, STRONG} recover: {N/A, WEAK, MODERATE, STRONG}
 <<Control>>		
 <<Control>>		

Figure 23 shows how this might work in practice. These properties are added to the Control Objective element to indicate its protection profile requirement and to each Control element to describe its protection capability.

In this example we see that the Control Objective ‘*Prevent Unauthorised Modification*’ requires an assurance level of ‘*Moderate*’ across all profile properties. There is no single Control that can satisfy this profile in full, but the set of Controls fulfil the requirement in aggregate: an Acceptable Use Policy that sanctions unauthorised use satisfies the ‘*deter*’ capability and a ‘*Regular Back-Up*’ requirement, fulfilled by a Back-Up & Recovery infrastructure service, provides the ‘*recover*’ capability.

By standardising these properties within ArchiMateSE, an analysis tool could verify that the protection profile required by the Control Objective is met or exceeded by the combination of Controls deployed to realise it.

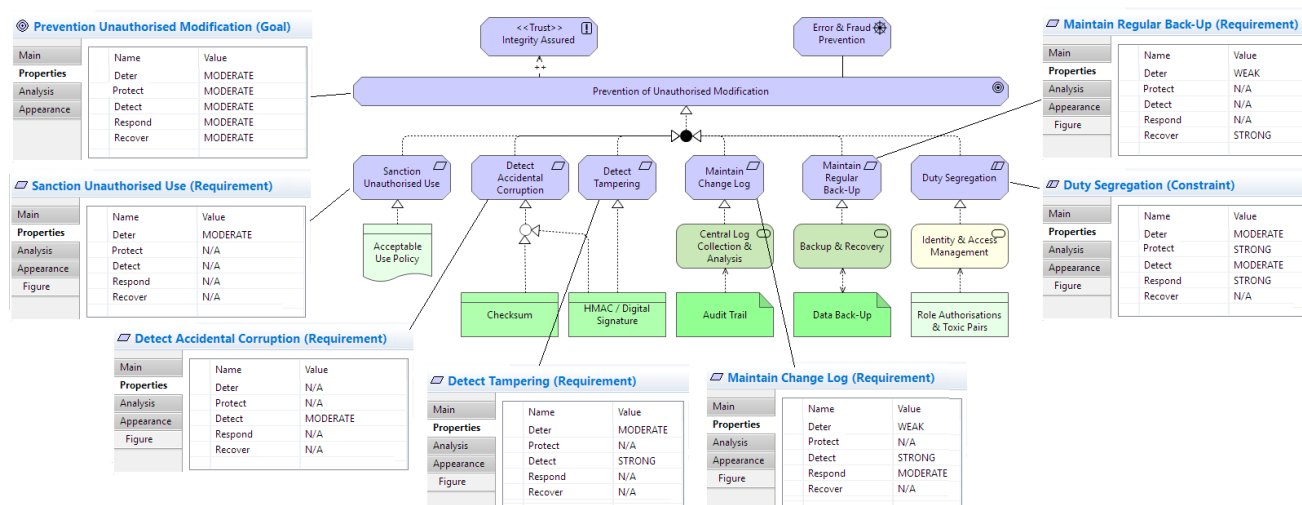


Figure 23: Example of Defence in Depth

### 5.1.8 Trust

The security overlay for the Motivation Metamodel of Figure 7 includes a Trust element as a stereotype of Principle. Discussion of this element is deferred until the section: Modelling the SABSA® Conceptual Security Architecture.

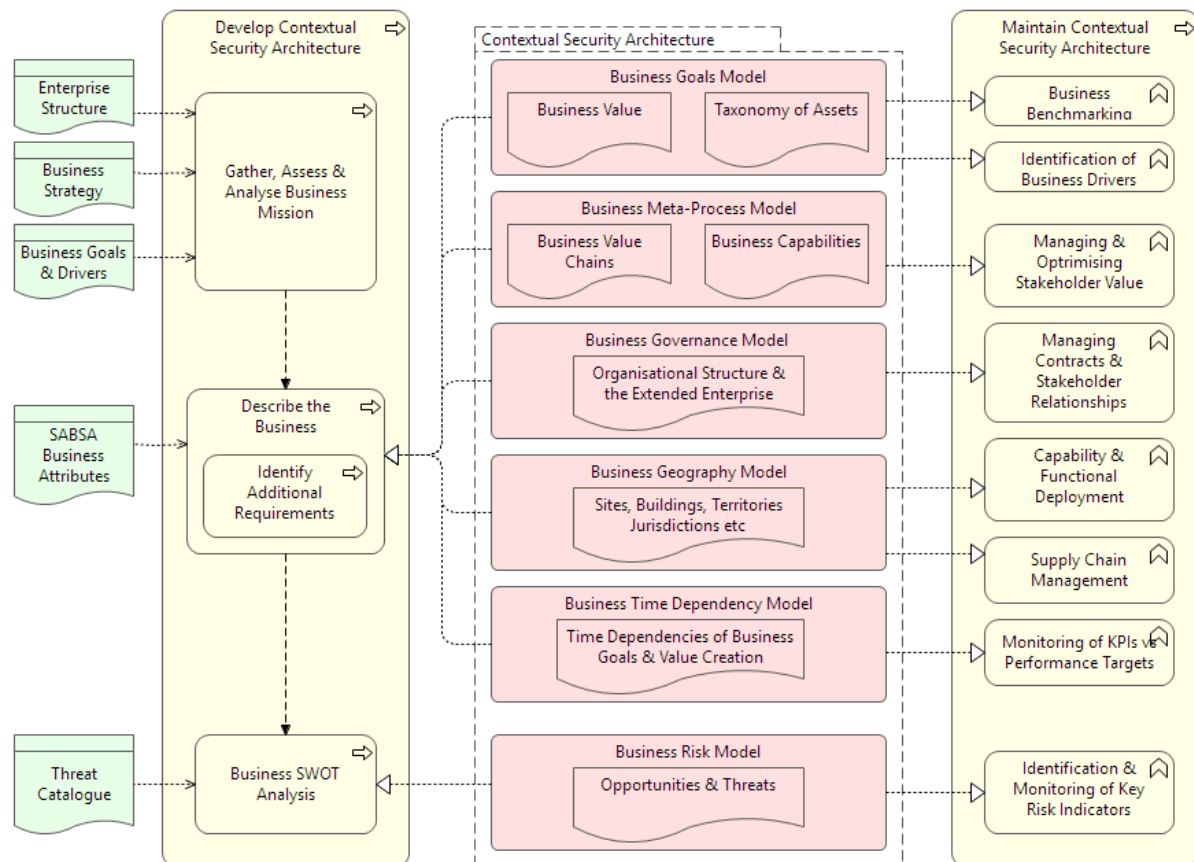
## 5.2 Modelling the SABSA® Contextual Security Architecture

The latest evolution of the SABSA framework is defined in the 2018 revision of the SABSA Matrices, [Ref. 10]. Rows corresponding to Contextual Architecture's artefacts and management processes are reproduced in Table 16 below.

Table 16: SABSA Contextual Architecture

Contextual		Assets (What)	Motivation (Why)	Process (How)	People (Who)	Location (Where)	Time (When)
		Business Goals & Decisions	Business Risk	Business Meta-Processes	Business Governance	Business Geography	Business Time Dependence
	Artefacts	Business Value; Business Asset Taxonomy	Opportunities & Threats Inventory	Business Value Chain; Business Capabilities, Services & Functions	Operational Structure & the Extended Enterprise	Inventory of Buildings, Sites, Territories, Jurisdictions etc.	Time Dependencies of Business Goals & Value Creation
		Business Driver Development	Business Risk Analysis	Capability Management	Relationship Management	Supply Chain Management	Performance Management
		Business Benchmarking; Identification of new Drivers	Analysis of Internal & External Risk Factors	Managing Processes & Capabilities for Stakeholder Value	Managing Suppliers, Service Providers, Customers & Employees	Supply & Demand Management; Deployment & Consumption	Define & Monitor Business-Driven Performance Targets

From this outline, we can begin to visualise the set of processes and activities necessary to create and maintain an architectural description (AD) of the Contextual Architecture, expressed in ArchiMate in Figure 24.



**Figure 24: Developing and Maintaining the Contextual Security Architecture**

This section will discuss how elements from ArchiMate’s Strategy & Business layers can be customised to express these Contextual artefacts as views of an ArchiMate model.

### 5.2.1 The Business Goals Model

The Business Goals Model is used to express Business Strategy. ArchiMate’s native lexicon of ‘Course of Action’, supported by Motivation elements (Analysis, Driver, Goal, Outcome, Principle, Requirement & Constraint) can be used for this purpose without additional interpretation.

Table 17 summarises the most useful ArchiMate elements in the construction of the Business Goals Model.

*Table 17: The Business Assets Mapping*


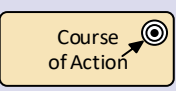

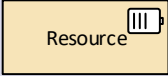
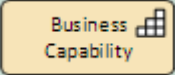
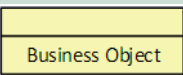
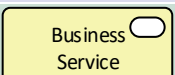
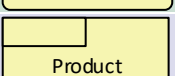

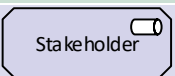
Assets (What)	SABSA Concept	ArchiMate Element	Rationale
Contextual	SABSA Business Attribute	Principle 	<b>Principle</b> is defined in ArchiMate as “an intended property of a system .... a general property that applies to any system in a certain context ... motivated by some goal or driver”. This is compatible for use in modelling SABSA Attributes.
	Business Strategy	Course of Action 	The ArchiMate Strategy layer offers <b>Course of Action</b> to represent the enterprise’s planning decisions: both strategic (long-term) and tactical (short-term).

Table 17: The Business Assets Mapping

Assets (What)	SABSA Concept	ArchiMate Element	Rationale
	Assets providing Business Value		Because ArchiMate does not offer a generic, abstract Asset element, assets will be denoted as any element to which the Enterprise attaches a Value to represent its cost, worth, utility or general importance to the Enterprise.
			
			Business Value is typically associated with the external appreciation of goods, services, information, knowledge or money, normally as part of some sort of customer-provider relationship but are supported by internal capabilities and processes at all layers that should be traced in the model.
			
			Value is most often expressed in financial terms, but non-monetary value is also essential to business e.g. practical / functional value (including the right to use a resource or service, a certification or licence to operate) and the value of information, knowledge or intellectual property.
			
			Many Business and Strategic elements have intrinsic value, such as Resource (physical assets), Capability (the ability to harness resources to produce wealth), the expected Outcome of some project or initiative, Business Information (Information Assets), Products and Services that can be bound in Contracts (commercial assets).
	Stakeholder		Stakeholder represents a role (an individual, body or organisation) interested in the realisation of Value,

In order to perform meaningful security analysis, many ArchiMate elements need to be elaborated with properties that have a bearing on the system's security posture. The ArchiMate specification supports user customisation of elements with properties but defines remarkably few as part of the standard. Achieving a consensus about the set of Element properties that are required to support security is a major goal of the ArchiMate SE initiative.

## 5.2.2 Security Extension of Business Elements

### 5.2.2.1 Business Object

A Business Object is used to represent information in the Contextual Layer. The security-related properties most commonly associated with a Business Object, shown in Table 18, might include:

- a confidentiality classification to declare sensitivity to unauthorised disclosure;
- an integrity classification to declare sensitivity to unauthorised modification;
- a non-repudiation classification as assurance of the information's authenticity / origin;
- the information's privacy sensitivity in terms of personally identifiable information (PII);
- any minimum retention period required by policy or legislation.

