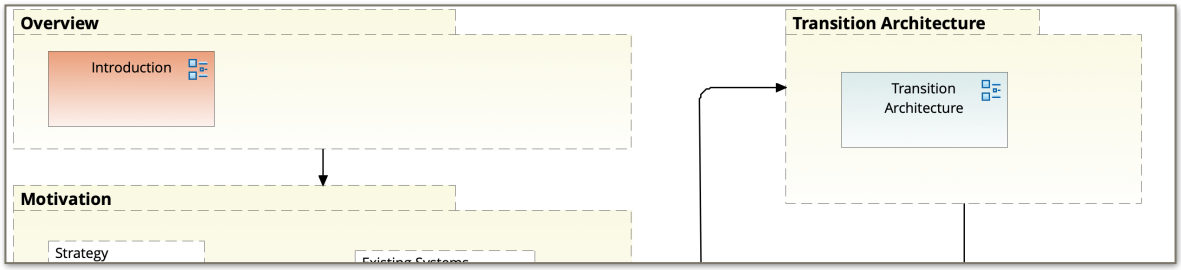
Richard Heward
Tame Blue Lion Ltd
8th October 2019

Documenting architecture models using ArchiMate

How using ArchiMate can help to define and create architecture deliverables



Introduction

This paper describes a method of defining documentation structures in the ArchiMate® notation[4] and goes on to use a 'driving view' to generate output from the Archi® enterprise architecture modelling tool[2]. This can be used to repeatedly create consistent looking architecture deliverables on a project. This example uses Archi and its jArchi scripting language[1] although other EA tools could achieve something similar.

It was created by the need to offer a trusted, up to date, document that can easily be accessed by non-tool users. The use of Markdown as the output format was the simplest one to try first.

Defining a document

An architecture deliverable document is often a structured set of views and accompanying descriptions. This is modelled using a simple ArchiMate view that uses a set of concepts to represent each section or subsection, and a means of representing the order. Archi supports creating 'view of views' by just dragging a view in from the model tree onto a view (as a 'View Reference').

I chose to use the grouping concept as these allowed for triggering relationships between them to show the order of sections in the document. Each View Reference needs to reside in a group, even if there's only one View Reference.

Generating the output

Archi has its own reporting capability using Jasper Reports or generating to HTML, but I found this to be quite complex to generate specific documentation. Archi supports scripting using its Patreon supported jArchi javascript based utility.

Here's a simple driving view to explain the components:

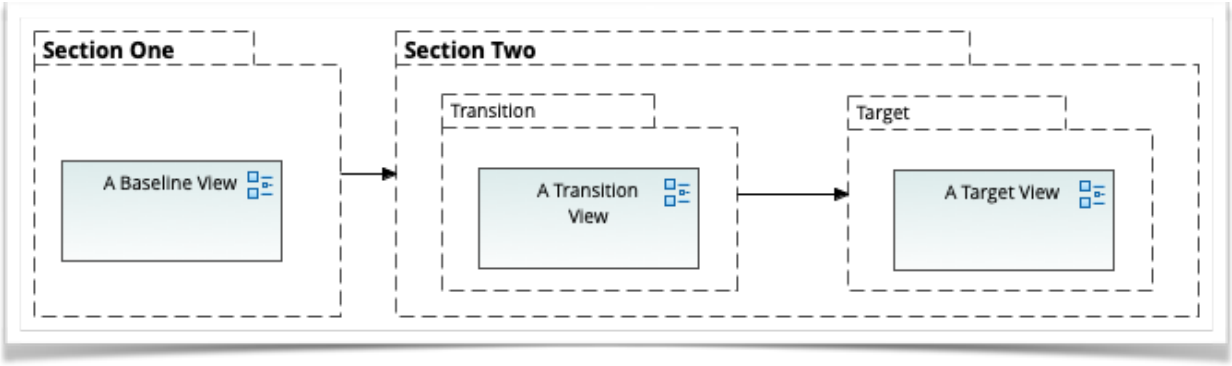


Figure 1 - A simple driving view

The script uses such a driving view and starts with the group that has no incoming trigger (Section One, in this example). It follows the triggers until it reaches a group with no outgoing trigger (Section Two). If there are sub groups (Transition and Target), it will also create a sub levels (level 2, here) for these.
Any documentation entered in the properties for the groupings will be extracted (representing its section text). The script generates an image for each Reference View it finds inside the groupings.

There are a few limitations, of course. It currently does not export the properties, although that should be a simple addition (see 'Next Steps'). The need for a sub level groups feels a little clumsy and could be left out if you don't care what order views are generated in.

