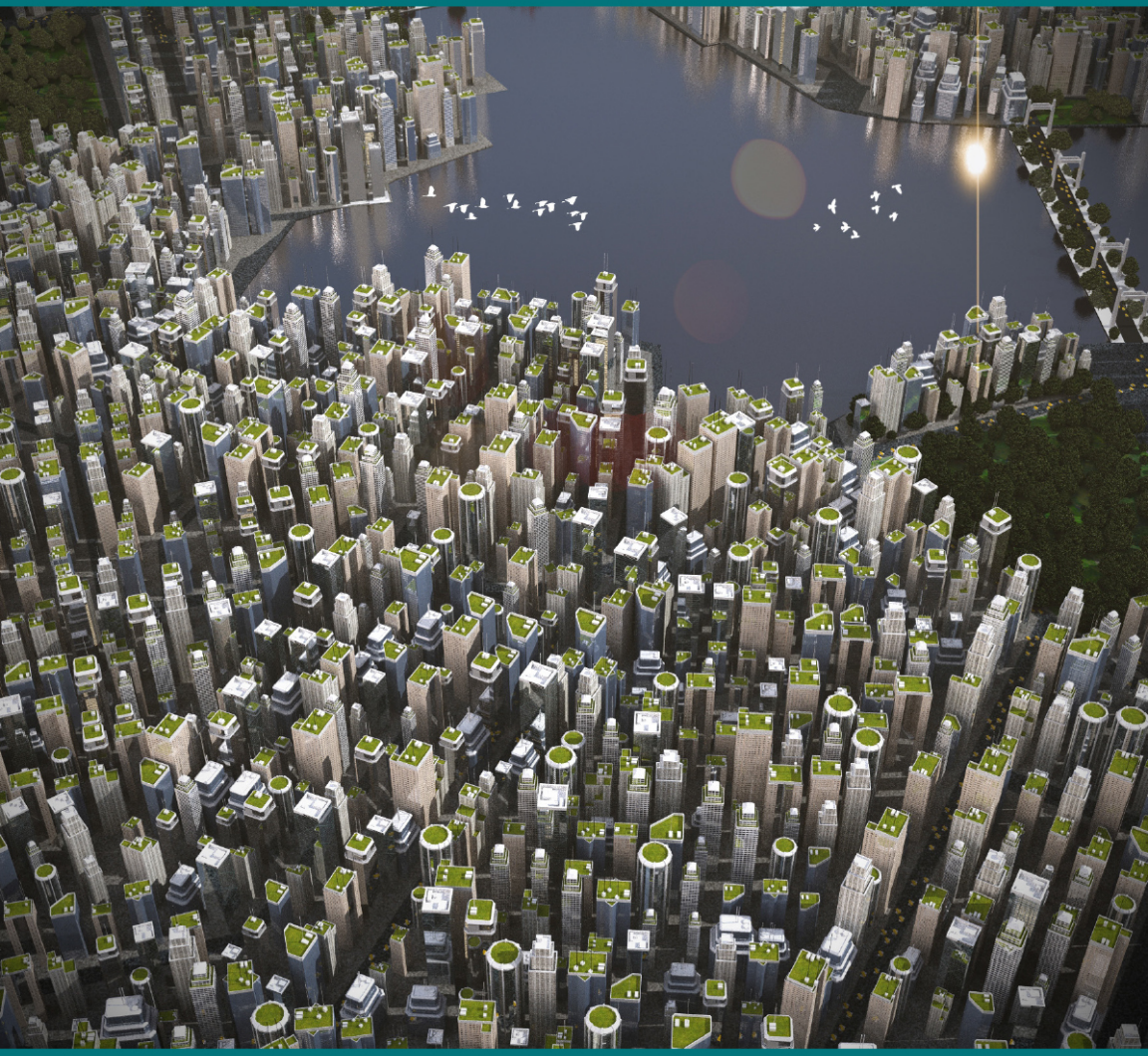
the Tetradian weblogs

Making Sense of Services in EA

Structure, design, governance and value-flow



Tom Graves

Making Sense of Services in EA

Structure, design, governance and value-flow

Tom Graves

This book is for sale at http://leanpub.com/tp-easervices

This version was published on 2022-06-20

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

SERVICES: SAMPLE

This is a sample of the content from the Tetradian ***Services and enterprise-architecture*** anthology.

This anthology from the **Tetradian weblog** provides overviews and commentary on how to make sense of services in enterprise-architecture and related fields, with an emphasis on how services need to be structured, and on the relationships between service and product.

This sample contains around one-tenth of the content from the full anthology. The complete book includes about 40 posts and 115 images from the weblog. These posts are split into three groups:

- *Services and Enterprise Canvas* - reviews the role and structure of the service-oriented Enterprise Canvas model-type.
- *Service and Product* - explores the relationships between services and product, and how those relationships work in practice.
- *Services: Other Themes* - presents further aspects of services in enterprise-architecture.

For further information on enterprise-architectures and more, visit the **Tetradian weblog** at weblog.tetradian.com[1]. The weblog currently includes some 1400 posts and more than a thousand images, and is at present the world's primary source on *whole-enterprise architecture* - methods, principles and practices for architectures that extend beyond IT to the whole enterprise.

[1] http://weblog.tetradian.com

For more ebooks and anthologies on enterprise-architecture and more, visit the **Tetradian website on Leanpub** at leanpub.com/u/tetradian[2]. (Each anthology contains around 30-40 posts from the weblog.)

Some books are also available in print format, from all regular book-retailers. For more details, see the 'Books' section on the main **Tetradian website** at tetradian.com/books/[3].

[2] https://leanpub.com/u/tetradian
[3] http://tetradian.com/books/

Services and Enterprise Canvas review - 5: Service-content

What is a service-oriented architecture - particularly at a whole-of-enterprise scope? How best could we summarise the content of each service - the elements that make up its structure? What do we need, to move beyond the IT-centrism that's dominated so much of the previous discussions around service-architectures, and to give us service-descriptions that really *can* cover the whole of the enterprise and its needs?

This is the fifth in a six-part series[4] on reviewing services and the Enterprise Canvas[5] model-type:

- 1 *Core*: what a service-oriented architecture is and does[6]
- 2 *Supplier and customer*: modelling inbound and outbound service-relationships[7]
- 3 *Guidance-services*: keeping the service on-track to need and purpose[8]
- 3A: Direction[9]
- 3B: Coordination[10]
- 3C: Validation[11]
- 3D: Investors and beneficiaries[12]

[4] http://weblog.tetradian.com/services-and-ecanvas-review-intro/
[5] http://weblog.tetradian.com/tag/enterprise-canvas/
[6] http://weblog.tetradian.com/services-and-ecanvas-review-1-core/
[7] http://weblog.tetradian.com/services-and-ecanvas-review-2-supplier-customer/
[8] http://weblog.tetradian.com/services-and-ecanvas-review-3-guidance-services/
[9] http://weblog.tetradian.com/services-and-ecanvas-review-3a-direction/
[10] http://weblog.tetradian.com/services-and-ecanvas-review-3b-coordination/
[11] http://weblog.tetradian.com/services-and-ecanvas-review-3c-validation/
[12] http://weblog.tetradian.com/services-and-ecanvas-3d-investors/

- 4 *Layers*: how we partition service-descriptions[13]
- 5 ***Service-content*: looking inside the service 'black-box'**
- 6 *Exchanges*: what passes between services, and why

In the earlier parts of the series, we saw how each service needs support from a variety of other services, to guide and validate its actions, to coordinate its actions with other services and its changes over time, and provide support from and for various forms of investment and the like. We've also seen how we can describe the services, and the architecture as a whole, in terms of a set of 'layers' of abstraction and/or realisation. What we need to move to next is more detail about the structures of the services themselves - what's inside the 'black-box' represented by each service.

(*Note*: this post deals only with what happens *inside* the 'black-box' of a service. We also need to describe what passes *between* services, of course - but we'll explore that in the next post in this series, 'Part 6: Exchanges'.)

