25/09/2023

The Lego State

A guide to thinking in components
(Version 1.0)

Gordon Guthrie
RESEARCH FELLOW AT SCOTTISH GOVERNMENT
BIUS WORKING PAPER NO 3

Table of Contents

1 Introduction

1.1 What is a Lego state?

A lego state is one where the administrative, regulatory and financial state is no longer a large group of unique, hand built (and perhaps hard to navigate) systems.

It's one where the digital state is built from common, and shared, components. Some are stand-alone services: identity, payments. But some are occluded from the citizen: shared organisations, shared digital systems, shared databases, shared legislative solutions.

It's a state where the thinking that has made Lego and IKEA and car manufacturing and high street coffee chains such successes are applied to government. The twin paradoxes of standardisation:

- choice by assembly, where standard components are assembled into tailored solutions
- flexibility from rigid composition

1.2 Who are you?

You are a policy person, a service designer, a data architect, a delivery manager, a member of a project team, an operational manager, an elected representative. You are in government or opposition. You work at a thinktank, or in parliament, or government, as a civil servant or political advisor. You care about how we build an efficient and effective state in the digital age – one that has the best outcomes, is easiest and unobtrusive to use and has the lowest costs possible.

1.3 Why should you read this?

There is a thinktank-to-government-services assembly line that takes ideas, turns them into policies, and legislation and onto departments and systems and service delivery. You should read this to help you think about this as an improvable process, one that you can shape to get the better outcomes you want, to help you speak to your colleagues in a shared language that will let you change processes and make the trade-offs in an informed manner.

2 The Blus Project

This is Working Paper No 3 of *Blus - Basic Law-Making For Legislative Computer Systems* which is a research project looking systemically at how the state creates the digital systems underpinning its services.

Working papers are being released gradually for comment:
Working Paper 0 – ***The locus of change*** (forthcoming)
Working Paper 1 – ***Data and the rule of law*** (published)
Working Paper 2 – ***Rules as code*** (published)
Working Paper 3 – ***The Lego state*** (this document)
Working Paper 4 – ***The remixable state*** (published)
Working Paper 5 – ***Law reform for data*** (forthcoming)
Working Paper 6 – ***A solera for data cleansing*** (forthcoming)
Working Paper 7 – ***Experimental digital legislative processes*** (forthcoming)

Blus working papers are designed to stimulate discussion about key elements of the relationship of the state to digital systems and their delivery. Your feedback, input, and particularly criticisms of this paper are most welcome. Feel free to distribute it however you wish.

Working papers are published via the *Digital Policy* SubStack.

Author/contact: gordon.guthrie@gov.scot or subscribe to Digital Policy | Gordon Guthrie | Substack[1]

The author is an independent Research Fellow at Scottish Government under the First Minister's Digital Fellowship programme. The views of this paper do not represent the views of Scottish Government.

[1] https://digitalpolicy.substack.com/

3 Context

3.1 How to think about componentisation?

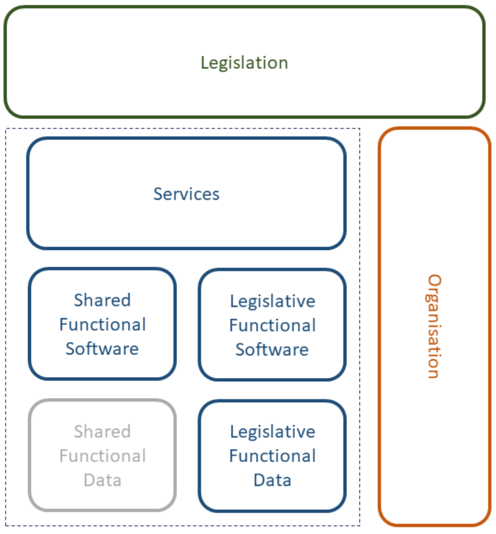
Developing complex propositions should usually follow the think big/build small model. This document unabashedly ***thinks big***. Consequently it is appropriately shallow. It is about sketching end-state patterns in broad outline.

Think Big
Build Small

When developing your lego strategy you should do the opposite – ***build small*** – and identify the smallest programme of work that will implement components. Legoisation is about capability more than it is about outcomes – if the capability is grown the outcomes will follow.

3.2 Legoisation described

Legoisation is the process of moving from hand-building state systems to assembling them from components. It can happen in many places.


Legislation
Services
Shared Functional Software
Legislative Functional Software
Shared Functional Data
Legislative Functional Data
Organisation

3.2.1 Legislation

The law matters. Ultimately the computer systems have to do what the law says. Consider licensing schemes. Governments issue lots and lots of licenses to do things. The more custom the laws for various licensing schemes are, the wider the range of behaviours of a single deployed licensing software system would have to implement. So legoising legislation should shrink the amount of work. It should also reduce the costs of developing legislation and policy.

3.2.2 Services

The citizen experiences the state as services – provided by people or digital systems or a mixture of both. Things like tax and social security are experienced as web pages or call centres, as data to be provided and hoops to be jumped through, as success of failure and appeal against the decision, finding, navigating and logging in, getting paid.

3.2.3 Software

Services are the implementation of the services in code – either built as a single system or composed from multiple services.

3.2.4 Data

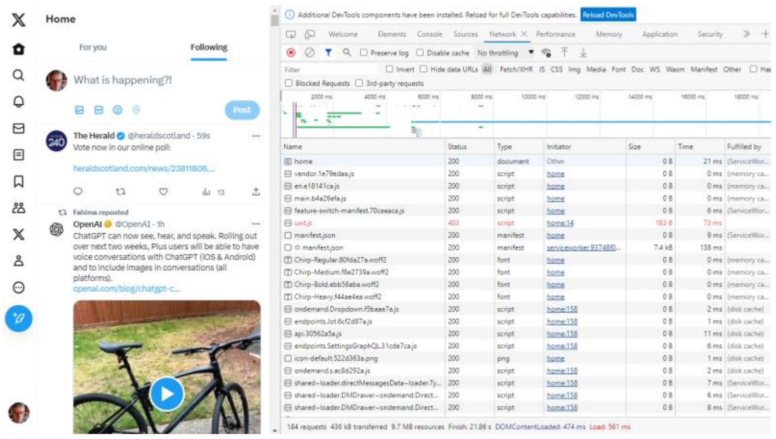
Code and services typically change and mutate, but underneath them is data which is the persistence of customer information in a database or file system. Data comes surrounded and hedged with procedures: who can do what to it when and under what circumstances.

3.2.5 Service Composition

Typically a software deployment will consist of a service, or services, implemented in software and with its own data persistence/database layer. These three elements tend to be fairly tightly coupled which is why they are shown linked in the diagram.

The service is usually a composition of many different tech and data components – what you might call technical services. This composition (and encapsulation) is a key feature, lots of small components become one larger one at a higher degree of abstraction.

For example the home page of my Twitter account on the web calls 184 different end points to retrieve the data and layout to display my Twitter.



The Herald @heraldscotland · 59s
Vote now in our online poll:
heraldscotland.com/news/23811806....

Fahima reposted
OpenAI @OpenAI · 1h
ChatGPT can now see, hear, and speak. Rolling out over next two weeks, Plus users will be able to have voice conversations with ChatGPT (iOS & Android) and to include images in conversations (all platforms).
openai.com/blog/chatgpt-c...