Running the script prompts for an output filename (defaulting to the driving view name). The console log (in Figure2) shows the progress:

```
Documentation Generation @Mon Oct 07 2019 18:32:31 GMT+0100 (BST)
Driving view is: Simple Document
path: /Users/richard/Downloads/
fileName: Simple Document.md
Level 1. indent:# (Section One)
Level 2. indent:## (A Baseline View)
Level 1. indent:# (Section Two)
Level 2. indent:## (Transition)
Level 3. indent:### (A Transition View)
Level 2. indent:## (Target)
Level 3. indent:### (A Target View)
Done
```

Figure 2 - A console log output

Figure 3 shows a snapshot rendered with an application called 'MacDown' (I use a Mac; alternative utilities are available for Windows).

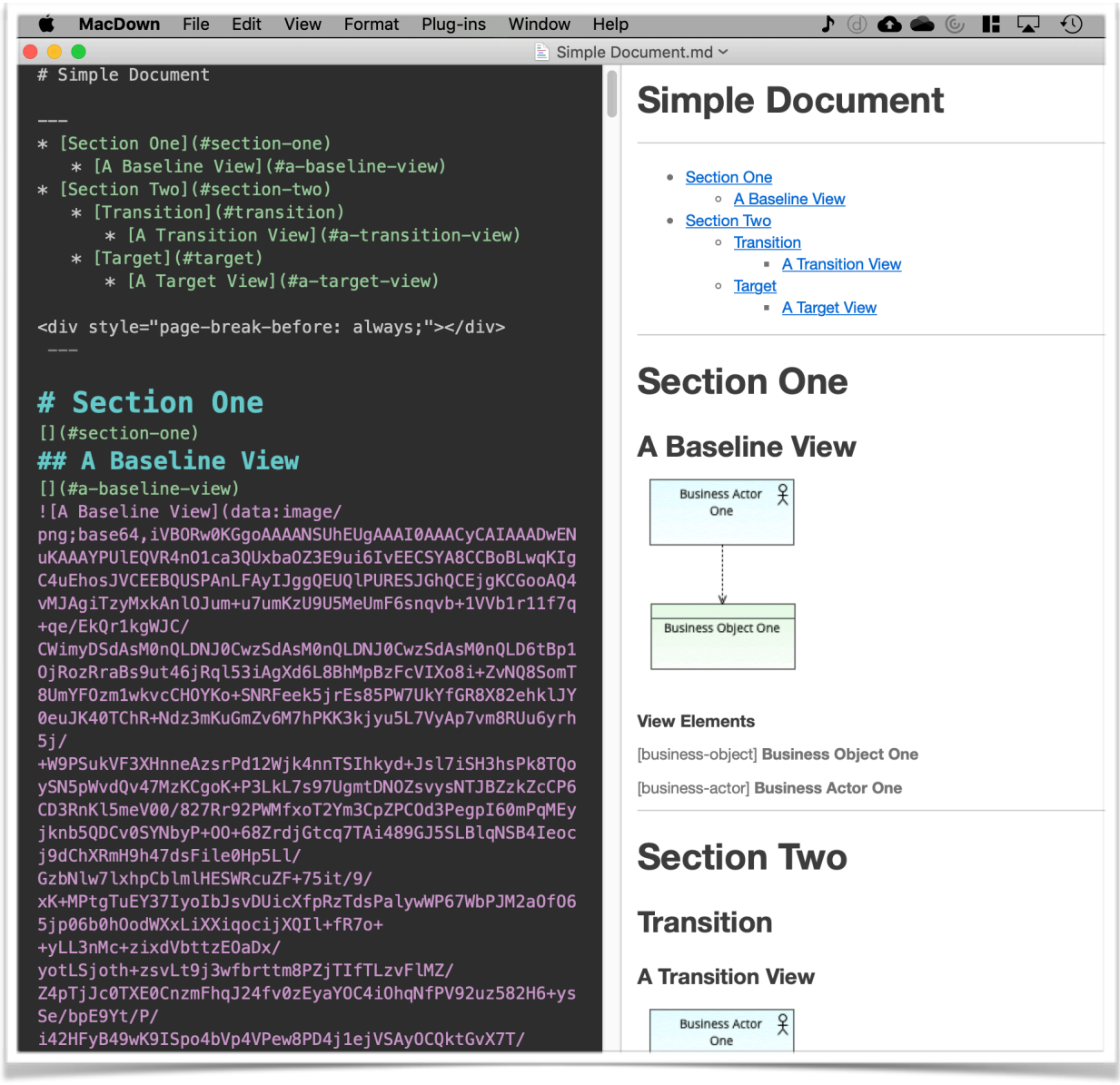


Figure 3 - Rendered markdown file

Note the table of contents that hyperlinks to the corresponding section.

The 'View Elements' section is generated for each view just to list the object type, name and any documentation that has been entered. If this is too much, the script allows for this to be suppressed by creating a property on the driving view called 'ExcludeViewElements', set to true.