Table 18: Business Object Security Properties

Element	Properties	Schema
Business Object	The security-related properties associated with a Business object should follow a standardised set of key names whose values can be configured to align with local classification schemes.	<p><b>Business Object</b></p> <p>confidentiality: {PUBLIC, RESTRICTED, CONFIDENTIAL, SECRET TOP SECRET}  integrity: {STANDARD, ENHANCED, ASSURED}  non-repudiation: {STANDARD, ENHANCED, ASSURED}  pii: {N/A, PERSONAL, SENSITIVE PERSONAL}  retentionYears: Integer</p>

These might be modelled in conjunction with SABSA Attributes as shown in the example of Figure 25 which shows a Business Object (Industrial Formula) that has been given a data confidentiality classification of 'CONFIDENTIAL' and tagged with the SABSA Business Attribute <<CONFIDENTIAL>>.

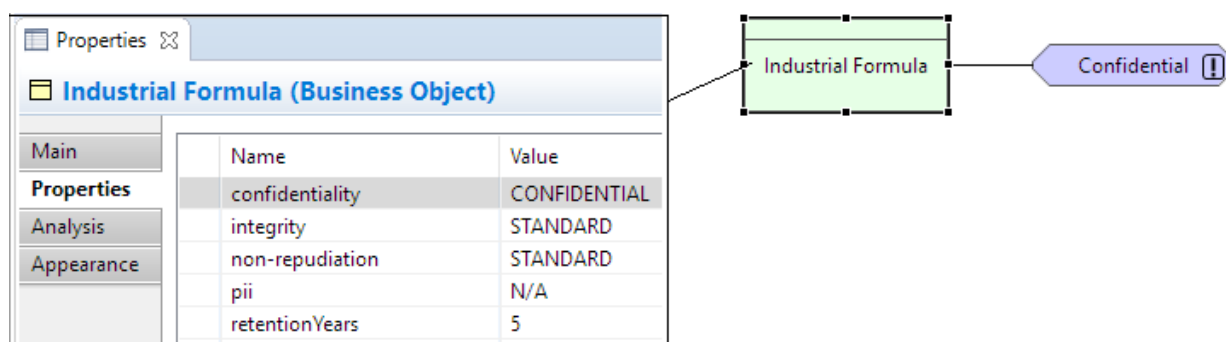


Figure 25: Element Properties vs SABSA Attributes

This pattern begs the question: 'why is the confidentiality aspect of this Information being modelled twice? Consider this scenario: classified government documents become published on Wikileaks. Are they still secret?

In one sense they are. They are still deemed highly-sensitive and their publication causes tangible damage to their rightful owner. In another sense they are clearly not, because they are irrevocably in the public domain.

Herein lies the answer: the element property is a label of intent, immutable over the information lifecycle unless reclassified. The Attribute represents a precious and fragile status that must be defended through controls.

The answer provides a useful insight into model validation. It would be curious if a Business Object classified as CONFIDENTIAL or above were not linked to the SABSA Confidential Attribute. Similarly curious if the Attribute were associated to information classified as PUBLIC. Modelling offers the possibility of SABSA Attributes being validated against such properties or patterns.

#### 5.2.2.2 Business Service, Interface and Service Level Agreement (SLA)

In terms of value analysis, the most important properties of Service elements are those committed in a Service Level Agreement<sup>20</sup> between a Service Provider and its Customers that incur penalties if not achieved.

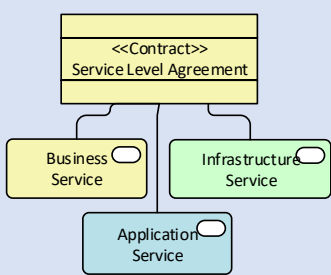
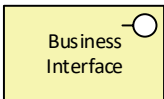
<sup>20</sup> Or to a lesser extent, an Operational Level Agreement (OLA) between business functions in the same organisation.

An SLA is a legally enforceable contract that sets out expectations of service type, quality and the resolution processes and remedies when these are not met. It usually includes a combination of technological services (applications e.g. web services, SaaS; infrastructure e.g. PaaS, IaaS, network), business (management) services (e.g. Support, Subscription, Accounting) and data/products.

The SLA element's free text description should summarise the key contracted services, responsibilities and service level expectations. These can then be modelled as SLA metrics using user-defined properties that reflect the most commonly occurring parameters (availability, recovery time and recovery point objectives, etc.). A bespoke Requirement/Constraint may also be used where a simple property is insufficient.

Business / Management Services require a distinct set of properties from technical ones because they are offered through human interfaces, but might include reaction & response times, service windows or capacity. Business Services may be offered via several Interfaces (appointment, telephone, form completion etc), each of which may exhibit different delivery characteristics: cost, capacity, partial coverage of the service hours, etc. See Table 19, below. From a security perspective, these Interfaces may also offer different capabilities for identification or authentication of parties: telephone for example being generally weaker than a physical meeting.

Table 19: Business Service Properties

Element	Properties	Schema
 <pre> graph TD     SLA["&lt;&lt;Contract&gt;&gt; Service Level Agreement"]     BS["Business Service"]     IS["Infrastructure Service"]     AS["Application Service"]     SLA --- BS     SLA --- IS     SLA --- AS         </pre>	<p>Business Services are delivered from physical interfaces by processes supported by human interaction.</p> <p>The SLA element consolidates information on all contracted services and their delivery expectations as a combination of free text and user-defined properties.</p> <p>Services reflect these same SLA properties with values showing service levels achieved.</p>	<div>Service Level Agreement</div> <div>             service hours:              reactionTime:              responseTime:              capacity:         </div> <div>Business Service</div> <div>             service hours:              reactionTime:              responseTime:              capacity:         </div>
 <pre> graph TD     BI["Business Interface"]         </pre>	<p>Business Services may be offered via several interfaces that differ in key performance parameters.</p> <p>They may also offer different capabilities to identify or authenticate the interacting parties.</p>	<div>Business Interface</div> <div>             cost:              capacity:              consumerAssurance: {LOW, MEDIUM, HIGH}              providerAssurance: {LOW, MEDIUM, HIGH}         </div>

In terms of validation, SLA metrics are strong candidates for association with SABSA Business Attributes during security analysis. Once tagged with an Attribute, model validation rules can ensure that it is supported by Control Objectives and Controls that satisfy the protection profile. It would also be possible to detect a service dependent on other services that do not meet its minimum service levels.

Interfaces exposed over a domain boundary will also merit analysis using a Trust Domain Framework.

## 5.2.2.3 Business Process / Function / Interaction

Just as Data Owners classify business information, business processes are often categorised in terms of their criticality to core business capabilities and value chains. Security analysis needs to be able to recognise such processes and protect them accordingly.

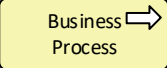
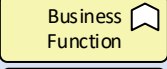
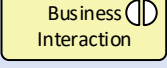
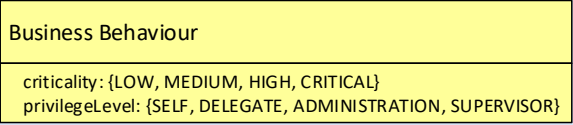
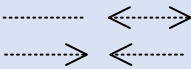
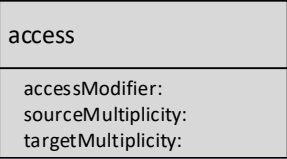
Behavioural elements may also be sensitive due to the way they operate on information. ArchiMate properties should reflect the privilege level expected to perform the behaviour element and those of the access relation to reflect the sensitivity of the information and the nature of the access: This means for behaviour elements:

- a criticality indicator that denotes the importance of the activity to the business;
- a 'privilege level' indicator that denotes behaviour requiring some degree of restriction, perhaps because it handles high value information or has potential to impact other users;

and for the access relation:

- access modifiers (read, write, create, update, delete, copy, rename);
- multiplicity i.e. whether each execution of the process operates on a single object or multiple objects.

Table 20: Business Process Properties

Element	Properties	Schema
 Business Process  Business Function  Business Interaction	The business behaviour elements that denote: <ul style="list-style-type: none"> <li>- business criticality;</li> <li>- 'privilege level' that indicates whether the process is performed on behalf of oneself, another user, as a supervisor of other users or in general administration.</li> </ul>	 <pre> classDiagram     class BusinessBehaviour {         criticality: {LOW, MEDIUM, HIGH, CRITICAL}         privilegeLevel: {SELF, DELEGATE, ADMINISTRATION, SUPERVISOR}     }         </pre>
<b>Relationship</b>		
	Suggested modifiers for access relations: <ul style="list-style-type: none"> <li>- read: scan, verify</li> <li>- write: create, append</li> <li>- read/write: update, <i>sign</i><sup>21</sup>, <i>encrypt</i>, <i>decrypt</i></li> <li>- access: copy, move, delete, rename, archive, grant/revoke access</li> </ul> Multiplicity notation as UML i.e. n (exactly n), n..m (range n to m), * (any), n..* (range n or more)	 <pre> classDiagram     class Access {         access         accessModifier         sourceMultiplicity         targetMultiplicity     }         </pre>

With these properties, the following validations become feasible:

- critical behaviour must be associated with at least one SABSA Attribute that characterises the nature and degree of the criticality;
- the privilege level of the behaviour must be sufficient for:
  - creation rights on non-repudiable information;

<sup>21</sup> Sign, encrypt & decrypt are technical accesses, not available in the business layer



- read access to confidential / personal information or modifying access to high integrity information;
- identification of sensitive processes: associated with access to high volumes of sensitive information that will require high assurance security services e.g. strong authentication.

## 5.2.2.4 Business Roles & Actors

In ArchiMate, Actors represent human or organisational entities that can be assigned to Roles that describe:

- the extent of their responsibilities with respect to a given business process;
- their consumption of Business Services.
- Security analysis should be able to determine the following relevant information from the model:
- whether an Actor is human or organisational (e.g. a corporate legal entity, department or team)<sup>22</sup>;
- if human, whether they are a data privacy subject, and if so, possibly a minor?
- whether the Role is considered privileged?
- whether the Role is considered part of a 'Toxic Pair' requiring segregation of duties?
- the expected number of Actors (individuals/entities) assigned to a Role (many to one);
- the expected peak number of parallel, in-flight service calls (peak demand).

Table 21: Actor & Role Properties

Element	Properties	Schema					
<p>The diagram shows a Data Subject (stick figure) associated with a Business Actor (stick figure). A Business Process (rounded rectangle) is associated with a Business Role (rounded rectangle). A Business Role is associated with a Business Service (rounded rectangle). A Business Actor is associated with a Business Role. A Business Process is associated with a Business Role.</p>	<p>An Actor element should indicate:</p> <ul style="list-style-type: none"><li>- its population size;</li><li>- its type (human, organisational or technical).</li></ul> <p>A human Actor may be specialised to Data Subject if their personal data is used by business services. A property flags the possible participation of minors.</p> <p>A Business Role should indicate</p> <ul style="list-style-type: none"><li>- 'privilege level' using the same categories defined for behaviour;</li><li>- Participation in any Toxic Groups.</li></ul>	<div><div>&lt;&lt;Data Subject&gt;&gt;</div><div>couldBeMinor: Boolean</div><div>Business Actor</div><div>population: Integer Range type: {HUMAN, ORGANISATIONAL}</div><div>Business Role</div><div>privilegeLevel = SELF: {SELF, DELEGATE, ADMINISTRATION, SUPERVISOR}</div></div>					
Relationship	<p>The diagram shows two types of relationships: an assignment relationship (solid line with a filled circle at the source) and a serving relationship (solid line with an open circle at the target). Both relationships support UML multiplicity notation.</p>	<p>Assignment and serving relationships should support the UML multiplicity notation i.e.</p> <p>n (exactly n), n..m (range n to m), * (any), n..* (range n or more)</p>	<table><tr><th>assign</th><th>serving</th></tr><tr><td>sourceMultiplicity:</td><td>targetMultiplicity:</td></tr></table>	assign	serving	sourceMultiplicity:	targetMultiplicity:
assign	serving						
sourceMultiplicity:	targetMultiplicity:						

These properties support the following validations:

- Data Subjects, which can only be specialised from human Actors, must be associated with at least one element of Business Information with the PII property set;

<sup>22</sup> Note that the ArchiMate specification does envisage a technical entity being modelled as an Actor. On one hand, it would not be appropriate for technical entities to appear in the business layer but on the other, they are Principals that have identities, need to authenticate, are granted access rights to consume services etc More on this in Logical Access Management.