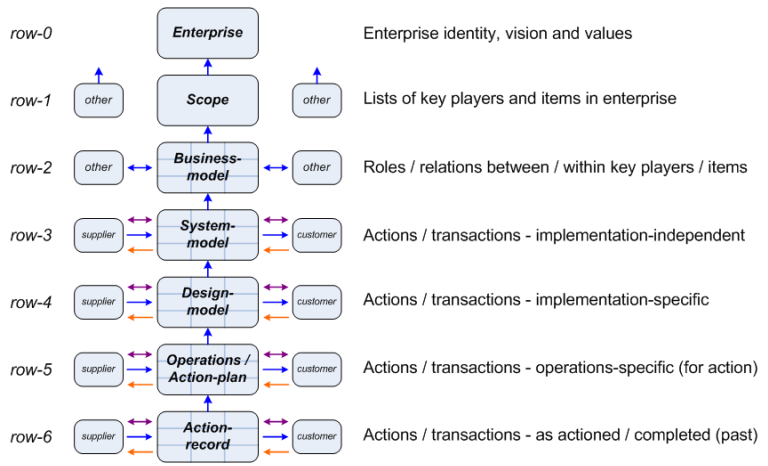
This, I'll admit, was one of the aspects of Enterprise Canvas that most gave participants some difficulty during the 'EA Masterclass[14]' series in Australia earlier this year. That's perhaps not surprising, though, because all that I gave them at first was this one diagram:

	Assets	Functions	Locations	Capabilities (action)	Capabilities (agent)	Capabilities (skill-level)	Events	Decisions	
Asset-types	*What*	*How*	*Where*	*(Who)*			*When*	*Why*	*Decision/skill-types*
Physical	Phys	Phys	Phys	Phys	Phys	Rules	Phys	Rules	Rule-based
Virtual	Virtual	Virtual	Virtual	Virtual	Virtual	Algor'm	Virtual	Algor'm	Algorithmic
Relational	Reln	Reln	Reln	Reln	Reln	Guideln	Reln	Guideln	Guidelines
Aspirational	Aspn	Aspn	Aspn	Aspn	Aspn	Princpl	Aspn	Princpl	Principle-based
Abstract			Time						

[13] http://weblog.tetradian.com/services-and-ecanvas-review-4-layers/
[14] http://weblog.tetradian.com/reflecting-on-ea-masterclass/

In practice we *do* need all of that detail - but we don't necessarily need it all at once! So this time, then, I'll take it rather more step-by-step...

Perhaps a first place to start is that, as described in the previous part of this series, about 'layers' and Enterprise Canvas[15], the service-descriptions are different at different layers of abstraction and realisation. Because of this, we're also going to need the 'layers' diagram from that post:



You'll also need to refer to **the post 'Fractals, naming and enterprise-architecture[16]'**, because that **provides definitions and explanations of the key terms** as used here.

The core point here, again, is that the content that we need to identify, collate and describe *will be different* at different layers - particularly between rows 0-3.

(Between rows 3-5, the content-*types* themselves remain much the same - it's mainly just the types and levels of *detail* that change between those layers. Row-6 'Action-Records' is different, but mainly because it's much simpler - but in essence most of it

[15] http://weblog.tetradian.com/services-and-ecanvas-review-4-layers/
[16] http://weblog.tetradian.com/fractals-naming-and-enterprise-architecture/

is a subset or description of what's happened in row-5 and in the transition through the 'travelling-NOW', and we probably don't need to deal with that here in this review.)

At the topmost layer, ***row-0*** '**Enterprise**', there *is* no service-content as such. That's because this layer represents the 'ultimate 'Why'[17]', the final 'Because.', to which everything else is anchored - or, to put it another way, it's what all of the services in the shared-enterprise ultimately aim to serve.

The content that we *do* find at this layer is anything which defines the enterprise *as* a shared-enterprise:

- *vision*, *purpose* or *promise* (and similar ultimate-anchor terms)
- derived *values* that act as the driver and anchor for the Value-Proposition and the validation-services[18]
- *principles*, *guidelines*, root-level *laws or regulations,* and other actionable guidance for the direction-services[19]

To use the ZapaMex example from the 'Layers' post in this series:

enterprise-vision as identified by ZapaMex

Remember that *this is not about the organisation alone*: the shared-enterprise has a much larger scope than that, and this layer of content is about identifying what drives the overall shared-enterprise - *not* solely the organisation or its own specific services. In the terms used in Enterprise Canvas, the organisation itself is merely one

[17] http://weblog.tetradian.com/two-kinds-of-why/
[18] http://weblog.tetradian.com/services-and-ecanvas-review-3c-validation/
[19] http://weblog.tetradian.com/services-and-ecanvas-review-3a-direction/

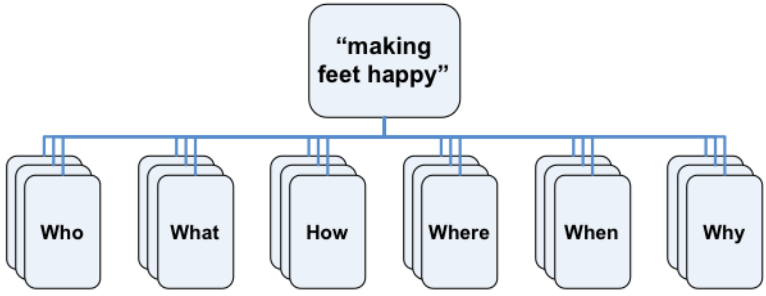
potential player amongst many within the overall shared-enterprise - see the post 'Organisation and enterprise[20]' for more detail on this.

(If we ever get this one wrong, and allow our modelling to become 'business-centric' rather than whole-enterprise holistic, we'll soon find ourselves in deep, deep trouble - architecturally-speaking, at least - without being able to understand why. You Have Been Warned, etc?)

As we move downward to ***row-1*** '**Scope**', we start to need real content about services. For here, though, all we need are lists of 'what's needed to make the enterprise happen' - and for this, the basic Zachman interrogatives will usually suffice:

What	*How*	*Where*	*Who*	*When*	*Why*

Using the respective ZapaMex example again:

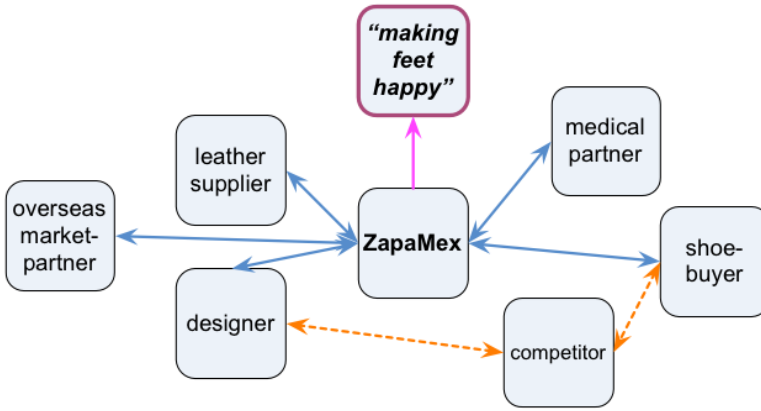


Note that these are just *lists*: we don't describe any *connections* between things at this stage, in this layer. Our organisation and its services or service-elements should appear in some way or other in most or all of these lists, of course.

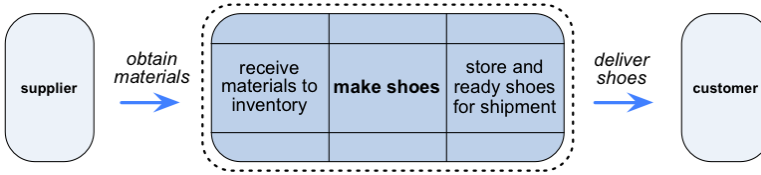
As we move down to ***row-2*** '**Business-Model**', we *do* need to start modelling the relationships between the various entities. At first this can consist of simple modelling of relative roles - self, competitor, supplier, regulator, customer and so on. And yes, given

[20] http://weblog.tetradian.com/organisation-and-enterprise/

that we *have* already established that the shared-enterprise is larger in scope than the organisation, for this purpose we *can* safely place the organisation at the centre of our modelling - as in this example for ZapaMex:



For these first stages, we'll use those lists that we derived in row-1, based on the Zachman-interrogatives. But we'll soon find, as we start to go into any depth at all, that those interrogatives alone won't be enough - even for an overly-simplistic transaction-oriented summary of ZapaMex's business-model:



At which point, the simple interrogatives...

What	*How*	*Where*	*Who*	*When*	*Why*

...need to morph into terms that are more descriptive, and capable of much more depth:

The full details for these terms are in that post 'Fractals, naming and enterprise-architecture[21]', but as a quick summary:

- **Asset** ('What') - a resource for which the enterprise acknowledges responsibility
- **Function** (external of 'How') - external-facing interface, responsible for service-contracts, protocols, SLAs, etc; accepts and returns *assets*
- **Location** ('Where') - a position within the terms of a specific schema
- **Capability** ('Who' / 'How' / 'with-What') - the ability to do something; includes the ***agent*** that enacts the capability, the ***action-type*** or *asset*-type acted upon, and the ***skill-level*** or competence of the *agent*
- **Event** ('When') - trigger for a *function* and underlying *capability*
- **Decision / Reason** ('Why') - sensemaking / decision-making for the service, and/or its type of guidance or governance

There are several points there that probably need a bit of extra explanation:

– ***There's no 'Who'***. That's not a mistake, it's a deliberate omission - and its intentional absence is meant to force us to think much more carefully about what 'Who' really means in real-world practice.