Name	Status	Type	Initiator	Size	Time	Fulfilled by
home	200	document	Other	0 B	21 ms	(ServiceWor...
vendor.1e79edaa.js	200	script	home	0 B	0 ms	(memory ca...
en.e18141ca.js	200	script	home	0 B	0 ms	(memory ca...
main.b4a26efa.js	200	script	home	0 B	0 ms	(memory ca...
feature-switch-manifest.70ceeaca.js	200	script	home	0 B	6 ms	(ServiceWor...
uwt.js	403	script	home:14	183 B	73 ms	
manifest.json	200	manifest	home	0 B	9 ms	(ServiceWor...
⚙ manifest.json	200	manifest	serviceworker.93748f0...	7.4 kB	138 ms	
Chirp-Regular.80fda27a.woff2	200	font	home	0 B	0 ms	(memory ca...
Chirp-Medium.f8e2739a.woff2	200	font	home	0 B	0 ms	(memory ca...
Chirp-Bold.ebb56aba.woff2	200	font	home	0 B	0 ms	(memory ca...
Chirp-Heavy.f44ae4ea.woff2	200	font	home	0 B	0 ms	(memory ca...
ondemand.Dropdown.f5baae7a.js	200	script	home:158	0 B	2 ms	(disk cache)
endpoints.Jot.6cf2d87a.js	200	script	home:158	0 B	1 ms	(disk cache)
api.30562a5a.js	200	script	home:158	0 B	11 ms	(disk cache)
endpoints.SettingsGraphQL.31cde7ca.js	200	script	home:158	0 B	6 ms	(disk cache)
icon-default.522d363a.png	200	png	home	0 B	1 ms	(disk cache)
ondemand.s.ac8d292a.js	200	script	home:158	0 B	2 ms	(disk cache)
shared~loader.directMessagesData~loader.Ty...	200	script	home:158	0 B	7 ms	(ServiceWor...
shared~loader.DMDrawer~ondemand.Direct...	200	script	home:158	0 B	6 ms	(ServiceWor...
shared~loader.DMDrawer~ondemand.Direct...	200	script	home:158	0 B	8 ms	(ServiceWor...

184 requests 436 kB transferred 9.7 MB resources Finish: 21.86 s DOMContentLoaded: 474 ms Load: 561 ms

3.2.6 Organisation

The final wrapper is organisation. Software service don't run themselves but are wrapped in people. And organisations can be layered. One organisation can develop, deploy and manage software that is used by another organisation. Development, deployment and management can themselves be split out by organisation.

Organisation is different to the other domains here – because ultimately human beings are the determining factor – all the other components and the contracts that we use to make them interoperate are creatures of the imagination, things human beings tell other human beings to help them organise their work co-operatively.

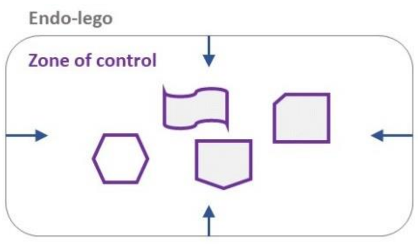
3.3 A Lego state has capability

A Lego state is a state that has the capability to create digital, legislative and organisational components and compose those components in more effective ways.

Legoisation is not something that can be bought, or applied, or completed. Becoming a Lego state involves a host of changes. Mindset, process, organisation, training and incentivisation, career and financial flows to name a few.

Modern states and civil service organisations are already a long way on the journey – albeit mostly as ***endo-legoisation***. Different professions are producing pattern books within their own domain.

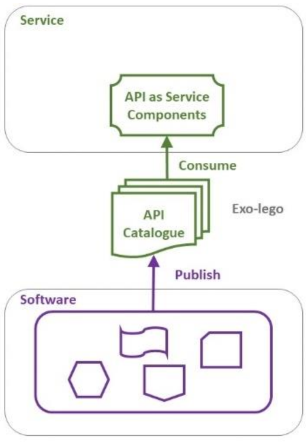
Endo-lego is made by professionals within their zone of control – data experts making standard data components, UX experts making HTML controls with JS and CSS libraries to support them and so on.



Here is a pattern book for common web components from the Government Digital Service.uk gov. The Scottish Government's Parliamentary Counsels have a pattern book *Common Legislative Solutions*. Two wildly different professions, web developers and legislative drafters.

And sometimes these patterns are crossing domain boundaries – the web components design pattern book has been implemented in code at GOV.UK Frontend. This is the first step in ***exo-legoisation*** – where patterns in *my discipline* start being promoted as canned solutions with catalogues in *your discipline* and we started seeing alignment across organisational boundaries.

Exo-lego is related to endo-lego by a translation and a promotion. An example would be written standards (this is how we build APIs) becomes a framework for making APIs. An instance of that API is exposed in a catalogue for Service Designers to use – and the API is consumed as a Technical Service Component. It is a promotion because the Service layer is at a higher level of abstraction than the software layer.



That process of how you take identified patterns in one world and translate and align them in another to simplify delivery is what this document is all about.

And the fact that it is an exo-process and not an endo-one should tell you a lot about how to think about it. If you are only talking to your team, your peers, your profession, then you are only talking inward facing - endo. Sorting out your professions view is an important part of the game – you can't align with anyone else if you don't, but it is simply the beginning, getting to the start line, the base camp. The finish, the summit still awaits you. When you are componentising your team, like your output must be endo, outward facing, including people ***NOT*** from your zone of control, ***NOT*** from your zone of comfort, ***NOT*** from your zone of expertise.

3.4 Promotion of components

One of the core concepts in component thinking is promotion – and the key thing is that there are lots of routes to promote. The endo/exo example took a well understood one software patterns to API-as-a-service but there are plenty more.

A standardised component is promoted from a pattern, a way of doing things, to a thing to be used, to be selected from a catalogue of things.

This is best explained with an example. Data on property was standardized[2], with the creation of UPRNs (unique property record numbers) and USRNs (unique street record numbers). This was a written standard – you should use their attributes in your data schemas.

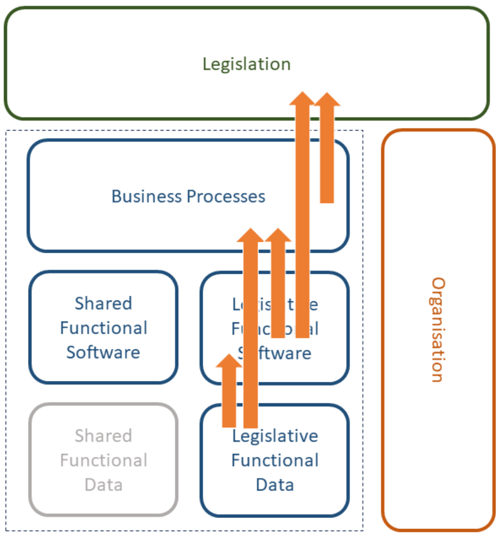
[2] Identifying property and street information - GOV.UK (www.gov.uk)

This didn't standardise postal addresses though – my flat can be described equally well as:

- 16-5 Hart Street
- 16/5 Hart Street
- Flat 5, 16 Hart Street
- Top Left, 16 Hart Street
- TL, 16 Hart Street
- 16-5 Hart St
- 16/5 Hart St
- Flat 5, 16 Hart St
- Top Left, 16 Hart St
- TL, 16 Hart St

The standard (UPRNs and USRNs) was ***promoted*** to a web service AddressBase[3]. Before it was a description of how to do something (but you had to build the thing), now it is a web API (which you can just consume).

Generally legoisation is about promotion – up the stack – and there are lots of routes:


Legislation
Business Processes
Shared Functional Software
Legislative Functional Software
Shared Functional Data
Legislative Functional Data
Organisation

In this case data has been promoted to a web service – the left hand, smallest and lowest orange arrow on the diagram.