- with respect to Toxic Groups that are segregated by Role<sup>23</sup>):
  - an Actor being assigned to multiple Roles belonging to the same Toxic Group;
  - a technical account participating in Role requiring segregation of duties;
- that the aggregate number of service requests remains within the service capacity;
- identification of high security risk roles: associated with large constituencies of Actors having execute critical processes or access sensitive services.

### 5.3 Modelling the SABSA® Conceptual Security Architecture

The Conceptual Architecture provides the security architect's view of the System of Interest (Sol). It is concerned with the enhancement / preservation of the value chain, the protection of assets & the assurance of processes that support key capabilities, the identification & definition of common security services and general management of the risks associated with the pursuit of opportunity.

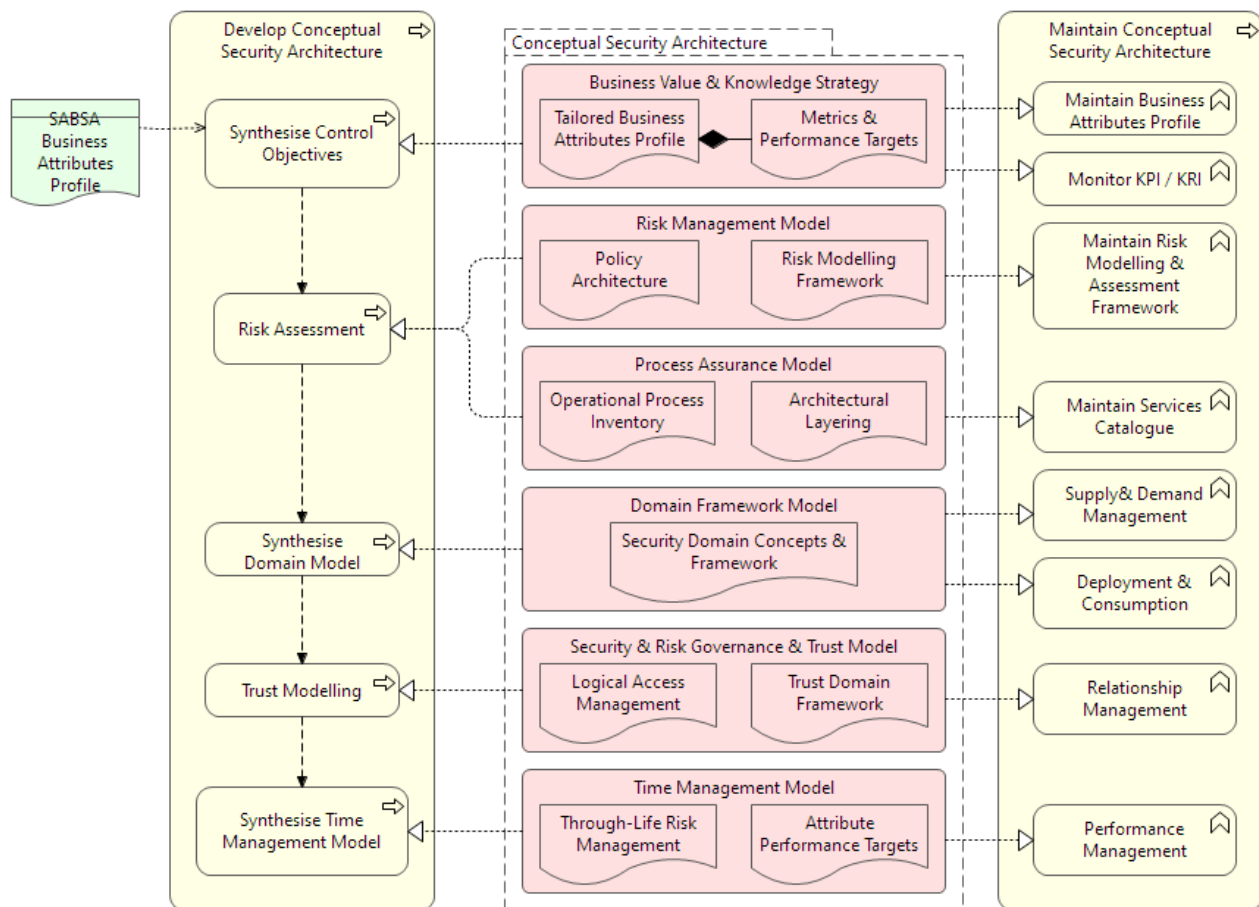
The rows corresponding to the Conceptual Architecture's artefacts and management processes are defined in the 2018 revision of the SABSA Matrices [Ref. 10], reproduced in Table 22.

Table 22: SABSA Conceptual Architecture

		Assets (What)	Motivation (Why)	Process (How)	People (Who)	Location (Where)	Time (When)
Conceptual	Artefacts	Business Value & Knowledge Strategy	Risk Management, Strategy & Objectives	Strategies for Process Assurance	Security & Risk Governance; Trust Framework	Domain Framework	Time Management Framework
		Business Attributes Taxonomy & Profile	Enablement & Control Objectives; Policy Architecture; Risk Categories; Risk Management	Operational Process Inventory; Process Mapping Framework	Owners Custodians & Users; Trust Modelling Framework	Security Domain Concepts & Framework	Through-Life Risk Mgt Framework; Attribute Perf. Targets
	Activities	Proxy Asset Development	Developing Risk Objectives	Delivery Planning	Role Management	Business Portfolio Management	Service Level Definition
		Defining Business Attributes Profile: Performance Criteria, KPIs & KRIs	Maintaining the Risk Modelling Framework; Risk Analysis on Business Attributes Profile	SLA Planning; BCP; Financial Planning; Transition Planning; Services Catalogue	Maintain Trust Modelling Framework; Liabilities; Define Roles & Responsibilities	Business Footprint; Points of Supply & Access	Manage Performance Criteria & Targets; Abstracting Attribute Performance Targets

From this outline, we can begin to visualise the set of processes and activities necessary to create and maintain an Architectural Description (AD) of the Conceptual Architecture, expressed in ArchiMate in Figure 26.

<sup>23</sup> Rather than segregated on the basis of distinct individuals who may share a common organisational or business role.



**Figure 26: Developing the SABSA Conceptual Security Architecture**

Although this layer has no ArchiMate equivalent, many of its concepts (attribute profiles, metrics, risk analysis, control objectives & layering) have already been discussed in the context of: The Security Overlay of the Motivation Metamodel.

This section examines how the remaining security concepts and artefacts can be modelled. It will also consider how the Conceptual layer can be inserted between the Contextual and Logical layers.

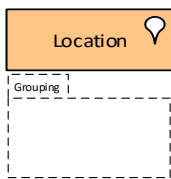
## 5.3.1 Domain Framework Model

The Domain Framework identifies the security domains of the System of Interest and is an essential basis for the creation of a Trust Model: the analysis of security concerns that arise from signals (service calls, system accesses, event triggers and data flows) that are enacted across domain boundaries.

A Security Domain can be synonymous with any bounded space (physical, jurisdictional, organisational, logical, network zone) created over any layers in the model. The ArchiMate SE overlay adds a `securityDomain` property to composite elements (**Grouping** and **Location**), set `FALSE` by default, as a means of denoting security domains.

The relevant ArchiMate elements for domain modelling are shown in Table 23. ArchiMate's syntax allows elements to be placed into containers using *composition*, *aggregation* or *assignment*. Composition is generally too strong due to its implied exclusivity: in practice, elements are often subject to several domains simultaneously. *Aggregation* is usually the best choice for grouping domains, *assignment* for Location.

Table 23: Security Domain Mapping

Element	Properties	Schema				
	Any Location or Grouping element can be marked as a Security Domain by declaring an optional securityDomain property and setting its value to TRUE.	<table><tr><td>Location</td></tr><tr><td>securityDomain = FALSE : Boolean</td></tr><tr><td>Grouping</td></tr><tr><td>securityDomain = FALSE : Boolean</td></tr></table>	Location	securityDomain = FALSE : Boolean	Grouping	securityDomain = FALSE : Boolean
Location						
securityDomain = FALSE : Boolean						
Grouping						
securityDomain = FALSE : Boolean						

By assigning elements to security domains, analysis can determine whether a signal traverses a domain boundary: a condition that prompts the need for Trust Modelling.

## 5.3.2 Risk Management Model

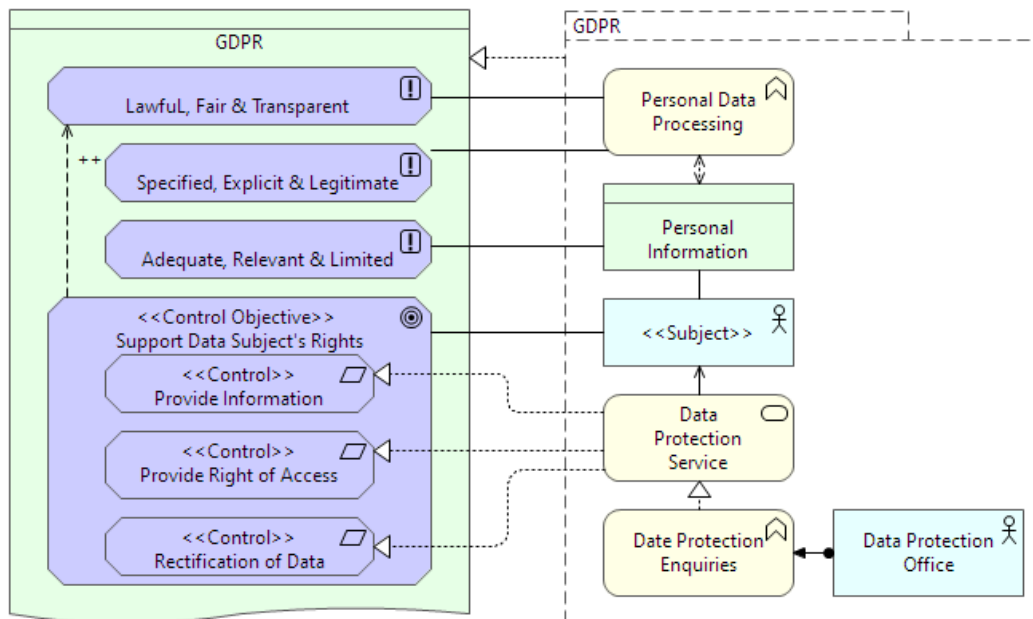
Many of the topics in this domain (risk modelling, attribute profiling, control objectives) have already been covered in the context of the Motivation Metamodel. What remains is a discussion of the Policy Architecture.