One aspect is that the 'Who-as-actor' part is sort-of subsumed into Capabilities - in particular, in the Agent, and the Skill-level of that agent. Yet note, though, that trying to equate that with 'Who' can

[21] http://weblog.tetradian.com/fractals-naming-and-enterprise-architecture/

be misleading, because although the agent is indeed sometimes a real-person, it's just as probable that the agent could be a machine, an IT-application, or even - for some types of services - something completely intangible, such as a brand. Shifting to the neutral term 'agent' allows us to to describe activities without always being forced to specify how they're implemented - an implementation-neutral stance that we certainly need in row-2 work, and most of row-3 work as well. It also allows us to explore substitutability, where one type of agent can take over the same task from another: often a fundamental need in business-process redesign, for example, where an IT-application might replace a human agent; or in disaster-recovery, where the opposite might need to occur.

The other supposedly-natural place for 'Who' would be as an Asset, in line with that usually-well-meant phrase "Our people are our greatest assets!" For me, though - and this, to me, is an absolute and non-negotiable fundamental for all enterprise-architecture - we must insist that **people are not 'assets': it is the *relationship* with the person that is the asset**. Hence the asset-dimension of 'relational-assets', as we'll see later. We link to real-people *via* relational-assets, but people *themselves* are *never* part of the architecture.

(Okay, there's the extreme special-case of the anchorite[22] - but I doubt that many people these days would have sufficient religious-fervour for their organisation's enterprise that they'd willingly present themselves to be permanently bricked-up inside the corporation's walls… :-))

– ***'Capability' isn't a Zachman-primitive***. No, it's not - and again, that's intentional. Capabilities are made up of other Services (and also assets as Products or Exchanges, as we'll see in Part 6 of this series), which in turn are composed of other services, that are themselves made up of other services, and so on, and so on, potentially ad-infinitum. The fact that Capability *isn't* a primitive

[22] http://en.wikipedia.org/wiki/Anchorite

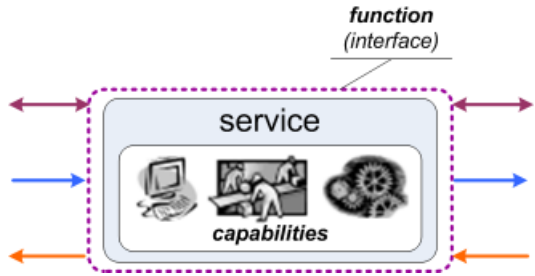
is what enables the recomposition and fractal-recursion that are so essential to a service-oriented architecture.

The other elements form sort-of-homogenous composition/aggregation-hierarchies: there are differences created by different combinations of the asset-dimensions, for example, but each of those core-types is still *fundamentally* discrete - an asset is not an event is not a function-interface is not a location, and is not *in itself* a decision or reason either. By contrast, a capability will *always* be heterogeneous, a compound-entity, a *mixture* of different-things-coming-together-to-make-things-happen - and yet that's precisely where its power resides. Without capability *as* a compound-entity, there would be no way to bring everything together *as* a service, and nothing would *happen* - which is why we need it here as an 'equal citizen' alongside the nominal Zachman-style 'primitives'.

(Another way to put this: if 'event' - for example - is a 'something's-happened' entity, and 'asset is a 'do-something-with-or-to' entity, then capability is a 'bringing-things-together' entity. Makes sense, I hope?)

– ***'Function', 'Capability' and 'Service' could all seem to be the same thing***. Yes, without sufficient care, they might indeed seem to be the same - *and that's the problem.* In many business-contexts, they *are* regarded as sort-of essentially the same thing, kind-of, but at different levels, maybe, sort-of… - and it's precisely that type of blurry confusion that the more precise terminology for Enterprise Canvas is designed to prevent.

Re-read those definitions above. Function is just an interface: on its own, it can't *do* anything at all. Capability is the ability to do something, but until it's linked with a Function (and everything else) it literally has no function. It's only when they're all linked together in and as a Service that things really start to happen - that needs can genuinely be served. As a quick visual summary:



There's more on this in other posts here, such as 'On function, capability and service[23]', 'Service, function and capability (again)[24]' and 'Function, capability and affordance[25]'.

– ***There's no 'Process'***. Yes - and that again is another deliberate omission. In part it's because, yet again, business-folks seem to have a really bad habit of using the word 'process' for what is actually a type or level of Service - in other words, creating yet more blurriness and ambiguity, which *really* doesn't help when we're trying to sort the resultant mess of miscommunications…

The reality is that, within a service-oriented architecture, process is just a way in which services are linked together into a usable sequence of some kind, to deliver some kind of broader *collective* 'service'. Because it's just a sequence, the boundary of a process is always arbitrary: the boundary is where we say it is[26] - nothing more than that. And as soon as we put a boundary around it, it becomes a type of Service, *because* it has a boundary that now needs function-interfaces and trigger-events and business-rules on that boundary. Hence we really don't need a separate term for 'process': other than that point about sequence, it's already implicit anyway in the concept and definition of a Service, and flows of Exchanges between those services.

– ***There doesn't seem to be an explicit 'How'***. You'd perhaps

[23] http://weblog.tetradian.com/on-function-capability-service/
[24] http://weblog.tetradian.com/service-function-capability-again/
[25] http://weblog.tetradian.com/function-capability-and-affordance/
[26] http://weblog.tetradian.com/what-is-boundary-of-a-service/

have to dig a bit deeper to notice this one… :-) - but if Function is only an interface with protocols and service-level agreements and suchlike, and Capability can consist of other Services but of itself is only an agent and an action-on something at a given skill-level, then where exactly is the 'How'? The answer is yes, it's true, that when we look deeper here there's no explicit 'How' in the usual sense expected in, say, a process-model - and again, that absence *is* intentional, for much the same reasons as that there's no explicit 'Who'.

The aim here is to force us, as architects, to look much more carefully at how things *actually* work in the real-world: and the reality here is that ***'How' is an emergent outcome*** - one that arises from the *interaction* between Function, Capability and Decision/Reason, and to some extent the other elements too. Look at an ISO9000[27]-type work-instruction: it's a list of action-on a given Asset, by a given agent, at a given skill-level, in accordance with a specific request from some interface, and all done in accordance with a set of applicable business-rules and assigned-responsibilities ('procedures') and policies and guidelines and validations and so on. There *is* no distinct 'How' as such: 'How' is just an emergent outcome from the ways that all of those elements interact.

The assumption that there should always be an explicit, identifiable, hardwired 'How' probably arises from too much exposure to attempts at service-implementation by automation alone - such as in poorly-thought-through business-process-reengineering and the like. In automation, yes, we generally do need to hardwire the 'How', because that's all that machines can cope with: but as soon as we need to change the agent - as ISO9000 explicitly warns us - then the 'How' inherently changes too. And in most architectures we'll *need* to allow for substitutability of agents and suchlike - at least, if we're to have an architecture that can cope with such key concerns as problem-escalation and disaster-recovery.

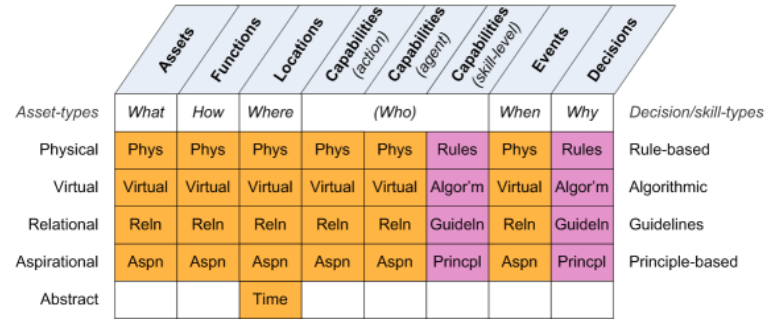
Hunting for a predefined 'How' is much the same mistake as

[27] http://en.wikipedia.org/wiki/ISO_9000

hankering too much for a predefined 'process': it's trying to apply the 'in-the-box' arbitrary boundaries of linear-thinking to a context where, in reality, everything is inherently fractal and recursive, where such 'boxes' don't actually make much sense, and where linear-thinking *really* doesn't work well at all. Instead of looking for a 'How', we need to explore more carefully the *business-need* (as specified in the Function), the ability of something to satisfy that need (as in the services-within-services of the Capability), the guidance that that 'something' would require (from the Decision elements), and the ways in which these and the other elements all interact in order to satisfy that business-need.