All promotions, irrespective of their 'from' and 'to' have the same characteristics:

- take something that is standardised in one domain and translate and transpose it into a more abstract but different domain
- make it technically available in some fashion
- publish it in some sort of catalogue so that a range of different organisations can find and use it

Critically publication must be a co-operative act with the consumers and not something imposed on them. Components are only components when they are used – otherwise they are just white elephants

[3] Access free address data using AddressBase - GOV.UK (www.gov.uk)

3.5 Remixability

Remixability is a property of a Lego state – the ability to reorganise institutions and refocus attention from one area of society to another. It is part of the world of componentisation because it involves reassembling components into different organisational structures, but, but…

There is a paradox of decentralisation – in order to decentralise digitally there are core components that you need to centralise first.

Remixability falls squarely into this world – and for that reason it is the subject of a different working paper ***Working Paper 4 – the remixable state*** (you can find it on SubStack at DigtialPolicy[4]).

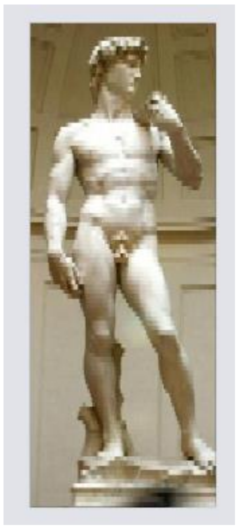
3.6 Tradeoffs

There are a range of trade-offs that need to be made.

Legoisation is a lossy process, you lose the ability to hand craft details. The degree of loss depends on the brick size. Imagine a statue of Michelangelo's David made in lego. Would it be good enough? Imagine it in duplo (the big lego for toddlers). Imagine it in breeze blocks.

So there is a brick-size dial that you can twiddle to trade off.

Marble


Lego

Duplo

Breeze blocks[5]

This loss of resolution is the consequence of the rigidity that enables the flexibility. During the design of your components you must focus on the rigid constraints, they should be as small and as few as possible, but no lesser and no smaller. If they stop being rigid they stop being able to define composable components.

[4] https://digitalpolicy.substack.com/

[55] Modern art is rubbish, innit? Just a lot of old breeze blocks and yeah that's me being polite, you can't even see if he's in the nip

The trade-offs will be described in terms of ladders as much as possible. Options on the lower rungs are suitable when there is low levels of functional alignment and the more alignment the higher up the ladder you can go.

Exactly how much do you want to componentise, and how much must be custom and how do you match the two? At the highest level each organisational unit might only need to be able to skin services into its branding with its logo and colours, but there will be other reasons for customisation that might require more structural divergence.

A good way to understand componentisation is to look at the Volkswagen family of cars. Audis, VWs and Škodas share about 75% of the same components, engines, chassis, gear trains, electrics and so on. But the variation allows the creation of product lines for premium, family and cost-conscious customers. (The book *The Machine That Changed The World* [6] is the classic read about this and the birth of lean production).

Once upon a time there was a separate car factory per brand – one for Audis, one for each VW marque, one for Škodas. By componentisation wider ranges of cars could first be built on the same line, then lines could be merged until one factory can churn out lots of different types of car. Gradually the cost base was shrunk and squeezed. And this restriction is experienced by the customer as an explosion of choice, making things swap-out in the macro makes them swap-out in the micro: seat fabrics, in-car music systems, colour, lights, windows, engines, wheels, all now offered in a range unavailable 20 or 30 years ago. "Any colour they like as long as it's black" it surely ain't no more.

As we go through the layers the same rough'n'ready reckoning will be used based on a VW percentage. The lower the VW percentage (low % of shared components) the lower down the ladder you will componentise (and the more operational systems you will end up with). The higher the number, the higher the componentisation and the less systems you end up with.

At a software level there is a centralised/decentralised trade-off. A software system for each of the many thousands of license types that Scottish Government issues is definitely excessive. But one software system for them all is probably not enough.

Centralising might reduce deployment cost at the expense of slowing down systems maintenance and increasing operational costs. Very similar systems can be economically supported by a common system, but too much variation, even if it is implemented in pluggable blocks, doesn't necessarily make for the best result. A single Scottish government system would impose massive cross-departmental communication overhead.

Legoisation is about making choices and trade-offs. There is no one correct answer – and the best answer is determined not by where you want to go on its own, but also where you are starting from – the current landscapes: legal, organisational and technical.

[6] https://www.amazon.co.uk/Machine-That-Changed-World/dp/1847370551/ref=sr_1_1?crid=2DJ9X7YQUH79&keywords=machines+that+changed+the+world&qid=1696506260&sprefix=machine+that+changed+the+world%2Caps%2C72&sr=8-1

There is another aspect that needs to be considered – organisational maturity – how many rungs are we capable of going up at a time. Legoisation is a learned skill and we are finding our feet at the bottom at the moment.

3.7 Most of this simply isn't new

All of the components and componentisation choices discussed in this paper are well known, well understood and have considerable literatures. This is about extending techniques that are tried and tested – and which in many cases are already implemented in the public sector. *You probably know more about how to do this than you think*.

3.8 Order of consideration

Government computer systems and their development fundamentally differs from private sector ones because the private sector has to seduce people to become users whilst the public sector can compel.

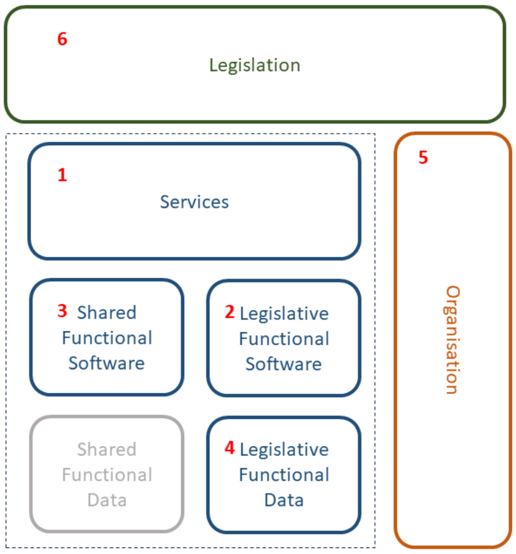
In thinking about legoisation we should start not at the top (with legislation) but in the middle (with services) and the citizen and their experience. The law should be shaped to make systems easy for citizens to use, and the technical and data architectures fitted to that world.

After services it's time to look at the software architecture and then the data architecture. The organisational architecture is how these are composed into services that can manage themselves operationally: costs, staffing, operational reporting and all the rest.

It is critical to remember that the organisational level is different to all the rest – the models and components and all the rest only exist in the minds of people. Fundamentally, unlike lego which has physical components, all of these components are "contracts" that need to be lived by human beings. Fundamentally at scale all software problems are human problems[7].

[7] hat tip to my old gaffer Mahesh Paolini-Subramanya for that zinger

Legislation is the final part – once there is an appropriate organisation and service model that meets our cost and flexibility requirements it becomes possible to make of the parliament a machine for stamping out bricks that we can build with.