### 5.3.2.1 Policy Architecture

The Policy Architecture is a framework of codified statements of regulatory, compliance controls or those documented in the organisation's own policies and standards. There are significant differences between controls mandated by policy and those derived from risk:

- Policies are predicated on statements of applicability that define the policy's scope and the conditions under which its controls apply. If the System of Interest is within scope and meets the criteria, the control statements must be applied, regardless of whether the organisation is willing to accept the risk or not.
- Emphasis on the traceability / audit of legal, compliance or certification requirements to controls;
- Policy statements are not only numerous, they are defined at high level and tend to be generally applicable. This typically generates a very challenging quantity of controls that all need to be traced. Consequently, a main aim of ESA should be to consolidate these controls into shared security services.

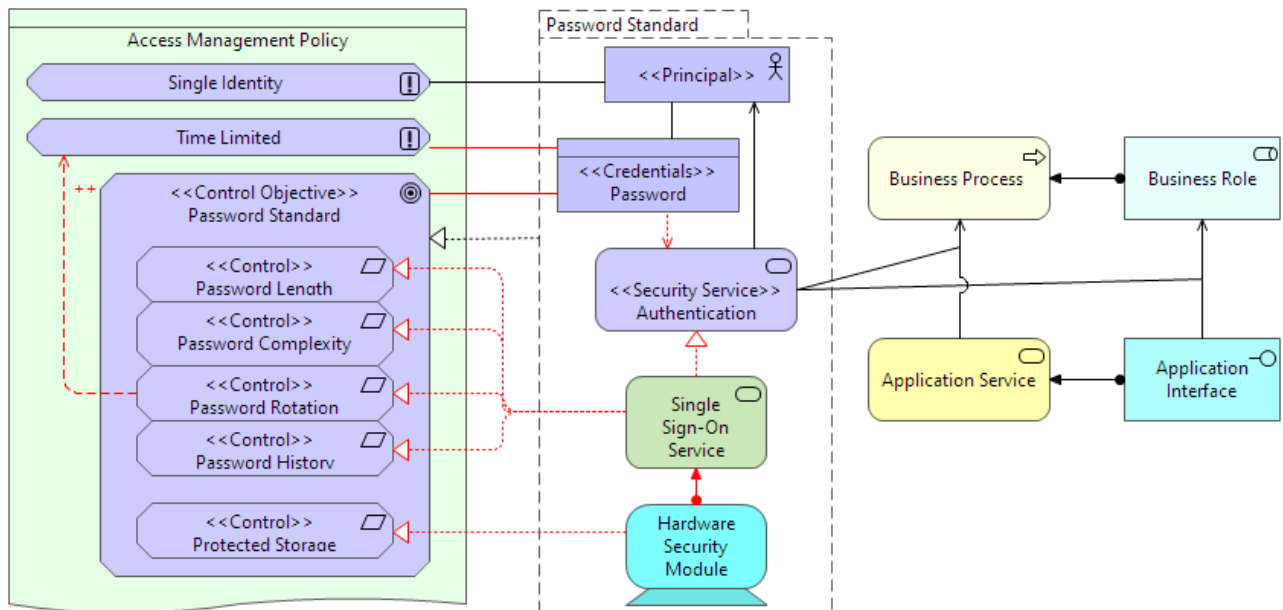
Policies are most easily modelled in ArchiMateSE as a Representation containing a collection of Principles, Control Objectives and Requirements. The policy's scope can be modelled a security domain Grouping as described earlier. A fragment depicting how the EU's General Data Protection Regulation might be modelled is shown in Figure 27.



**Figure 27: Modelling Policy Frameworks**

As with the earlier discussion of Control Objective and Control (Requirements), Control Objectives are applied to all elements that are in scope, realised by Control Requirements that should be associated with some part of the model and realised by an appropriate Role, Service, Process, Application or Technology etc.

Given their sheer quantity, the ability to trace policy statements through to the realisation of controls is one of the most valuable capabilities of model validation. The pattern that validation must look for is illustrated by the closed loop of relationships, highlighted in *red*, in Figure 28.



### Figure 28: Policy Validation

By analysing this model fragment, a number of statements can be made about the compliance of the Application Service to the Access Management Policy:

- The Policy's principle of time-limited access is supported by the Password Rotation requirement set out in the Password Standard;
- Both the Time-Limited Principle and the Password Standard apply to the Password issued to the Principal for authentication to the application;
- The Password Standard consists of at least five Control Requirements:
  - four of these are realised by a Technology Building Block (a Single Sign-on (SSO) Service);
  - the fifth by a Technology Component (a Hardware Security Module (HSM)).
- There is a closed loop<sup>24</sup> from each Control Requirement to the Password, which is realised by a compliant SSO Service to which a compliant HSM has been assigned.
- Similarly, there is a closed loop encompassing the applicability of the Time-Limited Principle to the Password and its adoption by the SSO Service;
- Authentication to the Application Service is therefore compliant with the Password Standard<sup>25</sup>;
- In this model fragment at least, there is no similar closed loop to confirm that the principle of Single Identity (i.e. one record per person in the User Registry) has been implemented.

<sup>24</sup> Note that realisation relationships between the Control Objective and each Control are defined in the underlying model. They can be implied in the diagram by the use of nesting despite being suppressed visually.

<sup>25</sup> A redundant link to the serving relation between the Active Structure elements, Role & Interface, is also modelled.

5.3.3 Security & Risk Governance Model

This model includes the Trust Domain Framework and Logical Access Management

5.3.3.1 Trust Domain Framework

Trust is essential to business transactions yet is rarely made explicit in conventional Architectural Descriptions. Trust causes business partners to accept risks that would otherwise be declined, based on intangibles such as reputation, history or the expectation of a long-term relationship. It is essentially a human quality that, though it cannot be reproduced outside the context of the business relationship, must be protected as soon as any part of that transaction is delegated to third party entity (e.g. an agent, a process or an IT system).

The analysis of signals crossing a domain boundary aims to identify any implicit trust in the context of the interaction. Trust must be represented explicitly in the model in order to derive the necessary protection requirements. Examples of inter-domain signals are shown in Figure 29.

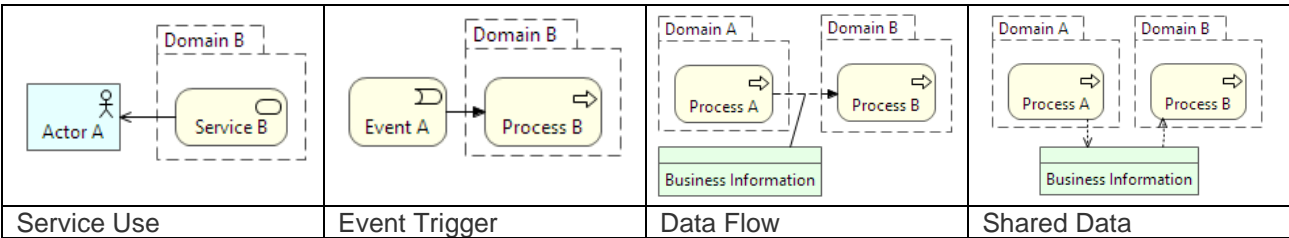


Figure 29: Common examples of Signals crossing Domain Boundaries.

While active structure & behaviour elements should be placed in the domain where an activity takes place (both visually and via structural relationships), the modelling guidance for passive structure<sup>26</sup> elements is more nuanced. Information is generally copied and distributed in the course of processing and comes to exist in multiple domains.

A modeller may opt to assign an Object to all domains in which it is used or “free” it by not assigning it at all. The choice will often be determined by the sensitivity of the information: objects that are subject to regulation for example, should be assigned to a security domain if access from outside that domain represents a concern.

Another characteristic of a valid trust model is that, while any oural element may be a “trusted” entity, only active structure (or behaviour assigned to active structure) can be a ‘trusting’ entity. In other words, information may be the object of a trust relationship (be it trusted or untrusted) but is incapable of doing any trusting.

<sup>26</sup> except Artefacts and Materials (in Technical & Physical layers) which represent physical objects.

To model trust in ArchiMate, the use of stereotyped flow relationships to indicate that “A trusts B” seems intuitive<sup>27</sup>. It is also able to convey a direction of trust that is independent of that of the underlying signal or show opposing flows to indicate bi-directional trust.

As with SABSA Business Attributes, the nature of the trust is elaborated using a stereotype of the Principle element that, like Attributes, should only be *influenced* in the model through the realisation of control objectives.

A simple example of trust concepts expressed using ArchiMate in this way is shown in Figure 30.

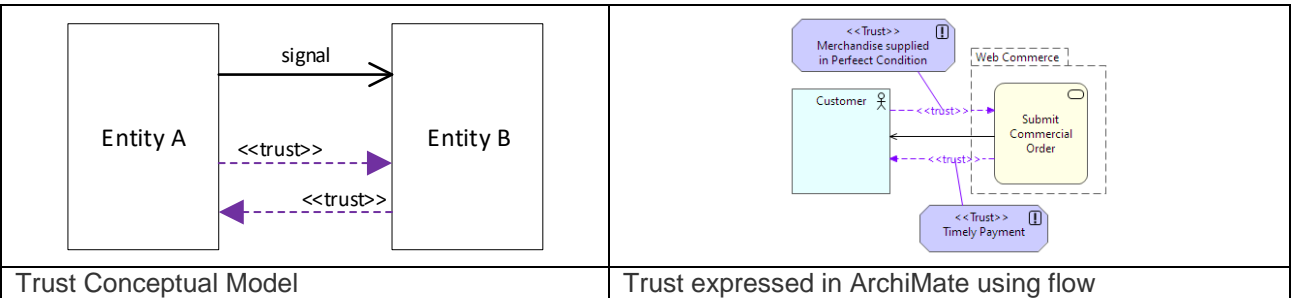


Figure 30: Simple Trust Relationships using Flow

This approach offers simplicity but, despite appearing to be a good first attempt, it has a couple of issues:

Firstly, there is no linkage between the trust relationships and the underlying signal. Whenever more than one signal is drawn between a pair of entities, analysis cannot contextualise which trusts belong with which signal<sup>28</sup>.

Second, ArchiMate imposes a language restriction that prevents connection of flow relations to passive structure. Trust in an Object as in “*I trust my bank but not every email purporting to come from my bank*” cannot be expressed using this approach.

To overcome these limitations, the overlay offers an alternative representation using the stereotyped Trust alone. Figure 31 illustrates how this is used to apply trust expressions to the patterns introduced in Figure 29.

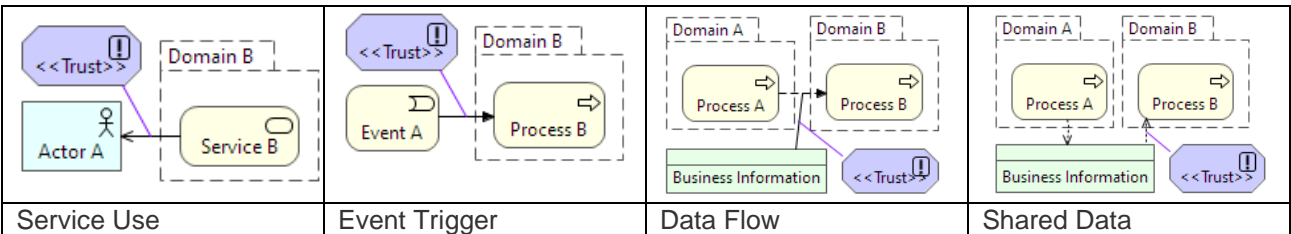




Figure 31: Trust Attributes associated with inter-Domain Signals.

<sup>27</sup> While deliberately avoiding the question of what actually flows from one entity to the other.

<sup>28</sup> Although the association can be shown visually via selective diagrams ArchiMateSE aims to enable automated analysis.

