

Reactive Fabric Technology

White Paper Part 2/3 - Informal Introduction

SDK For iOS, OSX (and coming for Linux)

Product White Paper

October 2019



"Language shapes the way we think and
determines what we think about."

Benjamin Whorf, American Linguist 1897 - 1941



Author: Terry Stillone (terry@originware.com)

Web: originware.com

Version: 1.0

@ 1. Introduction

1. Introduction to Part 2 of the White Paper.

This section (**Part 2**), centres on the background concepts of the **Reactive Fabric Technology**. It defines the "**what is**" of the technology in **conceptual terms** for a **non-technical audience**. The next follow on **Part 3**, will go into the technical specifics of the technology.

Summarising **Part 1**, The **Reactive Fabric Technology** takes a **connective** and **supportive** approach to the whole software developmental organisation unit. It connects the **technical** staff (who synthesize and refine **Software Designs** and produce **Implementations**) with associated non-technical parties (who **support** the general development process).

2. The Concept of "The Design Principle".

Nature as a designer, employs a number of **Design Principles** in its portfolio of **Biological Systems**. **Reactive Fabric** employs a major principle that nature exploits for **control** of these **Biological Systems**. In interactional terms, this design principle is:

An emitter element (termed a **Producer**) deploys an active **messenger** through a **transport pathway** to:

- ⇒ **Deliver the message** to a target element (called the **Consumer**).
- ⇒ To **Engage** the target element to **act upon the received message** and perform something.

Examples of this **Design Principle** in **Biological Systems** include:

- ⚙️ Electrical impulses routed through nerve pathways.
 - Employed to engage muscular action by the brain/reflex system.
 - To propagate sensory signals from sense organs to the brain.
- ⚙️ Chemical signalling messengers that traverse vascular pathways.
 - Used by the endocrine glands to effect and regulate body changes.
 - Engage and regulate immune system response.
- ⚙️ Electro-chemical messaging in synapse pathways within neural clusters.
 - Encode electrical based temporary memory patterns to permanent neural encoded memory patterns.

This **Design Principle** is also incorporated into the design and operation of electronic devices:

- ⚙️ A driver transistor injecting electrons along a conductive pathway to a receiver transistor.
- ⚙️ A laser emitting messenger photons along glass fibres to a target photonic receiver.

Reactive Fabric abstracts this natural **Design Principle** into the basic concepts of:

- The **Notification** (i.e. the message).
- The **Element** (that emits, receives and acts on Notifications).
- The **Transport Pathway** that inter-connects **Elements** for notification emission and reception.

Lets concrete these abstract ideas with some examples, lets treat the basic, mechanical operation of the **human body** as a **system** in itself and create a conceptual system design that reflects the interaction between these elements. **Notifications** will represent the electrical impulses between the elements (body parts).

Of course, the body in itself is an extremely complex organism, but ... we are not implementing a full system here, we are just **describing the conceptual high level design**. It is engineers who will then take our high level designs, form micro detail-level designs and follow though with a whole system implementation. That engineered system will still reflect our conceptual design (just with more detail).

Note

1. Introduction

3. Example Conceptual Design.

Here to the right is an example **body** design. The elements (**head**, **arms**, **hands** and **legs**) form the functional elements. The **blue forward arrows** indicate forward messaging pathways between the elements and smaller secondary **red (back directing) arrows** indicate back reply messages triggered by their forward counterparts.