6
Legislation
1
Services
3 Shared Functional Software
2 Legislative Functional Software
Shared Functional Data
4 Legislative Functional Data
5
Organisation

4 Componentisable Layers

4.1 Introduction

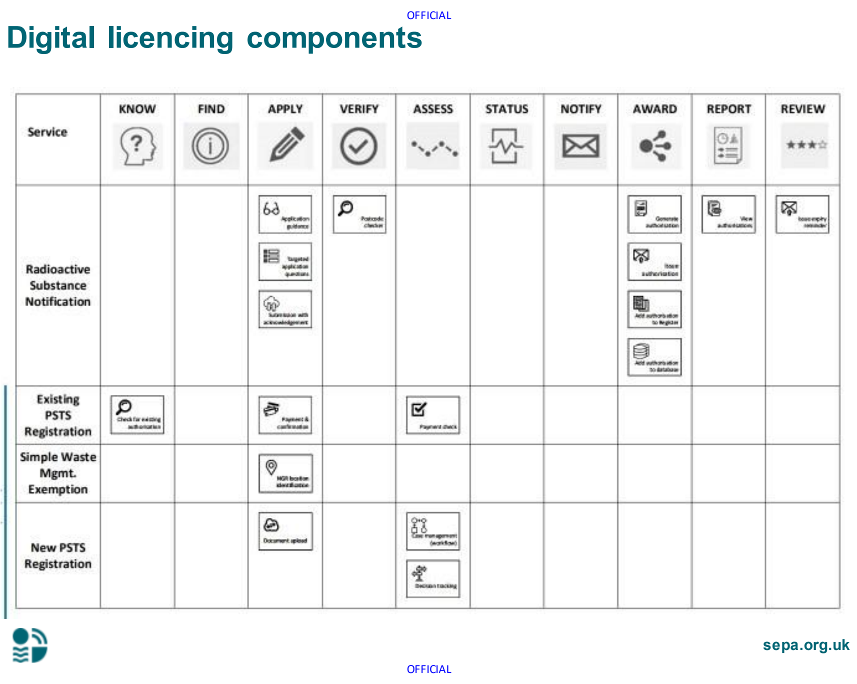
This section will step through the various layers that we can componentise and sketch out the options in each. Critically we can understand where promotion of components (say from a service to an Act of Parliament or a Ministerial Order) can pre-bake systems and reduce costs and implementation times.

4.2 Services

The Scottish Environmental Protection Agency issues thousands of licenses, in a variety of flavours to a wide range of organisations across Scotland. They have been through a couple of rounds of legoisation – building software systems and approaches that can consolidate the business processes of different licensing services whilst supporting the full range of required licenses.

In this slide from SEPA the licensing team have componentised their systems based on verbs extracted from service design – cataloguing what the user has to do to successfully apply for a license to perform some action. The software development process was then to build reusable systems that could support multiple licenses of which they have several thousands.

Their specific licences are themselves legoised – a custom license for a particular installation might be assembled from a selection of sub-components – think of a 'coffee' from a chain coffee store – *double decaff frappe hazelnut latte with oatmilk and 2 sugars* is both 'custom' and 'standardised' – back of the envelope calculation Costa has north of 1,600 variations of coffee alone – choice through assembly of components – the paradox of standardisation.


OFFICIAL

Digital licencing components

Service	KNOW	FIND	APPLY	VERIFY	ASSESS	STATUS	NOTIFY	AWARD	REPORT	REVIEW
Radioactive Substance Notification			Application guidance; Targeted application questions; Submission with acknowledgement	Postcode checker				Generate authorisation; Issue authorisation; Add authorisation to Register; Add authorisation to database	View authorisations	Issue expiry reminder
Existing PSTS Registration	Check for existing authorisation		Payment & confirmation		Payment check					
Simple Waste Mgmt. Exemption			NGR location identification							
New PSTS Registration			Document upload		Case management (workflow); Decision tracking					

sepa.org.uk

OFFICIAL

Common business processes are necessary but not sufficient for effective componentisation. They represent the functional specification of the system, but there might be non-functional requirements that prevent different things sharing the same platform.

When looking at 'Verbs' it is worth assessing the VW percentage of each verb. The verb ***Pay*** has a VW percentage of nearly 100% across all government systems that make payments (Social Security has a duty of care to release lump sum payments to vulnerable clients in a trickle, but apart from that bog standard).

Functional specifications are things like paying benefits in Social Security, or calculating tax in the Revenue – things that are specific to a particular system, that make it what it is.

Non-functional specifications are things like: must work in a browser, must have data saved to disk, must use 2-factor authentication. Things that are not specific to what the software does – more about how it does it.

Typical non-functional barriers to legoisation relate to rules around data in the widest sense. To access any police system you need to be a vetted police officer or police civilian worker, and for the police to access any non-police system they usually need a warrant from a sheriff. For good reasons you won't be seeing common platforms between police and non-police functions anytime soon.

A verb with a super-high percentage is a good candidate to be pulled out into a standalone service (as ***Pay*** already is).

There are a couple of things to look out for:

- does the presence of a handful of 99% verbs pull up the overall VW%?
- could one set of verbs with a low VW% be turned into 2 or 3 each with a much higher one?

Imagine a proposed set of (partially randomly generated) component verbs:

Verb	*Find*	*Identify*	*Overlook*	*Realise*	*Generate*	*Benefit*	*Interfere*	*Pay*
VW%	99%	99%	20%	31%	19%	23%	32%	100%

Overall it looks componentisable – but *Find* is a CRM, *Identify* is the digital ID service, *Pay* is the payments rails – pull these out as stand alone systems and you don't have much left.

Take them out and then split the candidate systems into 2 sets and you might get:

Verb	*Overlook*	*Realise*	*Generate*	*Benefit*	*Interfere*
Set 1 VW%	62%	71%	78%	N/A	N/A
Set 2 VW%	N/A	N/A	54%	65%	72%

This sort of juggling of business process is the way you design your ideal future state. Identifying where you would like to be. The next challenge is to figure out if you can, should or would get there. It is technically possible? is it legal? is it easy? do the necessary people have the capability to do it? is it cost-effective to do it? what else would doing it unlock? of all the things you could do now, is this the priority?

4.3 Legislative Functional Software

There are a range of models for legoising ***legislative functional*** software – that is to say software that implements a particular specialist requirement. The actual provision of an actual service (say social security) requires many different software solutions: the main social security system, a call centre system, HR, payroll, desktop support. This table applies to the social security system only – other shared components have their own consolidation ladder.

Rung		Description	When To Use
3	Software as a service	A configurable online platform that a particular government department can use to set up a service that conforms to a certain type	When there is legislation designed to be implementable on the government SaaS platform
2	Hosted Software	One unit of government runs a software platform and when another one wants to implement a service on it they contract with the hosting unit to build out an implementation of their service	When software components or add-ons need to be developed to support a particular new requirement
2	Open Source/Shared Software	One unit of government builds a system and publishes the software for other units of the government to use 2 flavours: • keep in synch – where each user contributes changes to the central version and the software organically grows to handle more types of government service • fork – where you start with a copy of the software and customise it into your own system	When you need to run your own instances (you have sensitive data and you can't let other state employees see it) It is possible to move Open Source/Shared software to Software As A Service if there is sensitive data but it requires secure data multi-tenanting and a hosting organisation that has the highest levels of authorisation.
1	Source of inspiration	One team has done a good job and the other team comes and studies their design processes, implementation and service model	When the two business processes are not closely enough related to make a common system worthwhile – perhaps one system is legoised already for all the Xs and the other group want to build a lego systems for all the Ys

Notice that Hosted and Open Source/Shared are at the same rung – functionally they are roughly equivalent – but there are non-functional reasons (data access, other operational requirements) that keep them separate.

The higher the VW% the higher up the rungs you can go.

We can also manage components in the business process Verb model by converting the Verbs to Participles.

Just say our analysis of the Licensing Verbs that SEPA identified ended up looking like this:

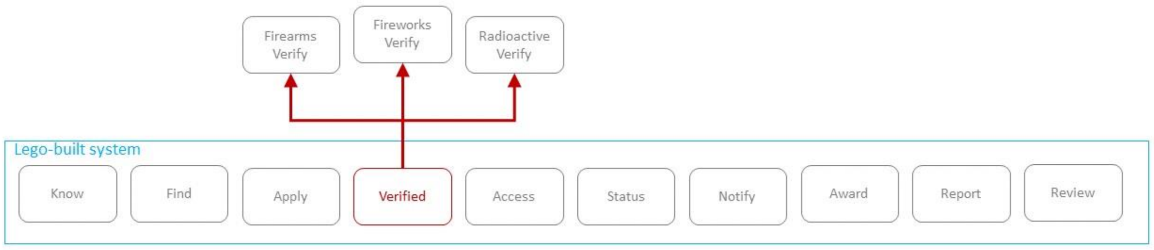
Verb	*Know*	*Find*	*Apply*	*Verify*	*Access*	*Status*	*Notify*	*Award*	*Report*	*Review*
VW%	70%	69%	72%	12%	68%	68%	71%	70%	69%	64%

So everything looks fine except ***Verify*** is madly low – Firearms require the police to verify in an interview, Fireworks needs a police record check, radioactive substances needs proof of professional qualifications, and on and on.