This alternative representation lacks a visual representation of the direction of trust that can be partly offset by good naming of the Trust element and elaborated by its textual description. For automated analysis, the overlay provides a `directionality` property that sets the direction of the trust relative to the associated

Table 24: Elements & Relationships used in Trust Modelling

Element	Properties	Schema
	Trust is modelled as a stereotype of the Principle element that can be associated to any relationship to make explicit any trust underpinning the transaction. A directionality property, defaulted to FORWARD, declares the direction of the trust.	<div>Trust</div> <div>directionality = FORWARD : {FORWARD, REVERSE, BIDIRECTIONAL}</div>
<b>Relationship</b>		
 <p>A trusts B</p>	Simple trust between entities that arise from the general context rather than a specific transaction, are modelled by a stereotyped data flow: the arrow indicating the direction of trust and its nature elaborated using a Trust element.	

The use of Security Domains and Trust Models support the following validations:

- Interactions that cross security domain boundaries are identifiable as prime candidates for trust analysis. Traditional architectural descriptions, focussed primarily on technical aspects, overlook the significance that trust plays in the business's risk appetite. Making this trust explicit in the model is a critical success factor in establishing a complete set of system protection requirements.
- A model declaring a serving relationship that crosses a trust boundary should also have a data or control flow (trigger) defined in order to be considered complete. This rule will identify any sensitive data (confidentiality, integrity, authenticity etc.) crossing the boundary in the context of that service.
- Many patterns, such as access to sensitive or private data from outside the domain, can be matched as a policy violation if the Trust element is not protected through Control Objectives realised by Controls.

## 5.3.3.2 Logical Access Management

Logical Access Management (LAM) maps the contextual entities (Actors & Business Roles) that were identified in the Business Governance Model onto Conceptual entities: Principals issued with Credentials and Authorisations that will eventually translate into the creation and provisioning of Accounts. This requires security services such as identification, enrolment, provisioning, authentication and authorisation. The model also examines how these entities interact in trust relationships across security domain boundaries as modelled in the previous section.

As is to be expected in this layer, ArchiMate has no native elements to represent these security concepts so the overlay must overload core elements from the Business layer. The relationship between these elements and the business-layer elements from which they are stereotyped is shown in Figure 32 (left).



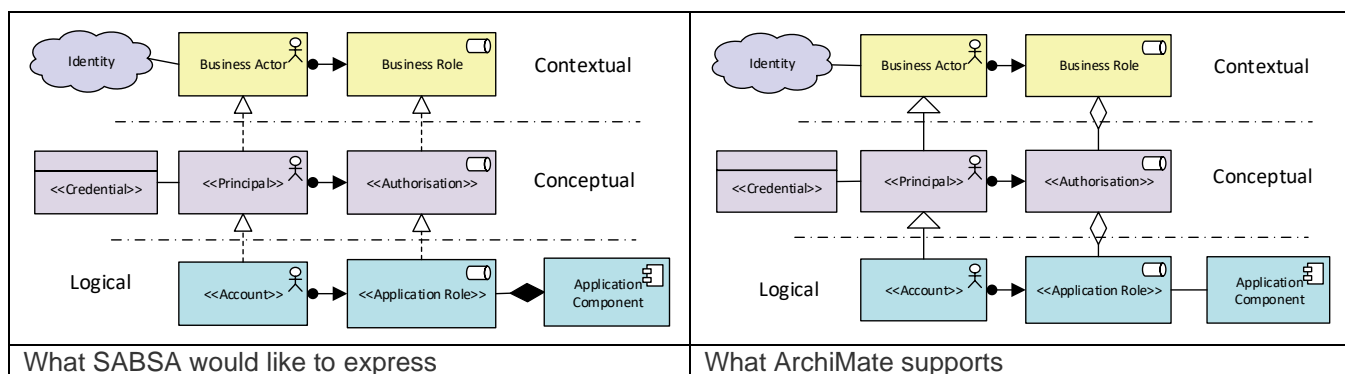


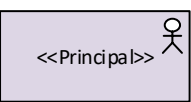
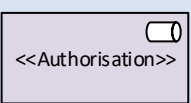
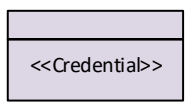
Figure 32: Extended Entity Metamodel

Ideally it would be possible to follow the precedent of the ArchiMate specification by using the **realisation** relation to derive structural entities from upper to lower layers. This would have allowed a coherent realisation structure from a Business Actor (*Contextual*), via a Principal (*Conceptual*) to an Account (*Logical*) and from a Business Role (*Contextual*), via Authorisation (*Conceptual*) to an Application Role (*Logical*) to express individual access rights.

It would also have been possible to show Application Role as a composite element of Application Component. Unfortunately, this intuitive approach extends ArchiMate in ways for which it was not designed: neither **realisation** between elements of the same type nor Role as **composition** from Application are legal syntax. The closest approximation supported by the 3.0 specification is shown in Figure 32 (right) with *specialisation* used to derive Actors, *aggregation* to derive Roles and *association* to show roles defined at application level.

Table 25 presents a summary of overloaded elements.

Table 25: Elements used in Logical Access Management

Element	Properties	Schema
	Principal is the conceptual representation of a real-world individual or entity, including technical entities. Principals are created through an identification process with a given level of assurance.	<div>Principal</div> <div>assurance: {LOW, MEDIUM, HIGH}</div> <div>type: {HUMAN, ORGANISATIONAL, TECHNICAL}</div>
	Authorisation is conceptual representation of actions that a Principal is authorised to perform. As it decouples Business Roles from Application Roles, is used to model central access management systems e.g. Active Directory Groups.	<div>Authorisation</div> <div>owner: Manager</div> <div>recerificationPerid:TimePeriod</div> <div>lastRecertified: Date</div>
	The conceptual representation of a proof of identity that enables a Principal to authenticate to a given application or authentication service. The assurance property denotes the associated authentication strength.	<div>Credential</div> <div>assurance: {LOW, MEDIUM, HIGH}</div>

Principal is the conceptual representation of a real-world individual or entity in the IT system architecture. Unlike Actor, the business layer entity from which it is stereotyped, a Principal can also represent technical

components (e.g. a robot, workstation or server) that needs to authenticate in order to provide or consume services. As well as expanding the type options to include machine actors, the security properties of Principal should indicate the assurance level of the identification process through which this type of Principal is enrolled.

Authorisation is the conceptual representation of permissions that can be granted to a Principal in order to perform actions. The conceptual layer decouples Business Roles from Application Roles, so this element is typically used to model centralised access management systems e.g. Active Directory Groups. The security properties of this group should support the recertification process, identifying the owner, the recertification period and the last recertification performed.

A Credential is the conceptual security representation of a proof of identity that enables a Principal to authenticate to a given application or authentication service. It has been created as a stereotype of Business Object and bears an assurance property that denotes its nominal authentication strength.

Linking these elements to the logical layer, Account is used to represent an authenticated Principal on the target system from which, access to a restricted set of resources (e.g. the individual's own email) can be controlled. Application Role represents roles that are defined in the application code to support for example, role-based access control (RBAC) mechanisms. These elements will be discussed further in the next section.

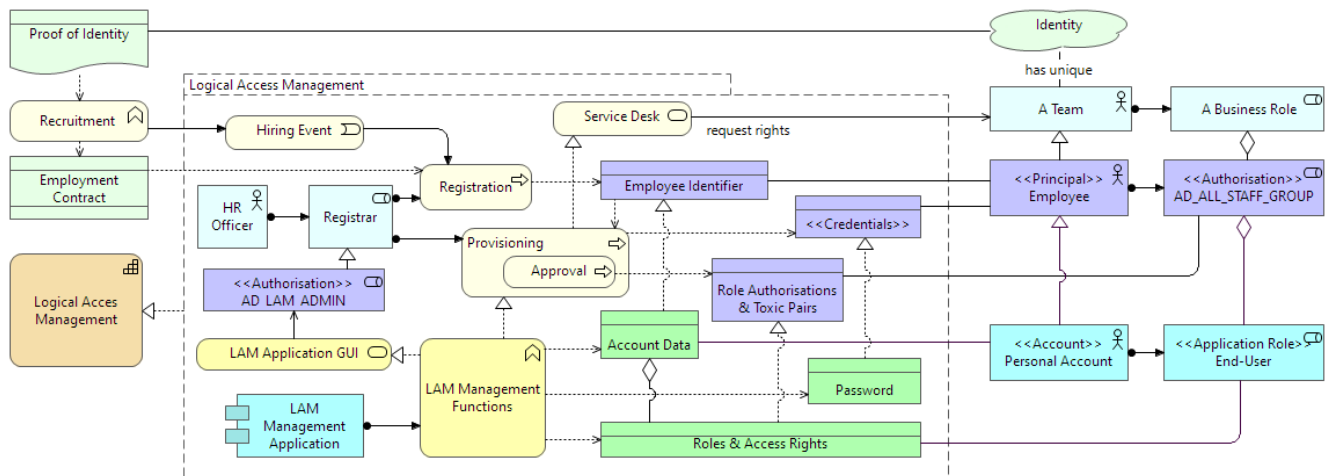
By modelling this structure, the following examples of security validation become possible:

- Consistency in on-boarding: that a single individual cannot enrol in the system under multiple Principals, thereby potentially by-passing segregation of duty controls;
- That human principals are not assigned machine accounts or can access interfaces and services intended for machine accounts & vice-versa.
- Highlighting areas of risk caused by sensitive functions being accessed by an Active Directory group that has no identified Owner or has not been recertified recently.
- Every instance of a Credential appearing outside the LAM domain (or the domain of its creation), should trigger requirements for secure distribution, storage, access control etc.

### 5.3.3.2.1 Identification, Authentication and Provisioning

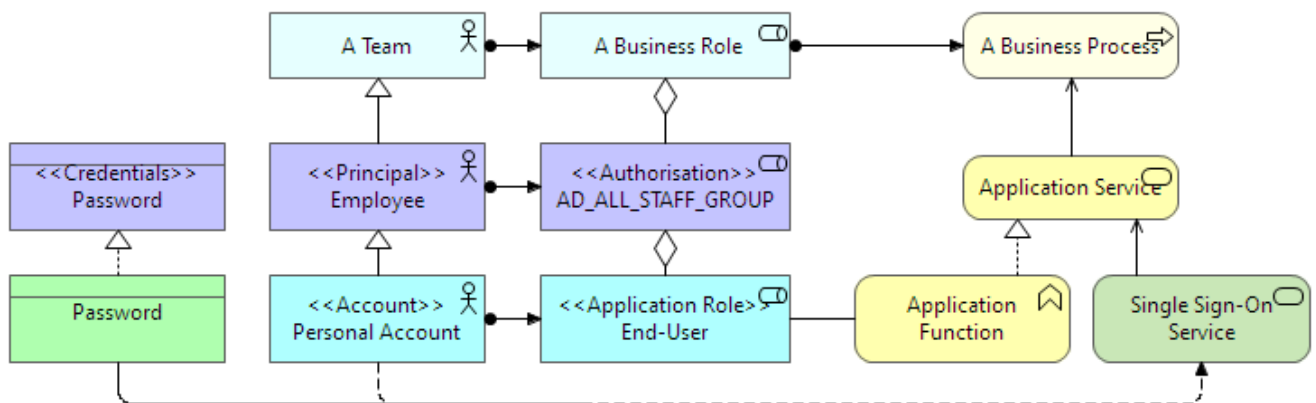
Security services are discussed in a more general sense in the section: '*Process Assurance Model*' but because of their high relevance in the establishment of trusted entities, LAM services: Registration (identification & account enablement), Authentication and Provisioning are described in depth by the models of this section.