Yes, it's a different discipline, a different kind of rigour, than what many people might expect. But unless the architecture is *solely* about implementing fairly low-level IT- or machine-based automation - and nothing else *at all* than that - we're going to *need* that additional rigour and discipline if our architectures are to work well.

By the time we get down to ***row-3*** '**System-Model**', and onward into ever-finer implementation-detail in ***row-4*** '**Design-Model**' and ***row-5*** '**Deployment-Model**', those coarse-grained categories alone will not be enough: we need to be more specific about what each of those elements is made up of, so as to know how they go together. This is where we discover that not only are the Zachman-primitives a bit misleading for architecture - such as around 'Who' and 'How', as we've seen - but there's actually an entire *dimension* missing from that framework. Which is what brings us to this expanded visual-checklist, with which we started here:



Asset-types	What	How	Where	(Who)			When	Why	Decision/skill-types
	Assets	Functions	Locations	Capabilities (action)	Capabilities (agent)	Capabilities (skill-level)	Events	Decisions	
Physical	Phys	Phys	Phys	Phys	Phys	Rules	Phys	Rules	Rule-based
Virtual	Virtual	Virtual	Virtual	Virtual	Virtual	Algor'm	Virtual	Algor'm	Algorithmic
Relational	Reln	Reln	Reln	Reln	Reln	Guideln	Reln	Guideln	Guidelines
Aspirational	Aspn	Aspn	Aspn	Aspn	Aspn	Princpl	Aspn	Princpl	Principle-based
Abstract			Time						

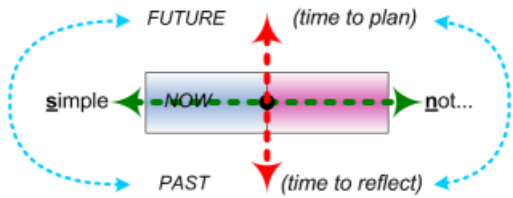
The key concern that's driving this, and also driving that movement 'up' and 'down' between the abstract and the concrete, is this:

Services are usable to the extent that they are architecturally-complete; services are *reusable* to the extent that they are architecturally-*incomplete*

By 'architecturally-complete', I mean that all of the elements for the service are in place: we know exactly what assets are or will be in play, what interfaces and protocols will be used, at what locations, using which capabilities with which agents acting on what types of assets with what skill-levels, triggered by what events, in accordance with what principles and business-rules. As the service passes through 'the travelling-NOW', the exact moment of service-delivery, all those things are and must be, for that moment, absolutely defined - otherwise service-delivery doesn't happen.

On the far side of 'the travelling-NOW' - as we move to the world of ***row-6***** 'Action-Records'** - different subsidiary-services come into play, looking *back* at that moment that's now in the past, and passing those records on to the guidance-services, the investor-beneficiaries, and the wider shared-enterprise, for review and to guide future service-delivery. Once we've moved to the past, nothing from *that* 'the NOW' can be changed - in a very literal sense, that moment has passed. But from the past, we loop back to the future - a point we can perhaps illustrate with one of the earlier

SCAN[28] crossmaps:



In practice, the role of architecture is to explore possibilities for the future, playing with the level of uncertainty - the 'incompleteness' - about elements of the architecture, in order to identify the most appropriate trade-offs and configurations both within each service and across the whole. That's *why* we need those varying levels of abstraction *about* detail, ranging from almost-nothing to the Zachman-interrogatives to the 'Asset / Function / Location / ...' set, to even deeper explorations using the ***asset-dimensions*** and ***decision/skills-dimensions***, here in rows 3, 4 and 5. It's this extra 'dimension of dimensions' that, yes, I'll admit, makes that service-content checklist-graphic seem a bit overwhelming at first: but it actually *is* necessary, and once we get the head around it, actually *is* quite easy to use.

As indicated in the checklist-graphic, the **asset-dimensions** apply to:

- asset
- function
- location
- capability::agent
- capability::action-on (that which is acted on by the capability)
- event

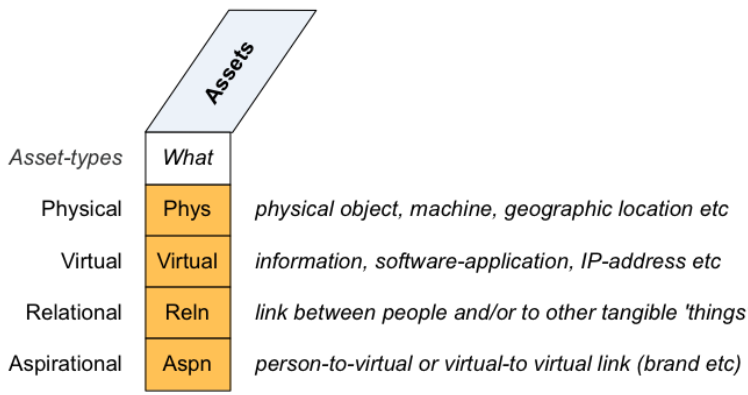
The key point here is that an *asset* - and, by reference, anything else that uses or relates to assets, as in the rest of that list just above

[28] http://weblog.tetradian.com/tag/scan/

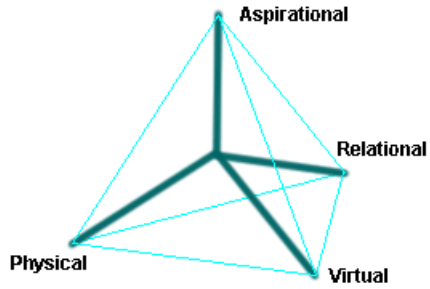
- is composed of (how best to phrase this?) four distinct clusterings, of sets of attributes or something like that, that are *fundamentally-different* and often *mutually-exclusive*. Other variations on much the same kinds of clusterings exist in other domains, such as the classic 'four states of matter' - solid, liquid, gas, plasma - or the even-more-classic Western-culture 'four elements' - physical, mental, emotional, spiritual. It's from the latter, and their implied mutual-exclusivity, that I've adapted the label-set we use in Enterprise Canvas:

- *physical*: physical object, machine, geographic-location etc
- *virtual*: data, software-application, IP-address etc
- *relational*: link between people and/or other tangible 'things'
- *aspirational*: person-to-virtual or virtual-to-virtual link (brand etc)

Which we could separate-out from the checklist-graphic as a single column, as follows:



Or, in visual form as four mutually-exclusive dimensions:



(This layout is what I've called a 'tetradian' because the mutual-exclusivity of the dimensions allows us to pack all four of them into a three-dimensional space, as the internal-axes of a tetrahedron.)

Some of the distinguishing-attributes that make up those asset-dimensions include:

- *intrinsic* (physical, virtual) vs *'between'* (relational, aspirational)
- *tangible* (physical, relational [both end-points], aspirational [one end-point, usually]) vs *non-tangible* (virtual, aspirational [other or both end-points])
- *transferrable* (physical, virtual [via copy]) vs *non-transferrable* (relational, aspirational)
- *alienable* [if I give it to you, I no longer have it] (physical) vs *non-alienable* [if I give it to you, I still have (a copy of) it] (virtual) (not-applicable to: relational, aspirational)

The reason why these dimensions and the distinguishing-attributes are so important is that they determine what we can and must (or must not) do with the respective asset. A purely-physical asset has to be handled and stored and managed in a physical way: it's unique, it's transferrable, it has mass, it occupies physical space. A purely-virtual asset has to be handled in a virtual way: it's *sort-of* transferrable, by copying, and it has no mass and occupies no space

- but we can get at it, or even store it, without giving it *some* form of non-virtual existence. (There's more on this in the posts 'Assets and services[29]' and 'Fractals, naming and enterprise-architecture[30]'.)