We can transform ***Verify*** to ***Verified*** with a fan-out architecture. The common components now look like:

Verb	*Know*	*Find*	*Apply*	*Verified*	*Access*	*Status*	*Notify*	*Award*	*Report*	*Review*
VW%	70%	69%	72%	100%	68%	68%	71%	70%	69%	64%

The core system no longer cares how you verify for a given license – it just kicks out to a system that implements the Verb for each license and the lego system just holds the result ***Verified/Not Verified***.



To use the car analogy, you might merge your production line so that you can build Audis VWs and Škodas before you are able to merge your engine lines, so maybe you have 3 of them each producing a low-power, a medium one and a high-performance, or a petrol line making a range of engines, and a diesel one likewise, or some other appropriate split out.

So you can twiddle the Verb-Participle knob to fix up things – but if you whack it up to 100% and make all verbs into participles you smash the whole of Government into thousands of fragments, each running their own tiny webservers to do minute slivers of work in an unmanageable horde.

It should also be noted that basic infrastructure is now commonly rented from major suppliers – software is run in the cloud. This is a form of componentisation (endo-legoisation within the tech world) that is now well understood and widely adopted. The key take-away about <the cloud> tho is that it is not some magic that you can rub on your software and it makes all your troubles go away (marketing hype ahoy!). Migrating to the cloud is an important first step, and an opportunity to test your organisation's maturity in component thinking but it is not the end by any means.

4.4 Shared Functional Software

Shared functional software is things like call centre software, desktops and fileservers, accounting, HR and payroll. There are a couple of reasons why we need to think about it.

Firstly our own government specific software needs to be delivered in a full operational context, staff you work on laptops and log in and take calls and send emails and get paid.

Secondly, the legoisation processes that we want to bring to government are already well advanced in normal commercial software – looking at shared functional software is a good way to get a sense of what our own delivery offerings can and should do. It is a good way to get shared understanding of what our as-yet-undelivered systems need to do by pointing at and learning from work that other sectors have already done.

In particular we can see that already software is moving into multiple layers of componentisation – from cloud provisioning, to fully service platforms-as-a-service to software-as-a-service. Things like multi-tenanting (where many customers share the same infrastructure) will all have their own futures across state systems (if they aren't already being implemented).

There already exist government guides to buying and implementing shared software components so I won't recapitulate them here. The ladder looks very much like the legislative functional one (although there is a plethora of marketing terms and fads).

4.5 Data

Data seems the easiest layer to deal with. The first problem is that it can seem trivial. It isn't. Keeping the same data held in multiple places in sync is very difficult and highly technical.

From a cost/simplicity perspective tho it is simple. Either you have the data in one place, or you have it in two or more. If you have two identical data sets with their own procedures and process and you replace them with one then you have eliminated work, and confusion and cost and all the things that we are trying to get rid of. Job done.

But merging datasets is easier to talk about than do.

To successfully merge datasets you need to align a number of operational factors.

Action	Description
Defining	Where data is defined, could be legislation, regulation or ad-hoc
Auditing	This is general looking at the data for data quality, conformance with human rights, and data protection, checking that data is not available to the wrong people, weeding and deletion activities
Appealing	The process and procedures whereby a person or organisation or thing gets onto or gets taken off the database
Partitioning	Where the data entity is partitioned, across local authorities, across health boards, internally within SG and its agencies
Creating	The point of creation - and who, how and why the data must be created
Reading	Access rights to use and see data
Updating	The processes for updating data - specifically updating by overwriting - so not ledgered/immutable data structures (ideally we want to use ledgers as much as possible and not update).
Deleting	The processes around the deletion of a data item (this is generally partial deletes of items and not total delete/weeding of a set of collated data which is covered by Auditing) (ideally we want to use ledgers as much as possible and not update).
Refreshing	Is the data once and done, or is it supposed to be up-to-date?

As a rule of thumb, for 2 data sets to be merged into one, all of these need to be either harmonised or managed. The problem comes from the fact that they are not uniformly specified. Some laws have a set of some of these in the primary legislation (Acts of Parliament) with the rest handed over to the operational implementors to decide. Some pass some or all of them over to secondary legislation (Ministerial Orders). Some legislation (primary or secondary) is handwaving, some is precise.

To make it worse, poorly described rules of managing or accessing data may have been sharpened up in case law where courts, faced with uncertainty or imprecision in the law, have adjudicated and set precedent.

The cost of working out if two data sets are or can be aligned can be expensive and uncertainty itself is a major barrier to joined-up government. Faced with time, cost, immediate impact and promotion or political pressures the default option 'go it alone' often seems to make the most short term sense, despite the long term impacts and degradation of services that brings.

Data also needs to be aligned with the Rule of Law at a design level[8].

[8] There is a BIus Working Paper that looks at this. Working Paper 1 – ***Data and the rule of law*** – and another Working Paper 5 – ***Law reform for data*** will go into the legislative side of it in more detail.

4.6 Organisation

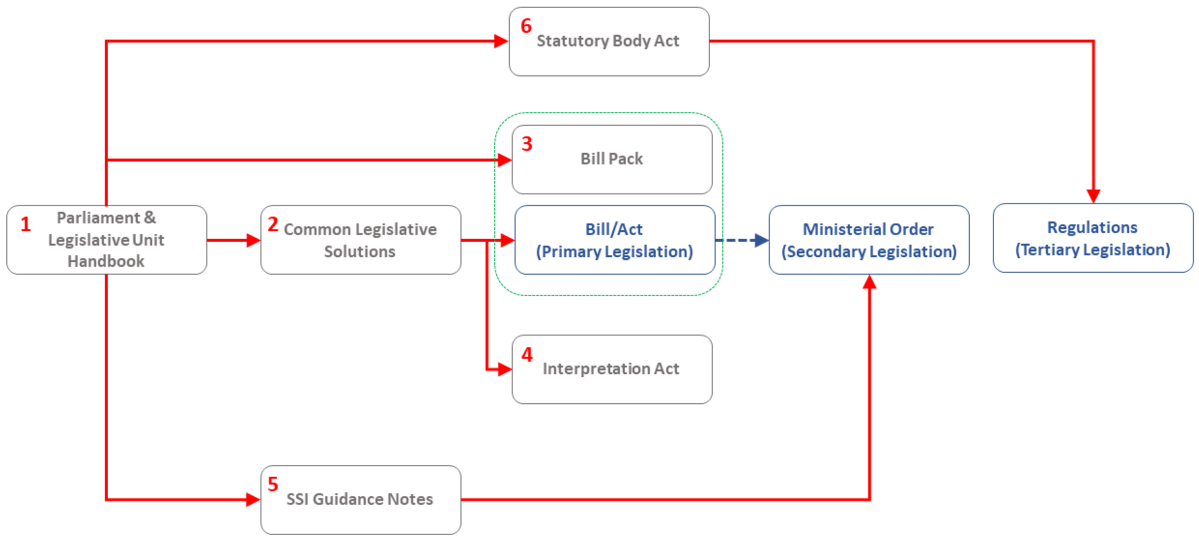
Organisational models have a similar ladder structure to technical consolidation models.