The foremost reference on practical ArchiMate [Ref. 7] advocates the structuring of models in 3 architectural planes: a primary architecture (describing run-time use), a secondary architecture (system development, administration & operations) and a tertiary architecture (ownership and governance). Authentication is run-time process in primary architecture but Registration and Authorisation are best modelled as part of a LAM capability in the secondary architecture as shown in Figure 33.



**Figure 33: Example of Registration & Provisioning in the Secondary Architecture**

In the example of Figure 34, a Service Provider initiated Single Sign-on Authentication has been modelled as an Infrastructure service because that is how it looks from the Application's perspective: the paradigm of the externally-visible service redirecting an unauthenticated session to an authentication service.

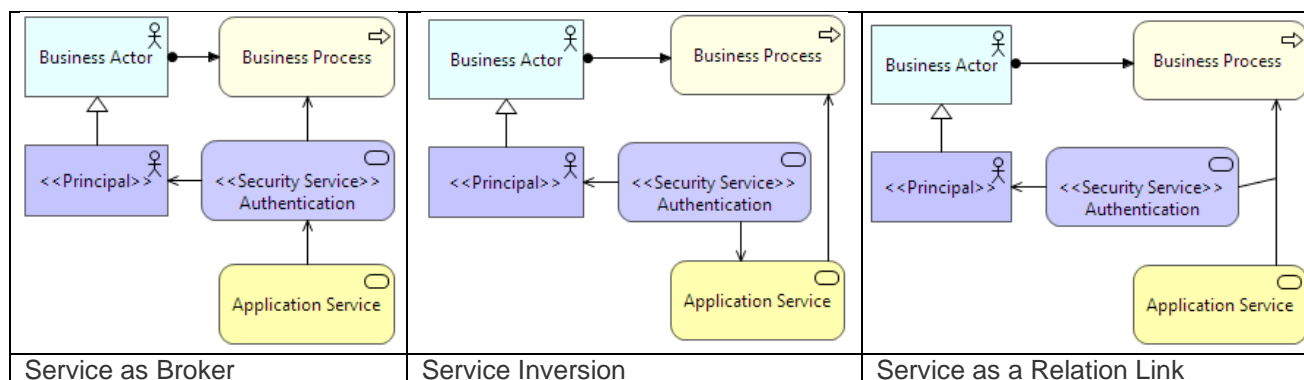


**Figure 34: Modelling Authentication in the Primary Architecture**

There are also good arguments for modelling Security Services in the Conceptual layer:

- to create a Security Services Catalogue that does not depend on specific technical implementations;
- to visually emphasise that it acts as a gatekeeper between Service Consumer and Service Provider.

This requirement raises an issue about the use of the serving relations. Figure 35 shows possible options.



**Figure 35: Security Services in the Conceptual Layer**

In Figure 35 (left), a conceptual Authentication Service is placed as an in-line serving relation between a business process and an application. Visually, it represents well the concept that the service mediates access, and for local authentication, the pattern works well. However, if the service is shared (e.g. Single Sign-on), the element becomes a hub in the model with spokes serving multiple business processes and served by multiple applications. In doing so, important links that denote which processes depend on which applications, are lost.

Figure 35 (centre) restores the direct link between business process and application service. It also shows that the application calls the authentication service, but it is no longer clear that authentication acts as a gatekeeper. This pattern also breaks an architectural design principle that services should serve the same or the next higher layer but not a lower one.

Figure 35 (right) provides a pattern that resolves all these issues: the direct link between business process and application is maintained, the security service is visually represented as guarding the service call and there is no inversion of the directionality of service invocation between layers.

## 5.4 Modelling the SABSA® Logical Security Architecture

The Logical Architecture provides the security perspective of the designer's view of the System of Interest. This section proposes security-related properties for ArchiMate's logical layer entities that enable security analysis.

### 5.4.1 Application Service

The primary security issue for an Application Service is the ability to enforce access control. This in turn is predicated on the Service's ability to establish a minimum level of assurance about the digital identity of the user. On the premise that on the Internet, "nobody knows you're a dog"<sup>29</sup>, the overlay needs to be able to express that while "*being a dog*" might not be a concern for some low-risk services, others will demand a level of confidence that the Principal accessing the service is a legitimate proxy for the real-world entity.

<sup>29</sup> Peter Steiner: The New Yorker, July 5, 1993

To model this, ArchiMateSE proposes properties that reflect the 3-level assurance criteria identified in NIST's Digital Identity Guidelines [Ref. 8]:

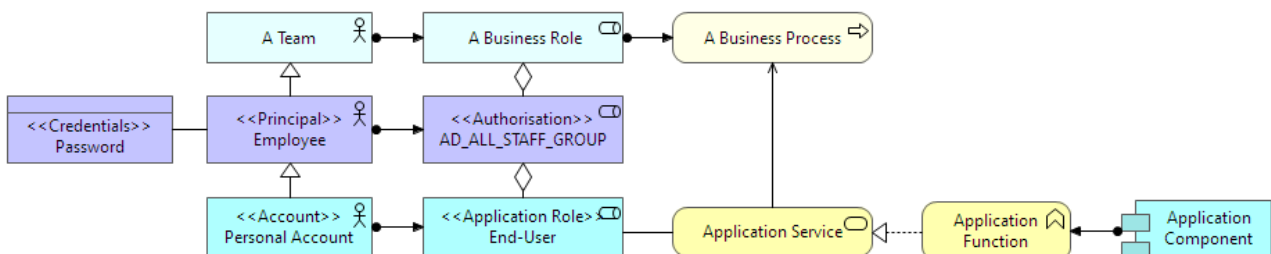
- `identityProof`: enrolment & credential issuance processes bind the applicant to a real identity;
- `authenticationStrength`: the assurance strength of the authentication mechanism itself;
- `authenticatorTrust`: assurance in way the credential is presented, considering aspects such as bearer-type, one-time use, freshness, challenge/response etc.

Each property is proposed as a 3-tier classification, `STANDARD`, `ENHANCED`, `ASSURED`, that maps easily to the assurance levels described in NIST but this could be adapted to other schemes such as ISO/IEC 29115's 4-level scheme or a proprietary scheme (e.g. numeric range), defined by the implementing organisation.

Access control is increasingly implemented via external security services (policy administration, information and decision points), but enforcement is predominantly local and based on the Principal having an authorised role (among other claims).

In the model, Application Roles are linked to the Application behaviour elements that they are defined to perform via *association* relationships. Credentials generally materialise in the logical level as some form of data (e.g. a passcode, dactylogram or retina scan) that must be compared against the credential on record. These may or may not be associated with the target system account, depending whether authentication is performed locally or via a token from an Authentication Service.

An illustrative example is shown below in Figure 36.



**Figure 36: Example Logical Access Model**

Application Service availability is also usually a concern. In the **Service Level Agreement** section, it was noted that applications exposed to external entities (self-service web sites, SaaS applications, web services etc.) can incur penalties if committed service levels are not achieved. Similar commitments may have been made to internal consumers via an Operation Level Agreement (OLA).

As with Business Services, committed Application Service levels should be modelled as user-defined properties reflecting the most commonly occurring parameters (availability, recovery time and recovery point objectives etc.) or a bespoke Requirement/Constraint where a simple property is insufficient.

Whereas an SLA property defines the committed level, the corresponding Service property should declare the level expected (or achieved). This may be self-evident for top-level services, but the value of this

approach is the ability to detect where an Application Service subject to a high service expectation depends on a sub-component only capable of a lesser one.

### 5.4.2 Account

Account was introduced in the previous section as the logical layer representation of a Principal from the perspective of a single application or target system. A Principal may hold multiple Accounts on multiple systems.

The principle security properties of an account are:

- `type`: indicates whether the account is intended for use by human individuals (`NOMINAL`), shared among a team (`FUNCTIONAL`) or machines (`TECHNICAL`);
- `domain`: identifies the policy domain that defines account namespaces, naming conventions, the assignment or right to claim a particular account name etc.

### 5.4.3 Application Role

Application Role was also introduced earlier and represents the roles (and their associated a set of access rights) defined in the application code or configurable on the target system. In the previous section (Figure 33 & Figure 34), the Logical Access Management model shows how Accounts can be provisioned with Application Roles using the standard *assignment* relation.

Ideally, it would also be possible to use the *assignment* relation to bind an Application Role to an Application Service or Function, but since it has had to be stereotyped from the Business Actor element, the ArchiMate syntax does not allow it. As Figure 36 shows, *association* is used instead.

### 5.4.4 Application Interface

Application Interfaces bind Application Services to an endpoint or channel through which they may be consumed. Interfaces can be categorised into two types: human (e.g. command line, GUI; voice-operated) and machine interfaces (e.g. remote procedure calls, message-driven), each of which are likely to be subject to different constraints or policies.

For example, an organisation may decide that all machine-to-machine authentication must use certificates rather than passwords or that machine credentials should not be valid when proffered to human interfaces.

ArchiMateSE proposes a `type` property to declare whether an interface is intended for human or machine use.

Each interface via which a service is exposed must satisfy the minimum `authenticationStrength` and `authenticatorTrust` requirements specified by the Application Service using a type-appropriate mechanism.

An `authentication` property declares the authentication mechanism supported by the interface, set to one from a list of predefined values: password, passwordHash, ticket, OTP, public key, certificate, SAML etc.

### 5.4.5 Application Component

The security-related properties of an Application Component might include:

- A `type`: this distinguishes components acquired through bespoke development from those purchased as commercial products. Strictly speaking, this is a derived property because the logical layer represents architectural – not solution - building blocks that can be associated with products. Nevertheless, it is useful to see this distinction in the logical model because it has a number of differences for secondary architecture processes such as testing, vulnerability management, vendor management etc.
- A `businessOwner`: responsible for authorising access and performing periodic recertification;
- A `technicalOwner`: responsible for defining a secure configuration;
- A `criticality` rating: a derived property determined by the most critical business process supported;

Application Components can also be associated to Software Defects.

### 5.4.6 Application Function

The key security-related property of an Application Function (and other internal behaviour elements) is the `authorisationContext` in which the function is executed. That is to say, it is important to declare when the application is performing the function using its own access rights or is acting as a proxy for the caller, under some form of impersonation or constrained delegation. The former indicates a highly privileged technical account that merits high assurance controls.

### 5.4.7 Software Defect

Software Defects are stereotypes of Vulnerability elements that are discovered during the Software Development Life Cycle (SDLC) e.g. Design Review, Static Code Analysis or Red Team Exercises. Like the Application Component itself, these defects are associated with specific products and versions – so strictly speaking part of the Technical Layer, where the element is described in full. Nevertheless, being a stereotype of a Motivation layer element (valid anywhere) and the fact that penetration testing, in particular, tends to uncover flaws in software behaviour, it can be very useful to associate functional defects with application behaviour elements directly.

### 5.4.8 Security Event

These elements can be used to model security-significant errors and exceptions thrown by application behaviour e.g. an exception encountered while validating a digital signature. In terms of model validation, it is essential that every event has at least one incoming trigger and at least one outgoing. The overlay supports two representations: a stereotype of Application Event with a `criticality` property and a regular Application Event with both `security` flag and a `criticality` property.



### 5.4.9 Data Object

The security properties of a Data Object declare any data-level privacy, confidentiality or integrity controls. The properties: `anonymisation`, `encryption` and `integrity` take values that indicate the mechanism used.

### 5.4.10 Security Configuration

This is a stereotype of Data Object that declares functionality that has been enabled in addition to the baseline (least common functionality) configuration. It is appropriate at the logical layer because different instances of the same building block may have different configurations in different contexts. Properties match those of the baseline configuration but with values toggled between `ENABLED` and `DISABLED`.

Table 26 presents a summary of overloaded elements discussed so far.