Every type of asset also has its own variant of CRUD[31] - Create, Read, Update, Delete. Which also illustrates, yet again, why the *relationship* is the asset, and not the person-as-asset - because whilst it should be possible to create, read, update and delete a relationship *between* people, we should *not* be able to do the same with people themselves! (Not in any normal business-process, anyway…)

In practice, most of the asset-dimension entities that we deal with in enterprise-architecture and elsewhere - assets, functions, locations, capabilities, events - actually represent some kind of combination of those dimensions. For example, consider a printed library-book:

- it's a *physical 'thing'* - so we have to manage it as a physical-asset, with storage, location-tracking, physical-maintenance and so on
- it contains *virtual information* - so we have to manage it as a virtual-asset, with possible concerns about copying and suchlike
- there are *ownership-relationships* with various people, sometimes temporary (as a library-loan), nominally also 'permanent (owned by the library) - so we need to track those relationships
- there are value-relationships or *meaning-relationships* with or for various people - so we might need to track what people feel about the book in general, or even this specific copy

Or, to give another example, the location of someone's office:

[29] http://weblog.tetradian.com/assets-and-services/
[30] http://weblog.tetradian.com/fractals-naming-and-enterprise-architecture/
[31] http://weblog.tetradian.com/crud-crude-action-acronyms/

- it has a *physical location* - so we'd need to note the physical schemas that apply, such as geographic location, building-floor, corridor-position and so on
- it may have a *virtual-location* - so we'd need to note any virtual schemas that might apply, such as room-ID, role-ID and suchlike
- it will imply various *relational-locations* - we'd need to note who uses this office at this time, or in the past, or is listed to do so in the future; and/or where each of those people sit within the organisational-architecture, or reporting-relationships, or other relational schemas
- it may imply *meaning-locations* - for example, this was where a famous scientist worked, this is where the first transistor was created, this is where the treaty was signed

And yes, all of this could easily go on to infinity… So remember always that core principle of Just Enough Detail[32]: start with the business-need in mind, and do *just enough* assessment to satisfy that need - no more, but also no less.

On now to the ***decisions/skills dimensions***, which, as the check-list-graphic indicates, applies to:

- capability::skill-level
- decision/reason

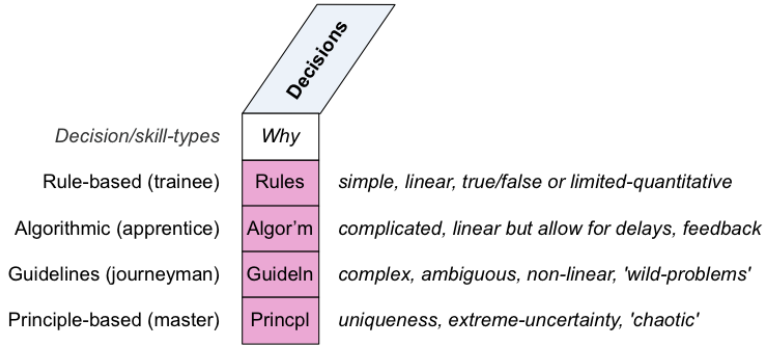
Which we can summarise as:

- *Rule-based* (roughly equivalent to *trainee* skill-level): simple, linear, true/false or limited-quantitative
- *Algorithmic* (or *apprentice*): complicated, still linear but can include delays and/or feedback-loops

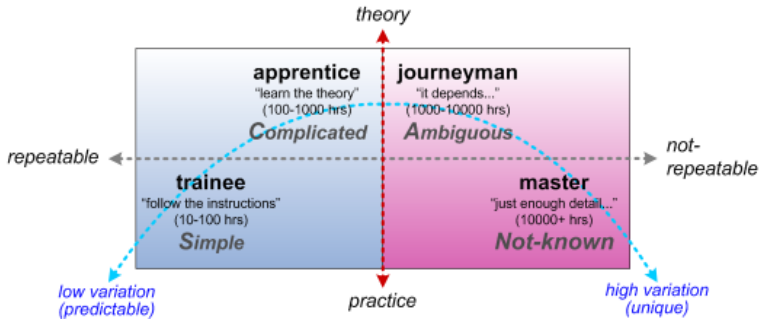
[32] http://weblog.tetradian.com/just-enough-detail/

- *Guidelines/patterns* (or *journeyman*): complex, ambiguous, non-linear, 'wild-problems[33]'
- *Principle-based* (or *master*): uniqueness, extreme-uncertainty, extreme non-linearity, 'chaotic'

Which, as with the asset-dimensions, we could separate out from the checklist-graphic as a single column, as follows:



Which we can then link to a SCAN crossmap on skill-levels, as follows:



All of which we can also loosely align with the asset-dimensions as follows (though we do need to take care to avoid inappropriate conflation here):

[33] http://weblog.tetradian.com/not-so-wicked/

- *rule-based/trainee* to *physical*-world; capabilities of physical machines as agent; can work well (though only with minimal uncertainty) at real-time speeds
- *algorithmic/apprentice* to *virtual*-world; capabilities of software-application or software-guided machine as agent; may struggle to keep up with real-time speeds
- *guidelines/journeyman* to *relational* (human) world; capabilities of human as agent, possibly with IT as decision-support and/or pattern-matching; may well struggle to keep up with real-time speeds
- *principle-based/master* to *intentional/aspirational* (high-skilled/purposive human) world; capabilities of human as agent; can work well with high-uncertainty at real-time speeds

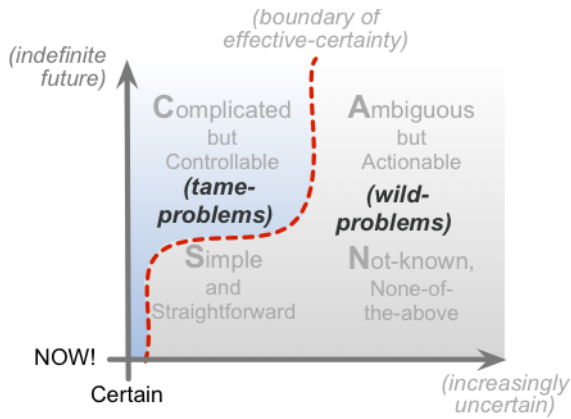
Which in turn brings us to some serious warnings about some serious architectural booby-traps, such as summarised in the post 'Learning and the limits of automation[34]'. Of these, probably the most important are:

– ***Don't try to do everything with automation***. This is such a common mistake - especially by the IT-obsessed - that this point may need to be hammered home at every possible opportunity. Yes, we can almost certainly do *part* of anything with some form of automation - sometimes quite a large part of that 'anything'. The catch is that, on its own, automation can *only* handle the parts of that 'anything' that work well with linear cause-and-effect logic - the 'trainee' and 'apprentice' levels of skill, the 'tame-problem[35]' parts that are to the left-hand side of that SCAN skills-diagram above. Yet Reality Department always - *always* - includes uncertainties and ambiguities that *will* break a linear logic: and if linear-logic is all we have, then the whole architecture will fall apart at that point, too. A whole-of-enterprise architecture cannot

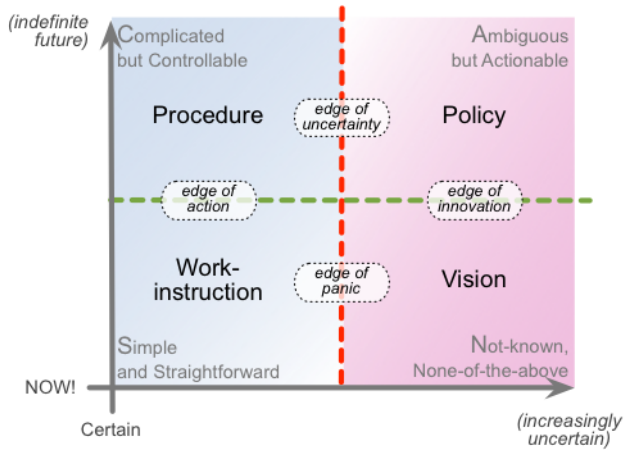
[34] http://weblog.tetradian.com/learning-and-the-limits-of-automation/
[35] http://weblog.tetradian.com/not-so-wicked/

be built successfully from automation alone: we will *always* need to fully include human elements within that overall picture. Which means that we need to include those elements *from the start* - not just as an afterthought, as happens too often in architectures at present.

– ***Provision for skills-learning is an essential part of the architecture***. This is another really common mistake - not just treating the skills needed to support the capabilities of the architecture as an afterthought, but treating the means of *learning* those skills as an afterthought as well. Compare that SCAN skills-diagram above with this SCAN crossmap of the effective limits of linear 'tame-problem' tactics:



And perhaps also this SCAN crossmap with ISO9000 levels, and the effective applicability and boundaries of each level:



The crucial point to note here is that, *at the moment of action* - the 'NOW!' - all we have immediately available to us are simple work-instructions, or the actionable principles that devolve from the shared-enterprise's vision. In other words, the trainee, and the master. And the trainee - or the automation - can only handle the work-instruction-friendly tame-problem parts of the context: as soon as they hit anything more uncertain than that, they'll get thrown over 'the edge of panic' into the Not-known domain, where they'll *need* a master's skills to sort out the mess.