Rung		Description	When To Use
3	Umbrella Services	A single organisation provides a host of multiple services using common infrastructure like call centres, website, help desks. It controls software development and how the IT presents to employees and the world	When there is a mass of closely related services and transfers between one service and another is common (think health services)
2	Platform Services	The 'owners' of the service supply non-technical staff (citizen facing, call centre staff) and purchase or use platform services (managed software) from another government organisation or private provider	Where the services being supported are structurally similar but culturally and operationally very different (think firearms licensing and dog licensing).
1	Infrastructure Services	One government department or supplier provides raw machines and database servers in an environment that has some basic management (backup, restore, failover, electrical and physical redundancy) and the IT department of the purchasing department install and run software on it	Where there is a need for very specific and custom software environments

Organisational design here is being discussed at a surface level – how it is experienced at the boundaries. As a discipline it focusses heavily on internal structures within an organisational unit. How you build and resource your teams will shape what you deliver. You should design organisational components that encapsulate data, software and service components such that they contain ***all the necessary powers*** to do their job. Poorly placed organisational boundaries can and will kill effectiveness and performance.

4.7 Legislation

Finally, we are at legislation. Before we can discuss legoisation at this level it is important to understand the legislation assembly line. Obviously, this will vary by jurisdiction, but most (Anglo-Saxon) jurisdictions have processes that are closely related to the Scottish system.


6 Statutory Body Act
3 Bill Pack
1 Parliament & Legislative Unit Handbook
2 Common Legislative Solutions
Bill/Act (Primary Legislation)
Ministerial Order (Secondary Legislation)
Regulations (Tertiary Legislation)
4 Interpretation Act
5 SSI Guidance Notes

Lets walk through this process step by step:

	Description
1	The process of creating primary legislation (acts of parliament) is outlined in the *LPU Bill Handbook*[9]. (The Westminster equivalent is the *Guide To Making Legislation*[10]). Neither of these documents mentions technology, data, services or systems – nor prompts people working on legislation to consider how it is to be delivered at all. These processes are targets for promotion of checklist questions: “have you considered? here is a way of doing this…”
2	The bill process develops policy which is then passed to the Parliamentary Counsels to turn into a Bill – draft legislation for consideration by the Parliament – that process is supported by a pattern book *Common Legislative Solutions*[11]. It has patterns for licensing, creating bodies and a variety of other common drafting problems. It is endo-lego. Again this is a target for promotion of checklists: “write this into the legislation”.

[9] https://www.gov.scot/binaries/content/documents/govscot/publications/foi-eir-release/2022/07/foi-202200306018/documents/foi-202200306018---information-released/foi-202200306018---information-released/govscot:document/FOI%2B202200306018%2B-%2BInformation%2Breleased.pdf

[10] https://www.gov.uk/government/publications/guide-to-making-legislation

[11] https://www.gov.scot/publications/guidance-instructing-counsel-common-legislative-solutions/

	Description
3	The law itself is not the only target for promoting into. Every Bill which is introduced into parliament is done so as part of a Bill Pack of which the Bill is just a part. At this point we slide out of the world of government and the civil service and across the constitutional boundary into the world of the parliament. The 'proper form'[12] of a Bill is determined by the Parliament. The Bill Pack contains a variety of elements – impact assessments, financial statements, explanatory notes as well as the proposed law itself. Not all elements of it have the same status. Basically when resolving a point of law, a judge may take into account the Explanatory Notes and the text of the parliamentary debate in forming an opinion on the correct interpretation. Other components of the Bill Pack are not in themselves justiciable.
4	The Interpretation And Legislative Reform (Scotland) Act 2010[13] is the primary pattern book in law. It is a repository of common definitions which other acts then refer. It is in effect a published catalogue of legal lego blocks – and is a promising target for promotion into.
5	Not all law goes through the Bill/Act (Primary Legislation) route. There are only some 22 Bills per year at Holyrood – as compared to about 400 Ministerial Orders (Secondary Legislation or Scottish Statutory Instruments). The SSI Guidance Notes[14] is pattern book for them, and a promotion target in its own right.
6	The final place for a catalogue is plain old regulations (sometimes called Tertiary Legislation) where a person or organisation has statutory powers to make a determination on standards and people are legally obliged to follow those determinations. This could be a statutory body (created by an Act of Parliament and under the oversight of the Parliament and not the government) or could be a statutory person (something like a Chief Medical Officer but for tech who could say "you must use this, you must do it this way…"). (I have written more extensively on statutory persons and their role in killing duff systems over on SubStack[15]).

[12] In the Scottish Parliament's case by Standing Order 9.2 https://www.parliament.scot/about/how-parliament-works/parliament-rules-and-guidance/standing-orders/chapter-9-public-bill-procedures#topOfNav

[13] https://www.legislation.gov.uk/asp/2010/10/contents

[14] I have a personal copy of the SSI Guidance Notes – I don't have access to the Scottish Governments Electronic Document Record Management systems. As far as I have been able to determine it has not be made publicly available by the Scottish Government – I suspect not because it is confidential, but because nobody outside government is interested in it

[15] https://digitalpolicy.substack.com/p/stop-the-line

```
 __          __               _
 \ \        / /              (_)
  \ \  /\  / /_ _ _ __ _ __   _ _ __   __ _
   \ \/  \/ / _` | '__| '_ \ | | '_ \ / _` |
    \  /\  / (_| | |  | | | || | | | | (_| |
     \/  \/ \__,_|_|  |_| |_||_|_| |_|\__, |
                                       __/ |
                                      |___/
```

Before crashing into this, take a step back and think about the constitutional consequences. The clear, easy and simple route is to create framework bills and push all pattern stuff into secondary legislation and guidance. ***This route is wrong***.

Decisions about data and data handling are very long lived – with lifetimes longer than human beings. Careful thought needs to be put in to figure out how to do this.

This is a core topic of my research and is explored in this Scottish Government blog series[16]:
Part 1 – we need a gearbox (blogs.gov.scot)
Part 2 – Frankenstein Bill (blogs.gov.scot)
Part 3 – technical pattern books (blogs.gov.scot)
Part 4 – a legislative architecture (blogs.gov.scot)
Part 5 – testing the proposals (blogs.gov.scot)

The starting point is clear, the first thing to do. The LPU Bill Handbook defines a process to follow – it can be thought off as a high-powered and well-written checklist. The first step is to inject our new pattern thinking into it in the form of a section focussed on the digital implementation that the policy/legislation will need to become a thing:

- have you talked to the data people?
- what existing technical systems will be impacted by this?
- who will deliver this and are they involved in the development of the policy?

and so on.

The process of injecting these patterns must include the owners and authors of the things-into-which-we-are-injecting-them. These professionals have their own patterns and ways of seeing the world – and the reconciliation of these two views of the same pattern cannot take place without their enthusiastic participation.

Creating patterns is all well and good, but unless the policy and bill teams adopt them, use them and see value in them it will be a wasted effort. The LPU Bill Handbook is therefore a critical normative force on future legislation.

[16] https://blogs.gov.scot/digital/2023/08/28/basic-law-making-for-legislative-computer-systems-part-1/
https://blogs.gov.scot/digital/2023/09/04/basic-law-making-for-legislative-computer-systems-part-2/
https://blogs.gov.scot/digital/2023/09/11/basic-law-making-for-legislative-computer-systems-part-3/
https://blogs.gov.scot/digital/2023/09/25/basic-law-making-for-legislative-computer-systems-part-4/
https://blogs.gov.scot/digital/2023/10/02/basic-law-making-for-legislative-computer-systems-part-5/

There are a wide range of ways in which patterns and catalogues could be expressed in the legislative and policy world. The final arbitrator as to which is the best cannot be an outsider like me, nor is it obvious that currently there is anyone who can make that call.