Table 26: Security Properties of Elements used in Logical Layer


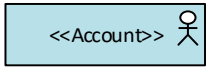
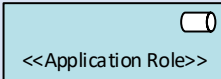
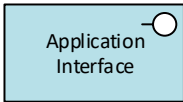


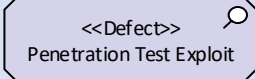

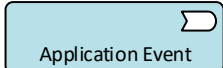
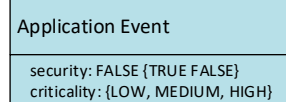
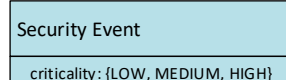
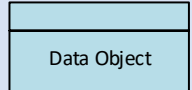
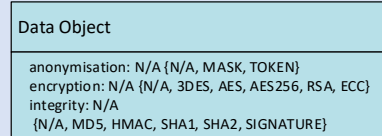
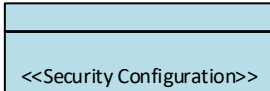
Element	Properties	Schema
	<p>Application Service properties should reflect any corresponding properties committed in an SLA / OLA.</p> <p>Whereas an SLA property defines the committed level, the corresponding Service property should declare the level expected (or being achieved).</p>	<div>Application Service</div> <pre>identityProof: {STANDARD, ENHANCED, ASSURED} authnStrength: {STANDARD, ENHANCED, ASSURED} authenticatorTrust: {STANDARD, ENHANCED, ASSURED} up-time latency: throughput: RTO: RPO:</pre>
	<p>An Account is the logical layer representation of a Principal from the perspective of a single application or target system. A Principal may hold multiple Accounts on multiple systems.</p>	<div>Account</div> <pre>domain: type: {NOMINAL, FUNCTIONAL, TECHNICAL}</pre>
	<p>An Application Role defines a set of access rights on a single target system.</p>	<p>An open schema in which to declare a user-defined set of access rights</p>
	<p>The <code>type</code> property distinguishes interfaces used by humans from those intended for machines.</p> <p>The <code>authentication</code> property declares the mechanism implemented at the interface</p>	<div>Application Interface</div> <pre>type: {COMMAND, GUI, RPC, MESSAGE} authentication: {PASSWORD, TOKEN, CERTIFICATE, ...}</pre>
	<p><code>type</code> declares the type of component acquisition.</p> <p><code>businessowner</code> is responsible for authorising access and performing periodic recertification;</p> <p><code>technicalOwner</code> is responsible for defining a secure configuration;</p> <p><code>criticality rating</code>: is a derived property set by the most critical business process that the application supports;</p>	<div>Application Component</div> <pre>type: {BESPOKE, OPENSOURCE, COTS, GOTS} (Derived) businessOwner: Manager technicalOwner: Manager criticality: {LOW, MEDIUM, HIGH} (Derived)</pre>
	<p><code>AuthenticationContext</code> indicates whether the function is performed using the Application's own access rights or those of the user by some means.</p>	<div>Application Function</div> <pre>authenticationContext: {SELF, PROXY, IMPERSONATE, DELEGATE...}</pre>



Table 26: Security Properties of Elements used in Logical Layer

Element	Properties	Schema
	<p>Vulnerability elements represent defects discovered in the software development lifecycle.</p> <p>It is often useful to associate them with the Application behaviour elements from where they might be exploited.</p>	Discussed in Technology Layer
 	<p>Models security-significant errors and exceptions thrown by application behaviour e.g. an exception encountered while validating a digital signature.</p> <p>Modelled by a stereotyped Security Event or a regular Application Event with a security property.</p>	 
	<p>Properties declare controls applied at the data level: <b>anonymisation</b> for privacy protection, <b>encryption</b> for confidentiality and <b>integrity</b> for tamper detection.</p>	
	<p>Security Configuration that declares functionality that has been enabled in addition to the baseline (least common functionality) configuration. Properties match those of the baseline configuration but with values toggled between <b>ENABLED</b> and <b>DISABLED</b>.</p>	<p>An open schema in which to declare the security configuration of the application as deployed in a specific application context.</p>

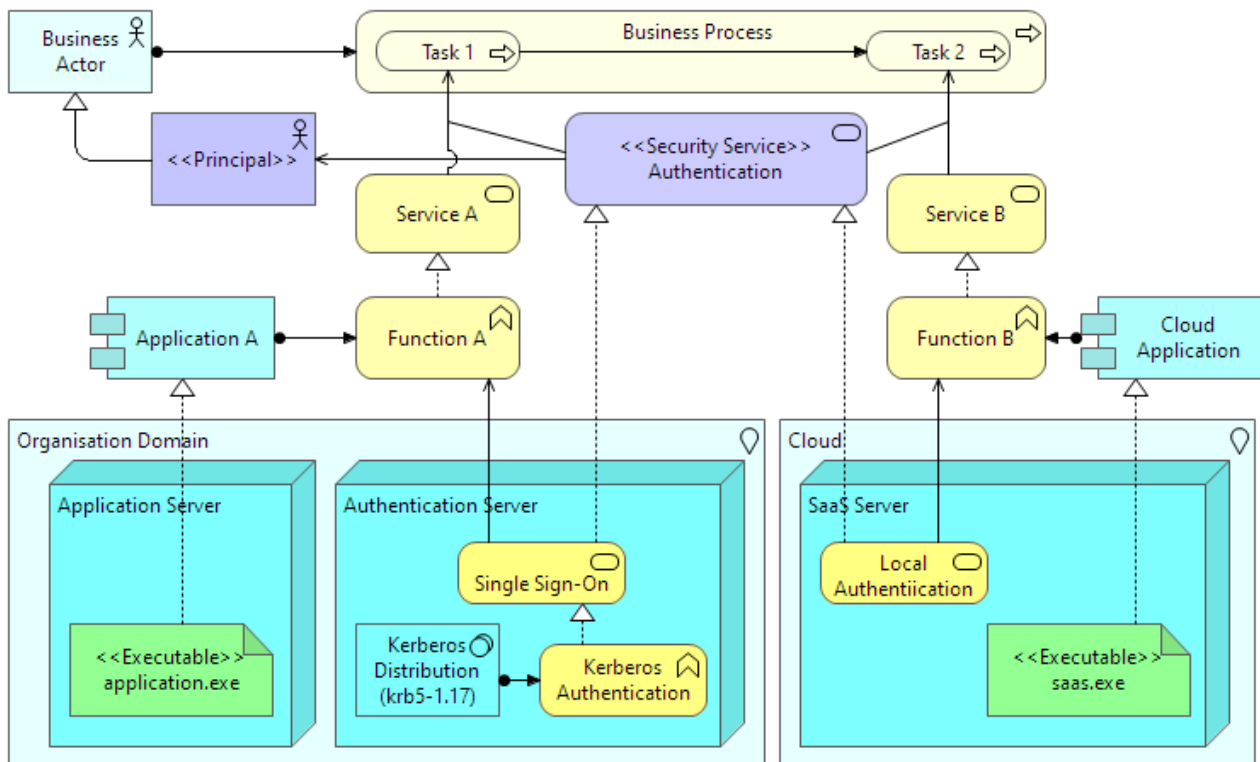
## 5.5 Modelling the SABSA® Physical Security Architecture

The SABSA Physical Security Architecture provides the security perspective of the builder's view of the System of Interest. It maps to the Technology layer of the ArchiMate core. This section proposes security-related properties for Technology layer entities that enable security analysis.

### 5.5.1 Technology Service

From a security perspective, Technology Services fall into two main categories: those that realise the security services identified in the Conceptual layer and those that provide general platform services to Applications.

Figure 37 illustrates the pattern by which a conceptual security service, in this case Authentication is realised by two distinct Technology Services: one which provides a single sign-on capability for the organisation domain; the other, the local authentication mechanism of a cloud application.



**Figure 37: Realisation of Security Services**

Similar patterns can be deployed for other security services provided by the infrastructure: Security Incident Event Management (SIEM), Privileged Access Management, Virus Scan, Data Leak Prevention and so on.

The closed loop between the invocation of the conceptual service and its technical implementation means that the model can not only validate that the Solution Building Block satisfies the assurance requirements declared by the Application Service but also trace to the compliance requirements shown previously in Figure 28 in the Policy Architecture section.

Turning to Technology Services in general, while the ability to enforce access control is as important as the earlier analysis of Application Services, it is often mitigated by virtue of being co-located on the same node, local network or network domain. Validation should focus therefor on invocations that cross trust boundaries.

### 5.5.2 Technology Interface

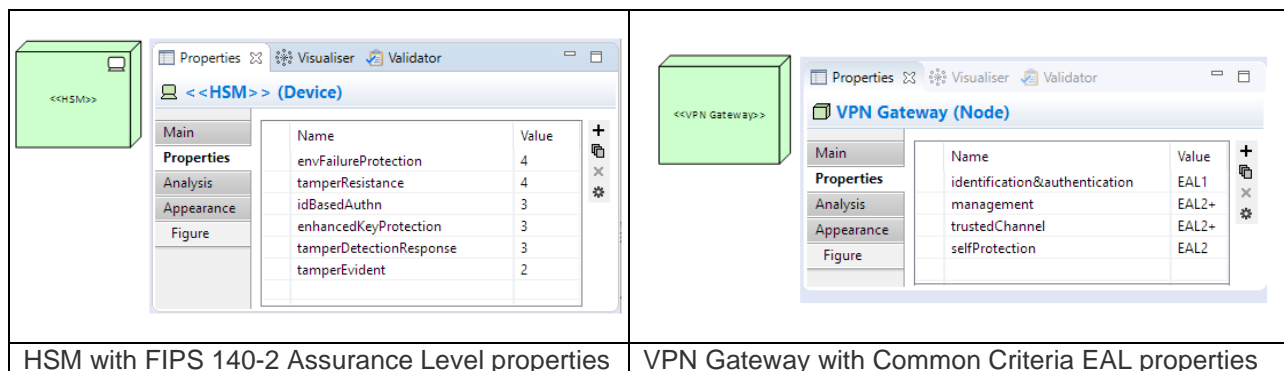
Technology Interfaces represent end points that specify how Technology Services can be accessed by other active structure elements in the environment. These may be other Technology Nodes, Application Components or Actors/Roles (e.g. to a Management Interface). The properties of the Technology Interface are essentially the same as those of the Application Interface.

### 5.5.3 Device and Node

Nodes represents a computational or physical resource that hosts, manipulates or interacts with other computational or physical resources. Devices model physical IT resources (i.e. hardware) upon which system software and Artefacts may be stored or deployed for execution.

When modelling security infrastructure, it is sometimes useful to model nodes and devices that have a specific security purpose as distinct stereotypes so that role-specific properties can be assigned.

For example, Figure 38 shows a Hardware Security Module (HSM) being modelled as a stereotyped Device with properties that reflect FIP-140-2 assurance levels and a VPN Gateway as a stereotyped Node with properties that reflect the Common Criteria Evaluation Assurance Levels (EAL) for the protection profile of type of product.



**Figure 38: Stereotyping Security Nodes & Devices**

### 5.5.4 System Software

The most important property of system software is its release version. This information is key to determining its patch status and any CVEs<sup>30</sup> (Software Defects) to which it may be vulnerable.

ArchiMateSE proposes two properties for this purpose: `version`, which declares the release version and a derived property, and a derived property, `patchStatus` by which it is made explicit whether the release version is current.

If the executable code carries some form of integrity protection, e.g. a hash or code signing, this can also be denoted using the `integrity` property.

### 5.5.5 Artefact

In standard ArchiMate, Artefact serves as the universal passive structure element of the Technology Layer. It is overloaded to represent any kind of data object in the file system: executables, scripts, data & configuration files, databases, documents, software key stores, specifications: everything in fact, except System Software.