A web browser (such as Google Chrome) can be used to open the file, as shown in Figure 5.

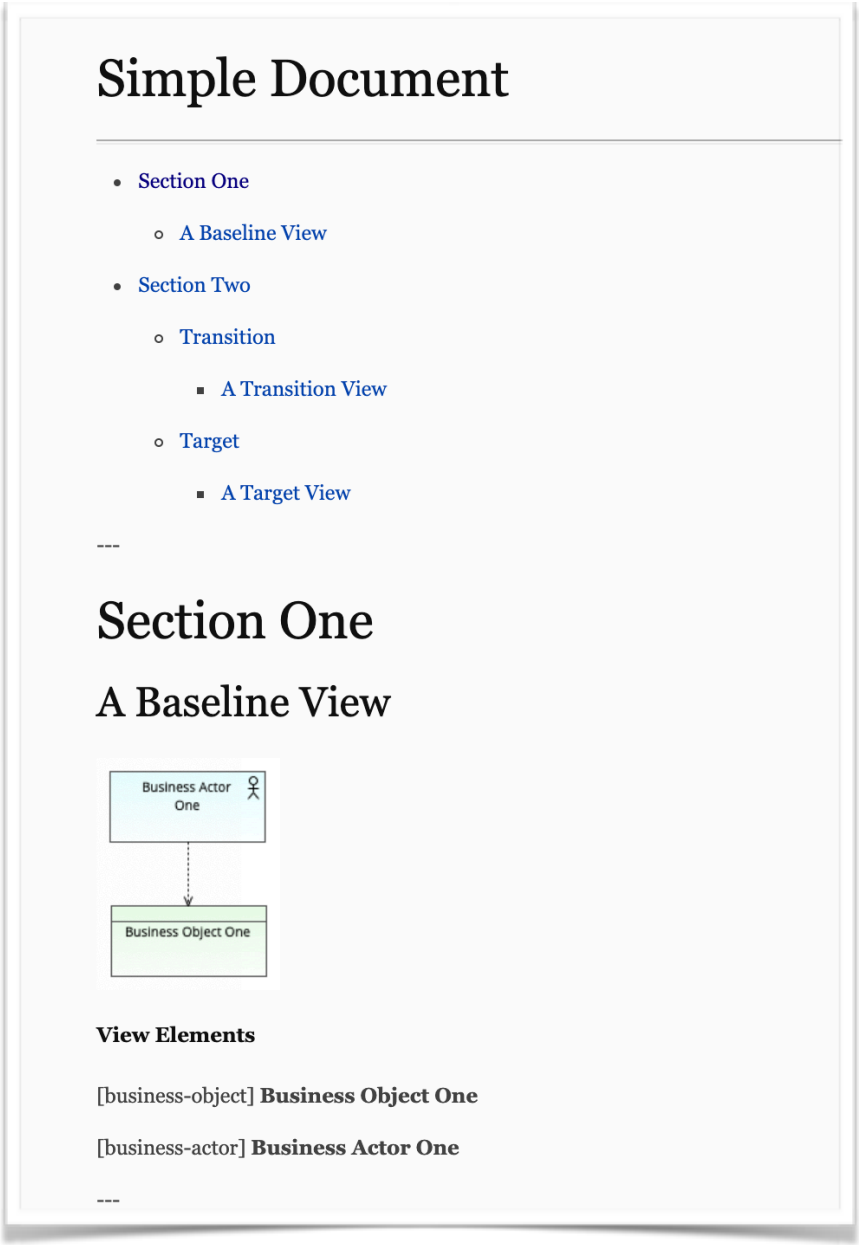


Figure 4 - Rendered markdown file in Chrome

The horizontal rule doesn't always render (e.g. in Google Chrome), so there may need to be some compromises without a formal Markdown engine in place.

The script supports multiple document levels through further sub-nesting of the groups, through some clever recursion. This is only limited by the ability for Markdown to render the number of levels used.

Practical Use of driving views

Figure 5 shows a more realistic example driving view which could be the basis for a standard architecture high level design.