The only sensible approach is to do this in an experimental manner with the right stakeholders and let the people using the patterns adjust the patterns and bed them in.

In that spirit I will romp through a range of different implementation patterns that might be useful. Bear in mind that different elements of the legoisation might need different implementation in different places. This list isn’t exhaustive, you should be able to come up with variants and options:

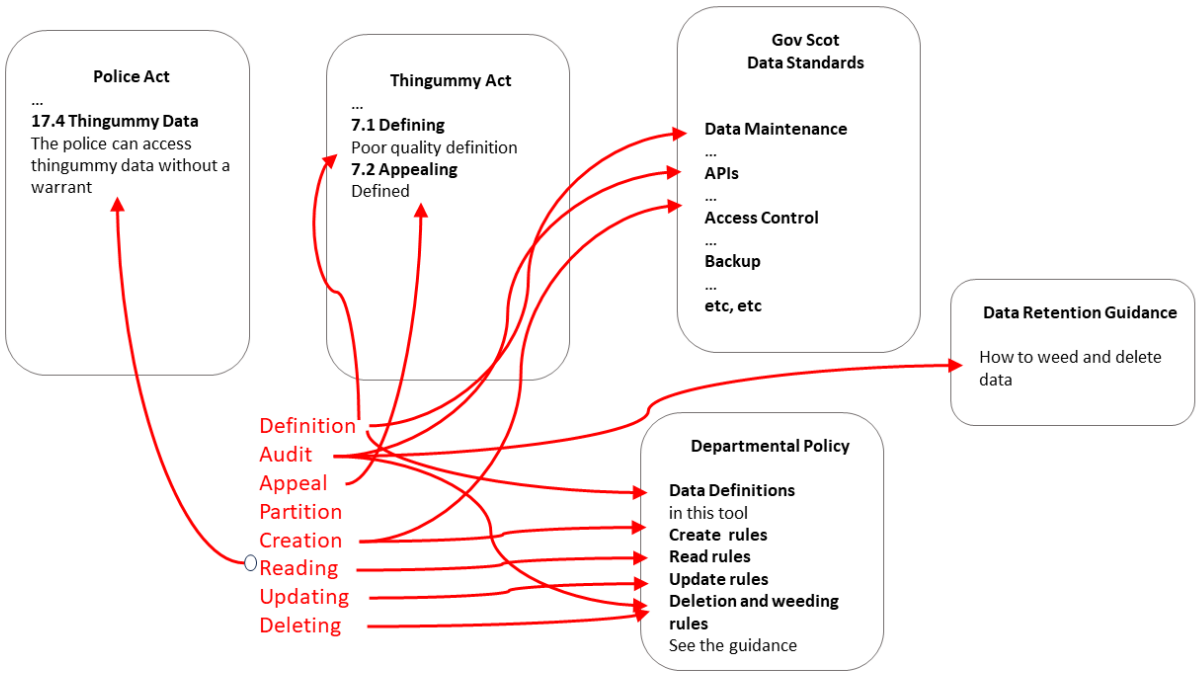
- Legislative tidy-up
- Standardisation of terms
- Drafting checklists
- Fully formed services
- Built-to-be-deployed legislative solutions
- Delivery frameworks

The data aspects of legislation will be discussed in more detail in the forthcoming Working Paper - ***Law reform for data***, and the delivery framework will similarly be considered in the forthcoming Working Paper - ***Experimental digital legislative processes***.

4.7.1 Legislative tidy-up

If we look at the things which we need to know to define data management we see that for different systems they might be smeared across the legislative and regulatory landscape.

Lets look at where the operational factors for data that were defined in Section 4.5 get applied:

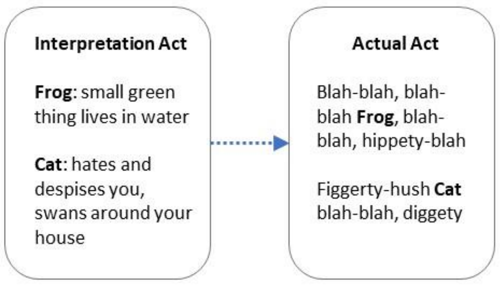


The things we need to know are smeared across multiple places (or simply not defined like partition). This is a world in which one-way is better than the best-way. Making legislation and regulation easier to reconcile with respect to data management will be a key win.

In a previous life I was Service Architect at Edinburgh City Council as part of the BT/CEC joint enterprise. I did a straw poll among my database folks asking them how many instances of people data they had in separate databases (names and addresses) and the answer was roughly 80. I asked why didn't we just migrate them to a common customer database. Well it turns out that that isn't a technical question, the problem is working out if it is legal – massive job.

4.7.2 Standardisation of terms

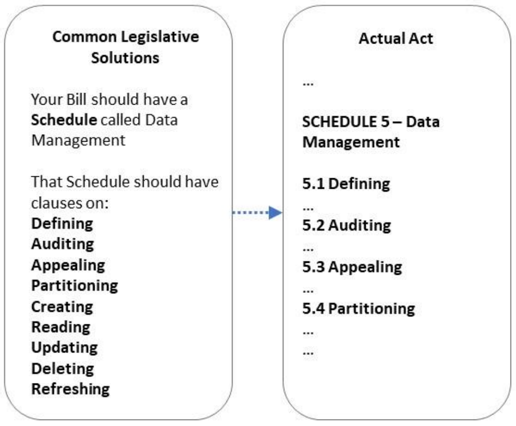
Interpretation Acts are great places for standardisation:


Interpretation Act
Frog: small green thing lives in water
Cat: hates and despises you, swans around your house
Actual Act
Blah-blah, blah-blah Frog, blah-blah, hippety-blah
Figgerty-hush Cat blah-blah, diggety

If you get to the point where components exist and there is a shared understanding of them across both government and parliament then promotion to a Interpretation Act is the final boss catalogue.

4.7.3 Drafting checklists

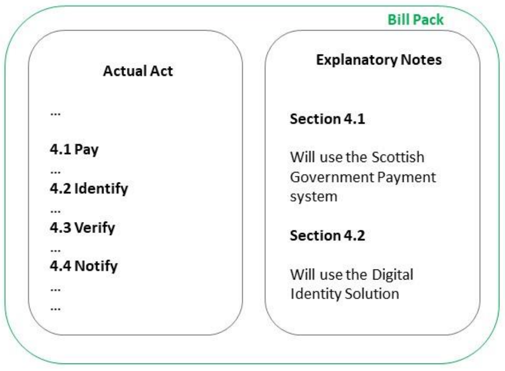
A parallel to the LPU Bill Handbook is the Parliamentary Counsels' Common Legislative Solutions. Structural patterns could be promoted into it. This would be a catalogue entry for Bill Teams – *I will have an X like wut you have in the Common Legislative Solutions*.



This is a particularly interesting example because it shows the 2 core components of exo-lego. Firstly the standardised object – here in the Common Legislative Solutions – and then the necessity to publish it. By the time it hits the Parliament Counsel for drafting a Bill process with associated policy might have been running for a year or more. The pick-me-from-a-catalogue moment will want to come as early as possible in the overall process. Again this a communications and engagement problem.