For security modelling, Artefacts should be refined into types that reflect their specific purpose with appropriate properties. ArchiMateSE proposes an initial set of stereotypes: Executable, Data and Configuration.

<sup>30</sup> A list of [Common Vulnerabilities & Exposures](https://cve.mitre.org/) maintained by mitre.org

## 5.5.5.1 Executable

Executables are enriched with the same version control and file integrity properties as System Software. Commercial software can also be associated with CVE vulnerabilities (Software Defects), whereas bespoke software vulnerabilities may be associated with internally-managed defect reports, arising from static analysis, penetration testing, bug reports etc.

## 5.5.5.2 Data

Data Artefacts realise Data objects from the Logical Layer. Their security properties also reflect those of the Data Object but here, *encryption* and *integrity* refer to controls applied at the file/database level rather than those managed by the application. A *type* property is used to distinguish between artefacts stored in the file system and those that are part of a database.

## 5.5.5.3 Configuration

A configuration file associated with an Executable or System Software element in the Technology Layer represents a Baseline Configuration that should include all relevant security settings (secure by default). Different instances of this component in the Logical Layer may opt to enable or relax these settings to achieve a Least Privilege configuration in a specific application context. Figure 39 illustrates this principle

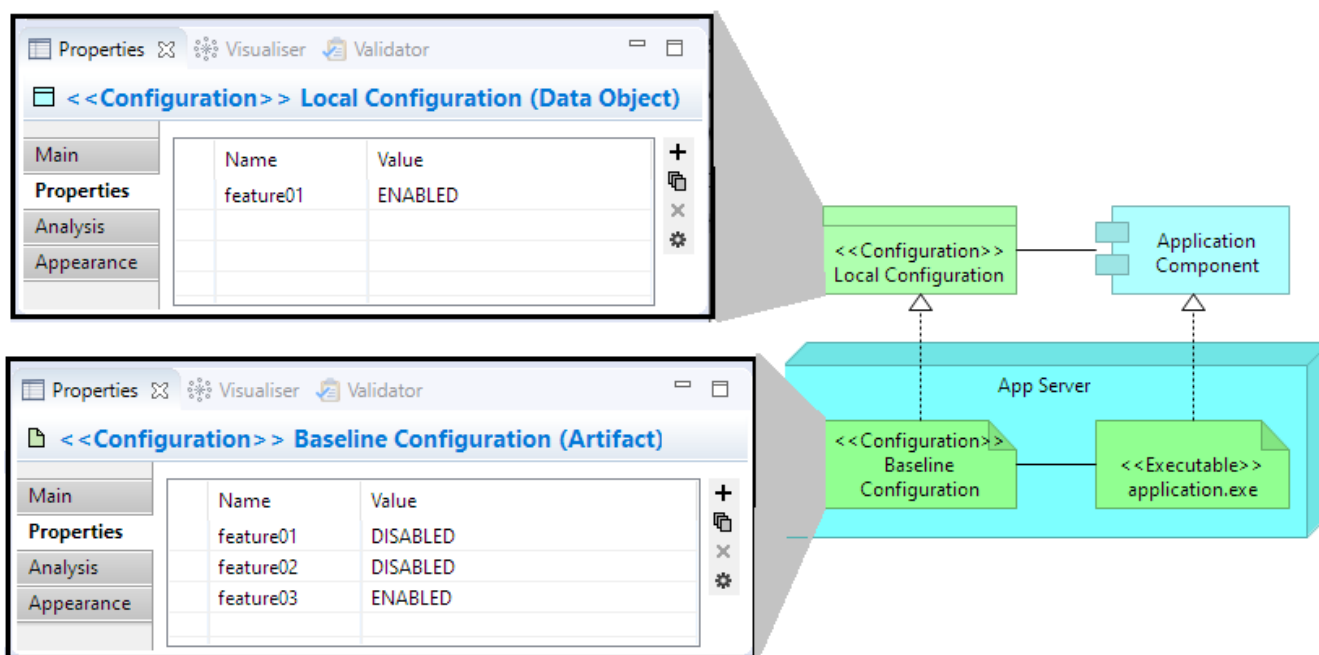


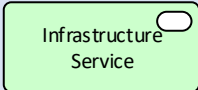
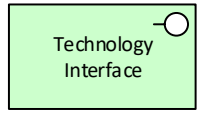
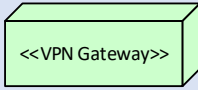
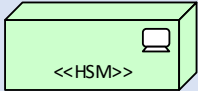
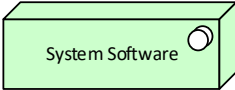
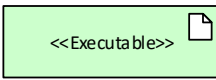
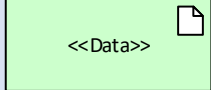
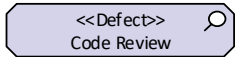
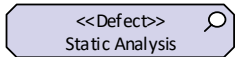
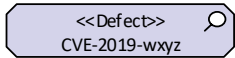

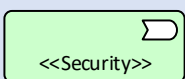
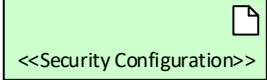
Figure 39: Configuration Files

## 5.5.6 Defect

Software Defects are stereotypes of Vulnerability elements that are discovered during the software development life cycle e.g. Design Review, Code Analysis, Bug Reports or Red Teaming. Such defects are associated with specific releases and versions – so strictly speaking part of the Technical Layer.

Table 27 presents a summary of Technology elements.

Table 27: Security Properties of Elements used in Technical Layer

Element	Properties	Schema
	<p>Infrastructure Service properties should reflect any corresponding properties committed in an OLA.</p> <p>Whereas an OLA property defines the committed level, the corresponding Service property should declare the level expected (or being achieved).</p>	<div>Service Level Agreement</div> <div> up-time latency: throughput: RTO: RPO: </div> <div>Technology Service</div> <div> identityProof: {STANDARD, ENHANCED, ASSURED} authnStrength: {STANDARD, ENHANCED, ASSURED} authenticatorTrust: {STANDARD, ENHANCED, ASSURED} up-time latency: throughput: RTO: RPO: </div>
	<p>The <code>type</code> property categorises the interface into types used by humans or machines.</p> <p><code>authentication</code> declares the authentication assurance level required by the interface</p>	<div>Technology Interface</div> <div> type: {COMMAND, GUI, RPC, MESSAGE} authentication: {PASSWORD, TOKEN, CERTIFICATE, ...} </div>
 	<p>Nodes and Devices that have a specific security purpose can be modelled as distinct stereotypes so that role-specific properties can be assigned.</p>	<div>Element Specific</div>
 	<p>System software properties declare the patch release version and status (derived) as well as any code integrity protection.</p>	<div>System Software</div> <div> version: patch: integ: {N/A, MD5, HMAC, SHA1, SHA2, SIGNATURE} </div> <div>&lt;&lt;Executable&gt;&gt;</div> <div> version: String patchStatus: {DOWNLEVEL, CURRENT} integrity: N/A {N/A, MD5, HMAC, SHA1, SHA2, SIGNATURE} </div>
	<p>The security properties of the Data artefact indicate any encryption or integrity controls applied at the file/database level.</p> <p>A <code>type</code> property distinguishes file system artefacts from those stored in a database.</p>	<div>&lt;&lt;Data&gt;&gt;</div> <div> type: FILE {FILE, TABLE, RECORD, FIELD} encryption: N/A {N/A, 3DES, AES, AES256, RSA, ECC} integrity: N/A {N/A, MD5, HMAC, SHA1, SHA2, SIGNATURE} </div>
  	<p>Defects are stereotypes of Vulnerability elements that are discovered during the Software Development Life Cycle (SDLC) e.g. Design Review, Static Code Analysis, Bug Reports or Red Team Exercises. Such defects are associated with specific releases and versions – so strictly speaking part of the Technical Layer.</p>	<div>&lt;&lt;Software Defect&gt;&gt;</div> <div> criticality: {LOW, MEDIUM, HIGH} </div>
 	<p>Models security-significant errors and exceptions thrown by infrastructure e.g. a SIEM Alert. Modelled by a stereotyped Security Event or a regular Technology Event with a security property</p>	<div>Technology Event</div> <div> security: FALSE {TRUE FALSE} criticality: {LOW, MEDIUM, HIGH} </div> <div>Security Event</div> <div> criticality: {LOW, MEDIUM, HIGH} </div>
	<p>Security Configuration declares functionality that has been enabled in the baseline (least common functionality) configuration. Properties are user-defined but set to either ENABLED or DISABLED</p>	<div>Element Specific</div>

## 6 Conclusion

This paper proposes a viable security overlay, capable of supporting the SABSA Architecture approach with the ArchiMate EA modelling language. The expected benefit of such an alignment is that ESA concepts can be incorporated into a unified EA model that is shared with other architects and reflects the reality of the enterprise's single architecture. Security becomes an integral part of the design.

The ability to generate consistent views of a single model will benefit all who rely upon them: the more accurate, consistent and complete the model, the better the analysis, decision-making, change planning and execution is likely to be. Should a rudimentary alignment have proved impossible, the prospect of creating distinct EA/ESA models, defined in different notations, by different tools, by different teams would be a death knell. Synchronisation would likely to be so complex and resource-intensive as to be unviable.

Equally important as what security brings to the EA model is what modelling brings to security.

Firstly, visualisation: security models are a complex mesh of concepts and relationships that are far better-suited to representation by a simple, yet expressive notation of symbolic nodes and lines than as checklists and matrices. Because diagrams are much more intuitive to create, understand, verify and review, the result is both more usable and of superior quality.

Secondly, the economics of modern IT drive projects to deliver more value earlier and repeatedly on ever shorter cycles. These “Agile” approaches shift emphasis heavily towards functionality, too often to the detriment of good structure and the planning, blueprints and documentation overhead that go with it. In this world, architecture is redefined as the opposite of Agile: the decisions that need to be made correctly at the start of the project because they are difficult to change afterwards. SABSA teaches us that security architecture is not another architectural layer but a cross-cutting concern, affecting all EA layers and as such, is especially exposed by Agile methods. How are we supposed to perform a security design review when there is no design?

It's not that Agile is implacably opposed to documentation – only that it should be an enabler and not a drag on the project. To this end, scaled Agile methodologies are turning towards Model-Based System Engineering (MBSE) as a means to keep the artefact set complete, up-to-date, consistent and tailored to specific stakeholder views.

By generating these artefacts as views of a single underlying model, this overhead is vastly reduced. Reliable documentation is maintained quickly and easily. Re-use becomes possible on many levels: elements, patterns and even analysis. As these models expand beyond the logical design (UML) to full system models (ArchiMate), a security overlay will enable SABSA itself to be practised lighter, faster and better adapted to the short-cycle, agile, minimum viable approach of modern projects.

Lastly, because ArchiMate models are defined in a semi-formal notation that can be exported in a standardised XML Exchange Format, they are machine-readable. The security overlay outlined in this paper

has maintained an eye on its potential use, not only for documentation but also to query and analyse them via automated means. Should a security notation such as the one described here become a standardised extension to the language, the prospect of tools to support, for example, policy compliance checking or computer-assisted requirements generation would become a realistic proposition.

In demonstrating this alignment, relatively few fundamental issues have been encountered. The most significant of these are the different concepts of an architectural Principle and the absence of the elements necessary to populate the Conceptual layer for which stereotyped elements and the adoption of conventions for Motivation patterns, these have been resolved without breaking the grammar and retaining a degree of elegance.

Although there is still a significant amount of work still to do, there is a basis here to work towards viable security patterns in ArchiMate that may well lay the groundwork for a standardised security extension to the language in the future.