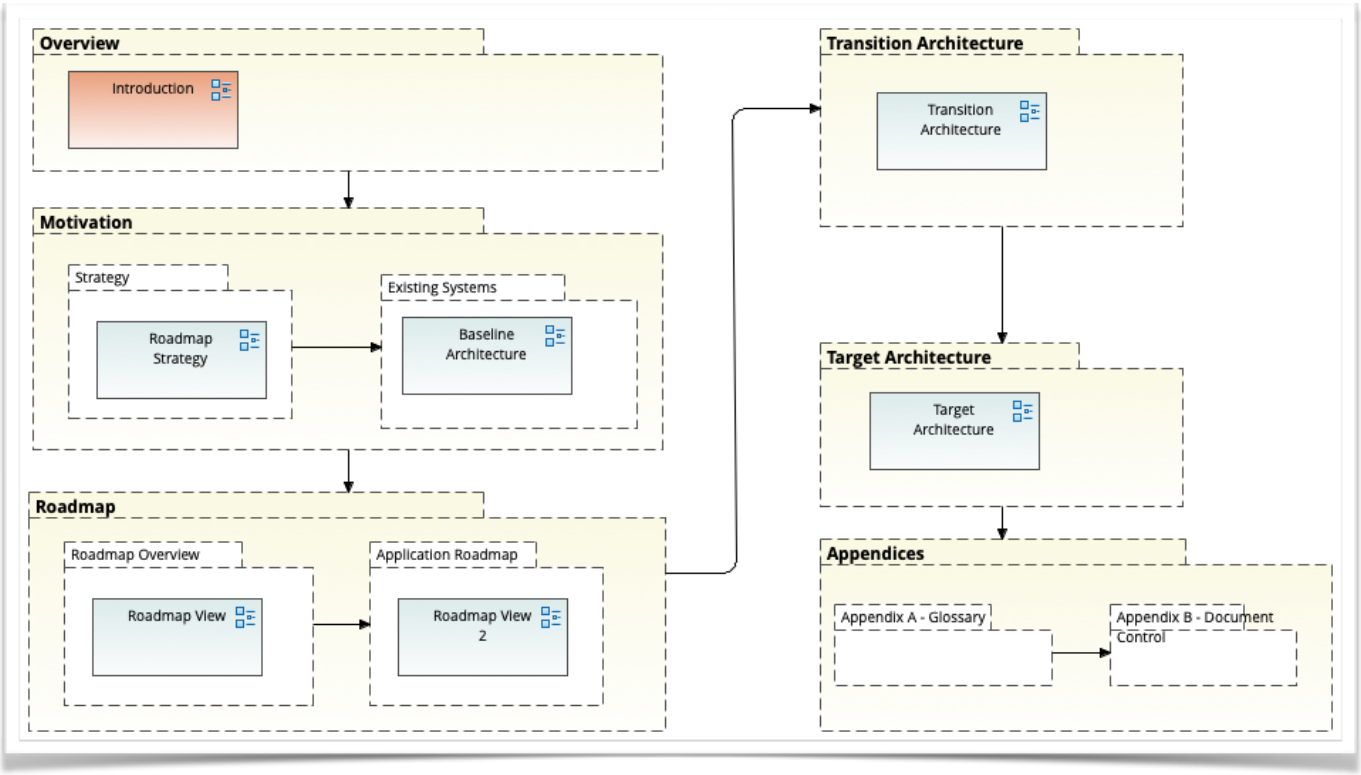


Figure 5 - A more complex driving view

It is possible to make the view conform to an organisation's standard for documentation structure. Each project would then only need to have the appropriate views created and dragged in. This is a great way to drive conformance and to do completion checks for a design review. This could be scripted or just done manually.

There would have to be a balance on how much design is done in ArchiMate and Archi compared to adding text later directly into the document using a more natural word processor, but that's always the challenge. A recent engagement formed the guidance that it was best to only have true architecture content in the model, and project/contract related text added directly into the document afterwards (or as part of a template).

Using the script to repeatedly create initial versions of a document until all the views are in place is useful, even if manual fixing and layout is done at the end.

Next Steps and Ideas

Whilst the aim is to keep things simple, the intention is to refine the script to allow for some of the following features:

- Export to Open Document Format, for MS Word, although utilities such as 'pandoc' does a good job of the conversion. This requires the ability to link and call a compatible javascript 'word' library in jArchi.
- Conversion to PDF is probably best left to pandoc.
- Export to Confluence via its API.
- Output to HTML (although the use of Markdown probably negates this).
- Include ArchiMate icons in the View Elements list.
- Include an option of object properties to be documented in a table.
- Spell checking in Archi?

Resources

The jArchi 'documentation.ajs' code is published to github for anyone to use:

https://gist.github.com/rich-biker/9a3c86c5a576ce0d8639856f3ee81651

Also credit to Steven Mileham's Markdown script gist from which I needed a small function for Markdown manipulation[3].

References

1. jArchi wiki pages

https://github.com/archimatetool/archi-scripting-plugin/wiki

2. Archi – Open Source ArchiMate Modelling

https://www.archimatetool.com

3 Exporting to Markdown by smileham

https://smileham.co.uk/2019/03/13/jarchi-scripting-export-to-markdown/

4. OpenGroup ArchiMate

https://www.opengroup.org/archimate-forum/archimate-overview