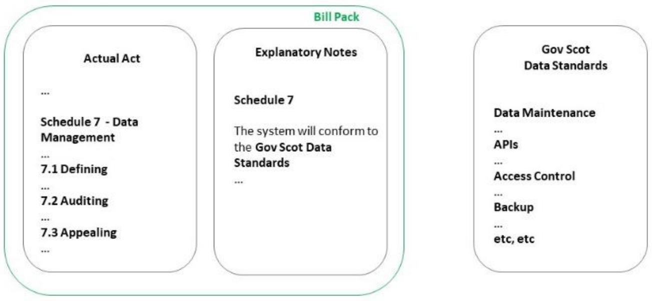
4.7.4 Fully formed services

Fully formed systems (digital identity, payment, etc, etc) can be injected either directly or indirectly into the Bill Pack. Lets look at direct injection first:



You will notice that here I am representing Verbs from the SEPA Service patterns of section 4.2 directly in legislation. And after using Service patterns to structure my legislation I am further embedding APIs-as-service-components (like Pay or Digital Identity) via the Explanatory Notes.

Remember that the Explanatory Notes are in the justiciable part of the Bill Pack. This might not be desirable. In this example of indirect injection regulations are used to embed data standards:



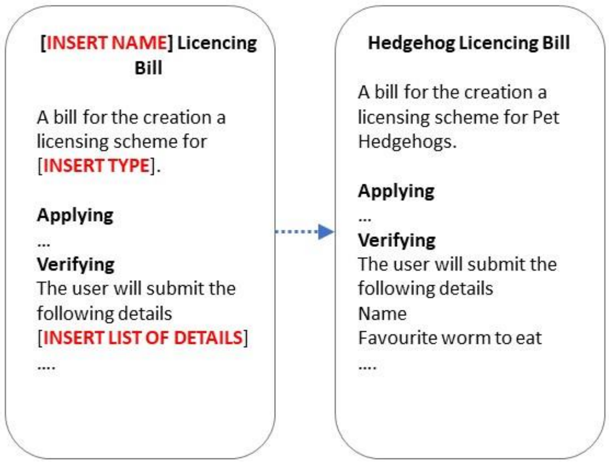
There are a couple of considerations about indirect injection:

- What is the status of the thing being injected?
- Who issues it? a statutory body, a statutory person or some general function?
- Does the issuer have the power to ensure it is adhered to?
- Is it volatile and likely to evolve and change? (best referred to indirectly) or static and unchanging (maybe better direct?)
- Is it compulsory or aspirational?

4.7.5 Built-to-be-deployed legislative solutions

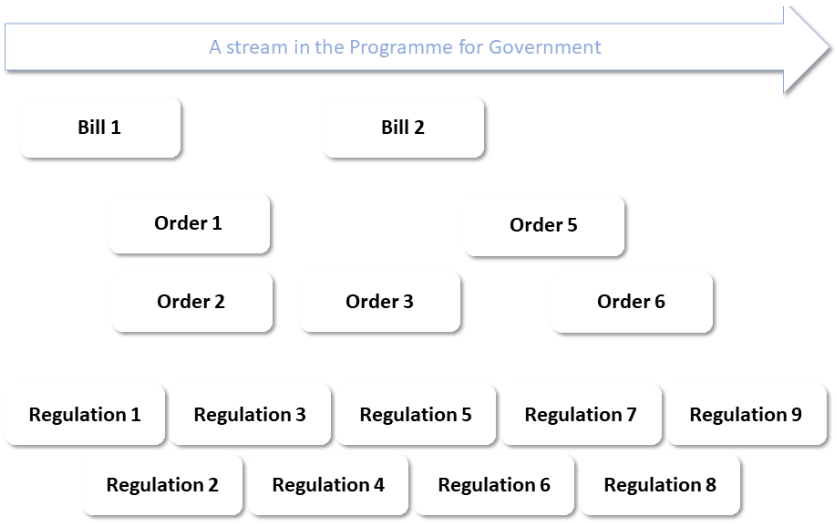
If the technical and service solution is sufficiently harmonised it might be possible to template legislation and have those templates available in the Common Legislative Solutions pattern book of the Parliamentary Counsel.

This model would work for very well defined and repeated problems (like licensing) where a target software system and associated service and organisation model exists and the government is seeking to get the authority to stamp out another one of them – licensing springs to mind:


[INSERT NAME] Licencing Bill
A bill for the creation a licensing scheme for [INSERT TYPE].
Applying
...
Verifying
The user will submit the following details
[INSERT LIST OF DETAILS]
....
Hedgehog Licencing Bill
A bill for the creation a licensing scheme for Pet Hedgehogs.
Applying
...
Verifying
The user will submit the following details
Name
Favourite worm to eat
....

4.7.6 Delivery frameworks

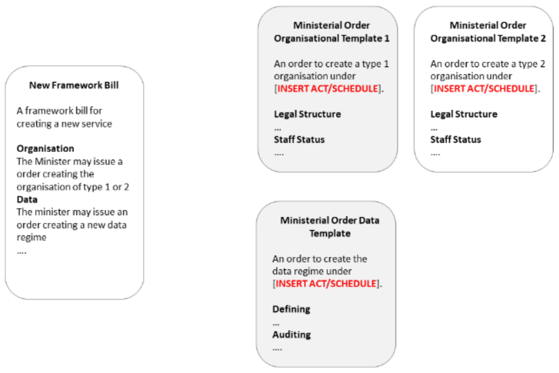
Passing an act of parliament is a point-in-time event – delivering a policy rarely is point-in-time. Something like the new Scottish Social Security system has been in ongoing delivery for over half a decade. Often secondary powers are used to time-smear legal powers over the delivery timetable. The diagram below shows different sorts of legal instruments being used to move delivery of a major programme – with time running left to right. A bill and a follow up. Ministerial powers being used to publish secondary legislation as appropriate and normal/business-as-usual regulations.



The constitutional form of Statutory Instruments (often called Ministerial Orders or secondary legislation) is that there is a full debate at the point of granting the powers – and that the powers be as a constrained as much as possible.

If the broad structure of a delivery programme is understood it might be possible to push patterns into the secondary legislation (as a way for constraining the Ministerial powers and making them more amenable to oversight.

In this model we would make of the secondary legislation a catalogue (of organisational structure, of systems use, of data rules, etc, etc).



New Framework Bill

A framework bill for creating a new service

Organisation
The Minister may issue a order creating the organisation of type 1 or 2
Data
The minister may issue an order creating a new data regime
....

Ministerial Order Organisational Template 1

An order to create a type 1 organisation under [INSERT ACT/SCHEDULE].

Legal Structure
...
Staff Status
....

Ministerial Order Organisational Template 2

An order to create a type 2 organisation under [INSERT ACT/SCHEDULE].

Legal Structure
...
Staff Status
....

Ministerial Order Data Template

An order to create the data regime under [INSERT ACT/SCHEDULE].

Defining
...
Auditing
....

This is obviously just the simplest sketch of how that might work.

Summary

This is a complex subject. And it is a subject that no one person can be expected to be an expert in every part of. But I hope that I have shown that within each profession there has been an ongoing process of standardisation and componentisation – and that between professions there have been the systematic promotion of package components from the lower to the higher.

The challenge is to put all these discrete things together – and that starts with a shared vision. This working paper aims to give the interest parties a shared language for discussing it.

It also hammers home the point that moving in the direction of a lego state is primarily ***a communications problem*** and ***a people problem***. Lots of moving parts – lots of professional and organisational boundaries to cross, lots of potential friction.

It is also important to remember that this way of thinking about components is only one side of the story – the civil servants/government/implementors side of it. There is another story which I have made no attempt to tell here – the constitutional side, the story of how parliament oversees, audits and keeps an eye on the lego state. Sufficient to the day is the evil thereof. That needs to be left to future Working Papers and publications.

There are simple and obvious first steps – getting new checkpoints into existing process definitions – making the process an explicit one and not implicit. There can be no continuous improvement without explicitly.

Remember, the outcomes will come from growing the capability, there is no short cut to results.

Good luck!