But where do those master-level skills come from? The answer is: the long way round, from trainee up to apprentice to journeyman and back down again to master, through many, many hours of analysis and experiment, much of it necessarily away from the real-time 'NOW!'. And while those people are *doing* that apprentice-level and journeyman-level skills-development, *they cannot be on 'the front line'* - often because at those stages they still only know enough to get into real trouble, but not enough to get out of it again. The reality is that most people learn by making mistakes[36], and then learning *from* those mistakes: and that means that we'll often need to place them in a safely-circumscribed 'safe-fail' version of the

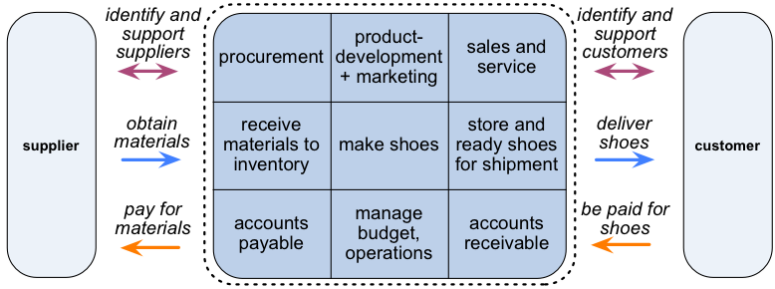
[36] http://weblog.tetradian.com/fail-to-learn/

real-world context - because real-world practice usually needs to be as 'fail-safe' as we can make it.

So if our architecture - and our organisation - demands and expects that people will and must always be 'at maximum efficiency', 'fully productive, 100% of the time', we would in effect block off the path via which people can develop their skills beyond trainee-level. What we'd have then is an architecture that looks amazingly productive and efficient in the *short*-term, but is *guaranteed* to fail in the medium- to longer-term, as those people with the master-level skills retire or move on, and there's no-one with the right kind of context-specific skills available to replace them. In short, *a full architecture must include explicit provision for development of the skills needed to maintain the capabilities of that architecture* - and that includes the provision of the necessary 'time off' and 'safe-space' within which to develop those skills.

Anyway, time to bring all of the above together, and wrap-up this part of the series.

Let's look, for example, at the row-3 ZapaMex business-model, from the previous part of this series:



The organisation as a whole represents a service which - as we can see above - is itself made up of other services.

Each of those nine cells within the business-model represents a service, each of which we'd probably describe in terms of a set of subsidiary-services.

From ZapaMex's perspective, everyone (or everything) in a 'supplier' or 'customer' role relative to ZapaMex and its services also represents a cluster of services with which it interacts.

Each of the flows on the business-model diagram above - explicit, as shown as an arrow, or implicit, at the borders between each of the services - itself implies a service at each end of that respective flow, to manage the interactions and exchanges required for that flow to happen.

So, for *each* of those services indicated or implied above:

- What *Assets* does that service need, in order to do its task?
- What *Function*-interfaces does that service need, using what protocols, with what service-level agreements, and passing which Assets (as Exchanges between the respective services)?
- At what *Locations* does the activity for that service take place?
- What Capabilities are needed to make the required actions happen for that service, with which agents, acting on what types of Asset, with what skill-levels?
- What Events occur, or need to occur, for that service, in order to signal the requisite transitions of state or action?
- What Decisions/Reasons - business-rules and more - should apply for that service?

Assess each of those in terms of the respective combinations of asset-dimensions and/or decision/skills dimensions.

If it sounds like a lot of work to do, well, yes, it is - but it's no more work than you'd do anyway, to do the architecture properly.

If it sounds 'complicated' (as someone complained about the 'layers' post in this series), well, yes, it probably is, at first - but no more so than for anything else that's somewhat new to you when you first work with it. In practice, it's actually a lot *less* complicated than trying to sort out the mess created from trying to link together

the largely-incompatible muddle of single-function, single-domain frameworks that currently litter so much of the enterprise-architecture space.

And remember, again, that so-essential guideline of **Just Enough Detail**. Don't try to 'boil the ocean': do only that amount of service-modelling that you must do to satisfy the business-need - no more, but also no less.

(I did promise, in response to a comment[37] from Anders Danielsson in the 'Investors[38]' post in this series, that I'd put in a real-world example here, about people and the role of relational-assets in a hairdresser's business. But this post really is too long already, and it's probably best left anyway until after the exploration of Exchanges in the next part of this series. I *will* use it as a worked-example in the 'summary and wrap-up' post, after the series ends - that's acceptable for now, I trust?)

Anyway, just one part left in this series - about Exchanges, the content of flows between services. Let's move on to that now, before wrapping up the series as a whole.

Source (Tetradian weblog)

- *Date*: 2014/10/27
- *URL*: services-and-ecanvas-review-5-service-content[39]
- *Comments*: (none)
- *Categories*: Business, Complexity / Structure, Enterprise architecture
- *Tags*: complexity, enterprise, Enterprise architecture, enterprise canvas, service-oriented architecture, service-oriented enterprise, services

[37] http://weblog.tetradian.com/services-and-ecanvas-3d-investors/#comment-7232
[38] http://weblog.tetradian.com/services-and-ecanvas-3d-investors/
[39] http://weblog.tetradian.com/services-and-ecanvas-review-5-service-content

Service, product, service - a novel concept?

Service and product as different views into the same space - is that a novel concept? If so, what does that imply for enterprise-architecture and the like?

The next post in this series on the relationship between product and service[40] was going to be about transfers of responsibility in a service/product chain - and I promise that I *will* get back to that as soon as I can! But the previous post in the series, 'Service, product, service - implications for architectures[41]', triggered off such a storm on LinkedIn[42] - well over 6000 views so far, and more than a hundred comments. Some of the conversations there - particularly those with Nick Malik, Jan van Bon, Kevin Smith and Jordan Julien - are definitely worth preserving in more permanent form, because they do help to clarify and resolve some of the confusions that can arise. So I'll do a brief side-series here, focussing on each of those four comment-threads, before going on with that exploration on responsibility. I'll edit some of my replies from there for brevity and/or clarity, but essentially the questions and responses in each case are the same as in the original LinkedIn post linked above.

We'll start with two important points from Nick Malik[43], the first which was laid out as a kind of book-end at each end of his comment:

> "I found the idea intriguing that a product is a bridge

[40] http://weblog.tetradian.com/2020/09/27/service-product-service-simplified/

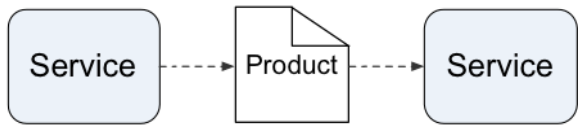
[41] http://weblog.tetradian.com/2020/10/02/service-product-service-implications-for-architectures/

[42] https://www.linkedin.com/feed/update/urn:li:activity:6717873004052905984/

[43] https://www.linkedin.com/in/nickmalik/

> from one service to another. … Your recognition of the product as a connector of one service to another is novel."

I'll admit that his description of that as 'novel' was a bit of a shock for me, because it implies there's a crucially important point that I'd missed. I'd always thought of "product as a connector of one service to another" as obvious and self-evident - as in this nearly-decade-old diagram of mine that I'd referenced in my post:



But if Nick is right - and a lot of the comments on LinkedIn and elsewhere now do seem to suggest that he is - then my failure to realise that it *wasn't* obvious was a huge mistake for me to make. From what Nick was saying there, I need to go back and do a full step-by-step reasoning on that part first - otherwise much of what I've been describing about the implications of the linkage would perhaps never make sense to others. Oops… If anyone has any suggestions as to a best way to do that step-by-step, please let me know?

Nick's other point - or rather, my response to it - may take a bit more explaining. This part of his comment (which I've shortened a bit here) was:

> The iPod changed all that [mess of MP3 players and loader-apps] with iTunes. Because the iPod had a service at one end. Now we have iTunes and the iPhone and we can put a service at the other end as well, with our apps. We no longer think of the iPhone as a product. It's part of the service of connected voice/video/internet/community.

In terms of what I was saying in the post, this isn't quite correct - or rather, *it depends where we choose to draw the boundaries*. If we draw the boundary as "the service of connected voice/video/internet/community", then yes, the iPod or iPhone would be seen as part of that service. But if we look at the iPhone *itself*, it's a product: a static container for services. And the apps on that iPhone are products too - and sold as such, as well. It's only when the app is *in use* that it becomes an *active* container for services. The services run *through* the product that is the app and, as a container, the iPhone. Even the data that pass into and out of the app and action the services are themselves products - literally, 'that which is produced', and then consumed, in a service. But whenever we draw a service-boundary, we render invisible any products within it - they would be seen merely as 'part of the service'. Which is why an iPhone seems to be just part of "the service of connected voice/video/internet/community" - but to make sense of what's really going on, we *also* need to understand it *as* a product in its own right.

Product is service is product is service… It's tricky…

That's why I'd argue that the only way we can make sense of this is understand that product isn't something separate and distinct from service: **the whole notion of 'product' is an *artefact* of how *we choose* to draw service-boundaries**.

And likewise, '**service' is an *artefact* of how *we choose* to draw boundaries across the overall flow of value undergoing change** - with those boundaries most often arising from how *we* perceive apparent gaps in time or place, or apparent change in responsibility.

To be blunt, we need to be a bit more honest about those boundaries, and how *we choose* to draw them… But that's the topic for the next post in the main part of this series: for now, back to those comments on that previous post, with a sub-thread on the structure of a service.

Source (Tetradian weblog)

- *Date*: 2020/10/18
- *URL*: service-product-service-a-novel-concept[44]
- *Comments*: 6
- *Categories*: Business, Complexity / Structure, Enterprise architecture
- *Tags*: boundary, complexity, Enterprise architecture, product, sense-making, service, service-oriented enterprise

[44] http://weblog.tetradian.com/service-product-service-a-novel-concept

Service-design: How long is a service?

It was my grandmother's 80th birthday. My parents wanted to make it a special occasion for her and for the family, so they booked us in for a meal at a place called Le Talbooth[45] - then, as now, a decidedly upmarket restaurant beside the River Stour, on the border between Essex and Suffolk counties in south-east England.

Back in those days, Le Talbooth only did a fixed menu, with the only choice between meat or fish, and with everything booked in advance. But one of our extended-family was vegetarian - at that time still regarded as a bit unusual, even a bit eccentric - so we had to warn the restaurant that we needed a different dish in her case.

So come the day, there we all were, in the grand private room of the restaurant. After the starters, out came the main course - meat or fish, as per the pre-agreed plan. And out came one of the serving-

[45] http://www.milsomhotels.com/le-talbooth/

staff, who asked, somewhat diffidently, "I understand one of your party is vegetarian?" She duly raised a hand, awaiting the usual tired omelette or baked-potato that was often all that a vegetarian could expect back then. Oh well.

But that wasn't what came up in this case. Instead, the kitchen had gone wild. They'd gone crazy. They'd obviously had a lot of fun. Because what was wheeled out, with some pride and ceremony, was an entire *trolley-load* of vegetarian treats - much of it food that most of us had never seen or heard of before, let alone tasted. Wow! I think every one of us turned vegetarian on the spot! - for the day, at least!

That was quite a long time ago - more than *four decades ago* now, in fact. But the memory of superb service from that restaurant still lives on, even that far back into the past.

Which, by a roundabout route, brings us to a key question for service-designers: **how long does a service-instance last?** What's its actual duration?

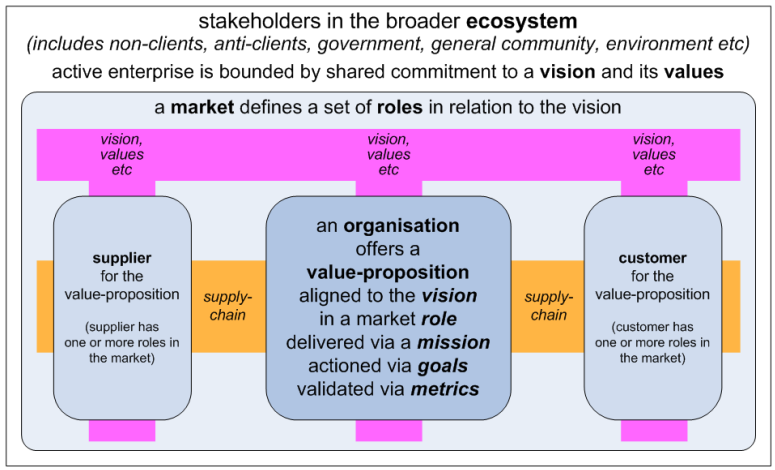
The usual view of service-instances is organisation-centric:

- the service-instance *begins* when the customer comes into the organisation's space, with the initial service-request
- the service-instance *ends* when the customer leaves the organisation's space

But as with all business-centrism[46] in business-architecture and enterprise-architecture, that view risks missing the point here - and often dangerously so. An organisation's value-proposition[47] isn't solely about its products or services: far more, it's about how those products or services can help customers, providers, employees and everyone else to reach towards a *shared-story*[48] - 'the enterprise' not as an organisation, but as a literal 'bold endeavour'[49], shared

[46] http://weblog.tetradian.com/2012/02/03/it-centrism-business-centrism-bizarch/
[47] http://weblog.tetradian.com/2013/01/23/what-is-a-value-proposition/
[48] http://weblog.tetradian.com/2010/01/26/the-enterprise-is-the-story/
[49] http://weblog.tetradian.com/2014/09/18/organisation-and-enterprise/

with all others in that story's space. Services are thus a means to reach towards satisfying the drivers for the story:



On the surface, and for the organisation, yes, service-delivery might appear only as the 'horizontal' supply-chain path on that diagram above, with the service-instance starting and ending as it transits the boundaries and timelines of the organisation itself. In effect, that's service viewed solely as 'how' and 'with-what'.

Yet the *drivers* for the service - the 'why' for service - are actually in the shared *'vertical'* links between each party and the same overarching story, as on that diagram above. That's how the connection is made: that's where marketing and branding sit, for example, and why branding is so strongly oriented towards *story* - towards creating a meaningful story about a means to satisfy a perceived or actual need.

And it's in that 'why' that we also find the criteria for success in service-delivery. We see it especially in experiences such as in our family's story above, where the expectations of service were not only met, but far exceeded, in ways that - as in our case - may literally ring on down through the generations.

We could probably say much the same, if in less happy ways, about *disservices*[50] - supposed 'services' that fail to deliver to the actual need. For example, consider this (mildly-redacted) item that came up from a US colleague in my Twitter-feed this morning:

> Anybody else get [restaurant-chain's] sticker shock yet? Monster price hike for take out. 2 dishes and appetizer $42. Seriously? It's not that good.

Again, the key here is that certain expectations were set, by the organisation's engagement in that story - and those expectations were not met. *Definitely* not met. Okay, the disgruntlement there may not sit for decades, as did our contrasting enjoyment of that family-occasion. Yet it still represents a real kurtosis-risk[51], and one that could well destroy the organisation's business[52] if left to repeat too often or fester too long - especially in these days of social-media, where a single anticlient[53]-Tweet can easily echo halfway round the world…

So to make sense of the true life-cycle of service-instances, we need to shift our view of services themselves, from an organisation-centric perspective, to the perspective of the service-customer.

(Technically the correct term is 'service-consumer', of course - but I *strongly* object to that term ever being applied to real people!)

In which case, if we use this example above, of my grandmother's birthday-dinner at Le Talbooth, then:

- **The service-instance *begins* when the customer first engages in the story**. In this case, that occurred when my parents first decided that they'd like to celebrate that anniversary - which would have been quite some while before they even thought of contacting Le Talbooth to enquire about a booking.

[50] http://weblog.tetradian.com/2015/06/28/services-and-disservices-1-introduction/
[51] http://weblog.tetradian.com/2015/06/15/hidden-risks-in-business-model-design/
[52] http://weblog.tetradian.com/2014/04/07/playing-pass-the-grenade/
[53] http://weblog.tetradian.com/2013/05/05/anticlients-are-antibodies/

- **The service-instance *ends* only when no-one remembers it any more**. In this case, *that service-instance is still continuing today* - more than forty years later - *because we still remember it.*

In effect, the quality of service-delivery continues to be re-assessed, indefinitely, until it is finally forgotten. It's that factor that drives anticlient-impacts and the like - or, on the positive side, that likewise underlies more directly-measurable impacts such as Net Promoter Score[54].

Try it: what difference does it make for service-design if you shift the perspective from organisation-centric to customer-centric? Let me know in the comments, perhaps?

Source (Tetradian weblog)

- *Date*: 2016/03/14
- *URL*: service-design-how-long-is-a-service[55]
- *Comments*: (none)
- *Categories*: Business, Enterprise architecture
- *Tags*: Business, business architecture, disservice, enterprise, Enterprise architecture, Le Talbooth, service, service-delivery, service-design

[54] https://en.wikipedia.org/wiki/Net_Promoter
[55] http://weblog.tetradian.com/service-design-how-long-is-a-service

Product, service and trust

Sirens blaring, blue lights flashing, the large white truck howls across the junction just ahead of us. Unusual markings, too: 'Bomb Disposal'.

In these troubled times it could be anything, of course; but out here, in the quiet backwaters of eastern England, it's most likely that it's that workers on a building-site have come across yet another all-too-live product from the Second World War. Yet another incompleted service-delivery, we might say…

Almost seventy years after that war ended, those incidents are still all too common here. The same applies to the incompleted service-deliveries of every other modern war: there was a lethal example in Belgium[56] the other day - a left-over from an even earlier European war - whilst abandoned landmines and cluster-bombs are an ever-present danger for children and others[57] on battlefields and farmlands alike across all too many parts of the world.

Bombs and shells and mines are the products of war, or for war. Yet the point of any product - even this kind of product - is not the product itself: it's the *service* that matters, the active *delivery* of the desired outcome.

And whatever we might think of modern weaponry, its purpose or promise as a product is that it will deliver on its service of damage and destruction *at the required time*. Yet the literally-painful fact now is that much of it failed to deliver on its promise *at the required time*: for example, some 30% of munitions from the First World War

[56] http://www.bbc.co.uk/news/world-europe-26654314
[57] http://www.maginternational.org/

failed to explode, and the reliability-record in the Second World War wasn't much better. And the reason *why* it's a literally-painful fact is that much of that remainder is still capable of delivering on its designed-promise now. And does - even though it's very much no longer 'the required time': that's the problem… Hence the bomb-disposal truck, speeding on its way through the town, delivering on *its* service of preventing that service-delivery from happening…

All of which is perhaps a rather roundabout way of introducing a theme that I think of as fundamental to business-models, operational-models, business-architecture and enterprise-architecture: **the relationships between product, service and trust**.

In more general terms (and, we would hope, preferably non-lethal too), we could summarise the above as:

A product represents the promise of future delivery of (self)-service, via use of that product

Hence, for example, if we buy a lawnmower, what we're actually buying it *for* is the future service-delivery of a mown lawn - the service itself delivered, via that product, *by* us, by some other family-member, by some neighbourhood teenager or whoever. Alternatively, we could buy just the *service* of the delivery of a mown lawns, from a lawn-mowing service: in principle, there's no difference. In practice, of course, there usually *is* a relevant difference - such as that there's no lawn-mowing service available, or we regard doing it ourselves as cheaper or more satisfying or convenient[58], or we just like the status-feeling of 'owning' the lawnmower or suchlike. Or, in other words, a difference of perceived-effectiveness[59] of service-delivery.

Importantly, though:

When we obtain a product, we trust that that product will deliver on its promise of service-delivery at the required time

[58] http://weblog.tetradian.com/efficient-effective-convenient/
[59] http://weblog.tetradian.com/efficient-versus-effective/

For example, an insurance-product is a promise of future support or recompense if some kind of 'insurable loss' were to eventuate. But the key point there is the promise, and trust in the identifiable future delivery upon that promise: because if the promised service *isn't* delivered - such as when the insurance-provider tries to back out of their promise, or to 'adjust' the nominal loss in their favour - then that purported promise feels more like a betrayal, a lie. And *nothing* destroys trust quicker than perceived-betrayal. It's also the fastest way that an organisation can create anticlients[60] against itself: not wise…

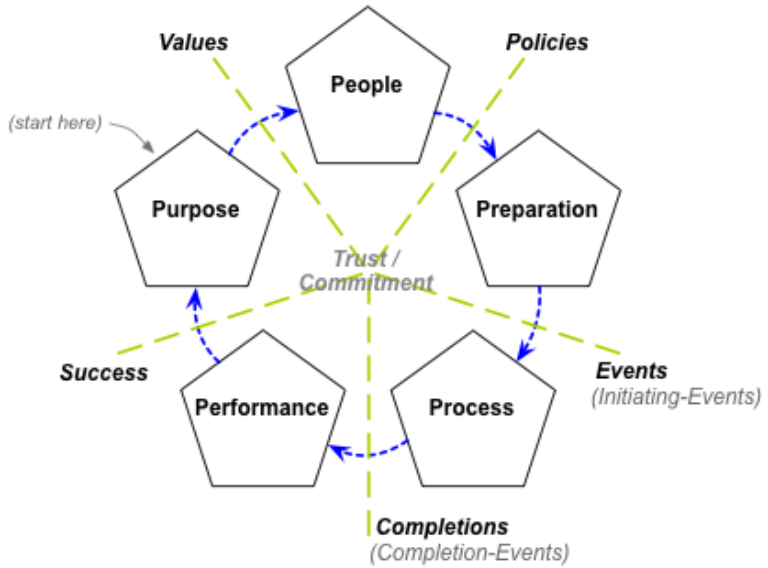
Trust is at the centre of *everything* in any enterprise - *every* product, *every* service. Without trust - or at least in proxy form, as reputation - nothing is going to start; and when it's lost, everything stops.

It takes a lot of work to build trust - yet often only seconds, or less, to lose it. And once lost, regaining trust is often harder - sometimes *much* harder - than building it from scratch.

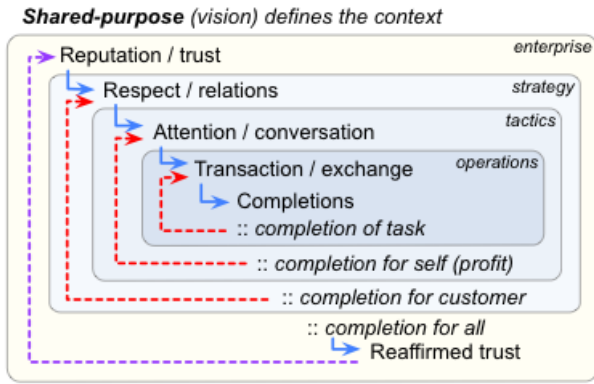
Hence, unsurprisingly, organisations really *do* need to monitor trust, in every aspect of their operations and beyond, inside, outside, above, below and all round. Which is a lot trickier than it might at first seem, because 'information *about* trust' - such as we might hold in an IT-system, or display on an 'executive-dashboard' - is *not* the same as as trust *itself*: yet it's the latter - not the former - that we most need to track.

So, for example, in what ways do each and every one of our activities and services reinforce trust, or place trust at risk? At a first level, we could map this in terms of the Five Elements sequence-like set that underpins much of my own work on enterprise-architectures:

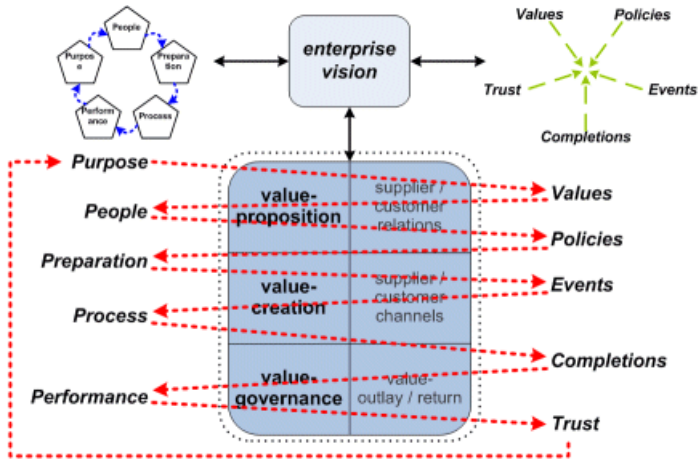
[60] http://weblog.tetradian.com/how-anticlients-happen/



And we could map the overall flow of trust, and support of and reaffirmation of trust, within the service-delivery cycles of each service, using a model such as the Service Cycle:



From there, we can map that pair of interaction-sets across the flows and sub-tasks of service-delivery, from 'inside' to 'outside' and back again, again supporting and reaffirming trust:



For enterprise-architects and business-architects, tracking these threads and themes and interactions *should* be a key part of enterprise-modelling and business-modelling. It worries me a lot that I don't see many EA-folks doing so as yet… Another related theme is waste:

Waste is a product whose services are no longer required and/or are no longer trusted

And/or:

Waste occurs when the inherent promise of products or future services is lost

Both of those should be self-evident when we look at 'waste' in terms of 'lean manufacturing[61]' and suchlike. Yet think of that latter example also in a human sense: the waste and loss of hope implied when a life is damaged or cut short before its time… When such promise is lost, there's also that sense of betrayal, of loss of trust - even trust in life itself.

Some of the implications of this can go very deep indeed - yet the core of it comes right back to these three themes:

[61] http://en.wikipedia.org/wiki/Muda_(Japanese_term)

- product as a promise of future service
- trust in those promises as a foundation for enterprise - especially 'enterprise as service'
- waste as the loss of the promise or loss of trust

How would you map these in *your* enterprise, and for *your* products and services?

Nothing more that I'd add to that for now: it's just a thread that's been weaving around for me over the past few days. Over to you for comment, perhaps?

Source (Tetradian weblog)

- *Date*: 2014/03/25
- *URL*: product-service-and-trust[62]
- *Comments*: 2
- *Categories*: Business, Enterprise architecture
- *Tags*: Business, effectiveness, enterprise, Enterprise architecture, product, service, trust

[62] http://weblog.tetradian.com/product-service-and-trust