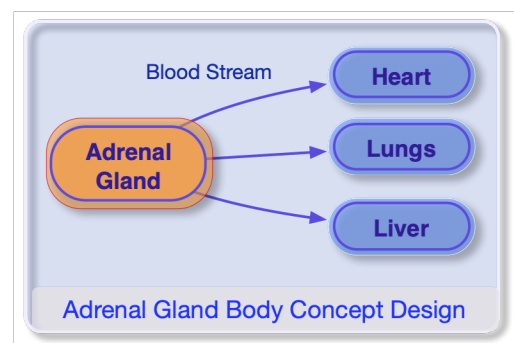
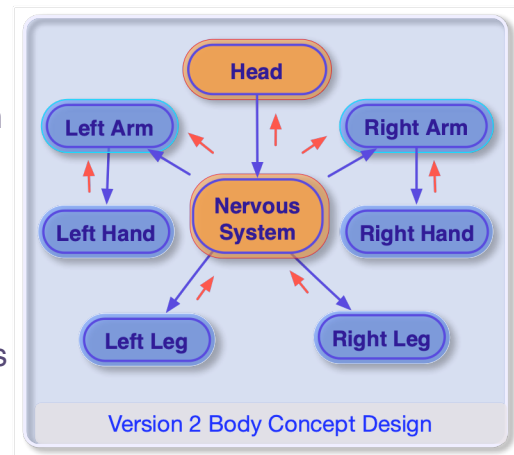
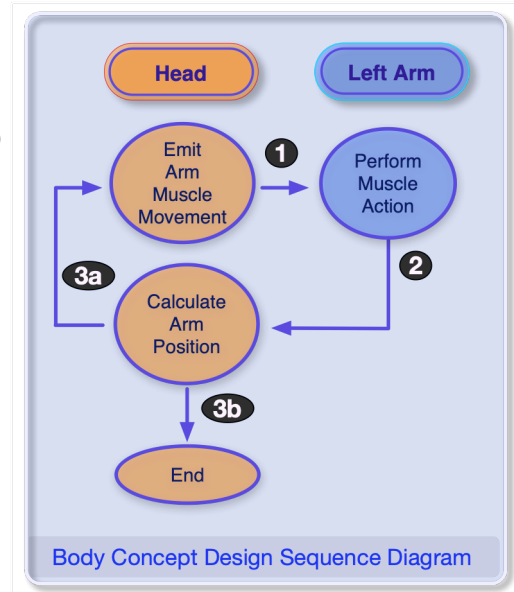
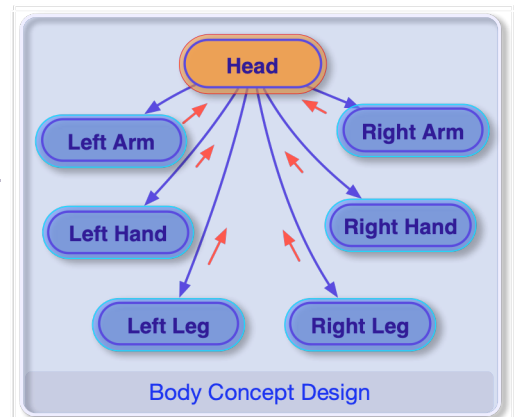
When the **head** element commands the left arm to move, it issues a notification (message) along the **blue** arrow pathway to the left arm. The **arm** element then responds to the forward notification with muscular action and replies back with notification (feedback) telemetry (in the **red** arrow direction). This telemetry indicates the active muscular response of the arm. The **head** then interprets those muscular responses to compute the current **arm** position and in turn regulates its emission of forward notifications to effect the intended full movement action and then bring the arm motion to rest. This notification interaction is described in the next **Sequence Diagram** to the right.

This particular design exhibits a **centralised interaction topology**, utilising low level notification commands. It results in the notification pathways exercising a high volume of notification chatter with possible overburden on the processing and energy demands of our system. So if this was a **Version 1 Design**, let's propose a design iteration change for **Version 2** with more appropriate energy consumption.

Let's get the individual elements to be addressable, more intelligent, and handle high level movement commands which indicate the position to move to. In our new design (next diagram to the right), all notifications are now routed through a **Nervous System** element. When the **head** needs to effect a left **arm** movement, it emits a high level notification that includes the target element to engage into action and the relative position it is to move to. The **Left Arm Element** is more self-aware in the new design, in terms of its position relative to the body and back-replies when it has moved into position, so as to verifying the position it has reached. This new design trades off notification volume for more complex, self-aware elements.

The new topology also provides for a central routing element (i.e. the **Nervous System**) and so provisions the possibility of that element being used as a "**Test Point**" for monitoring and injecting notifications and for testing individual body elements. The **Test Point** can also be used to record real time notifications which can then be used for playing back into the **Nervous System** and simulate activity. The design also makes the **Head** element more independent of the **Left Arm** behaviour as it only needs to understand arm positions and not muscular dynamics and arm mechanical behaviour.

In these body designs, we have described a "**macro system behaviour design model**". We can also come up with medium level designs. Here on the right is a simple concept design for the body adrenal glands. Notice this design is for a one way messaging model as opposed to the previous bi-directional model. The Adrenal glands message through the blood stream pathway and messaging is through the use of a hormone molecule rather than an electrical impulse.

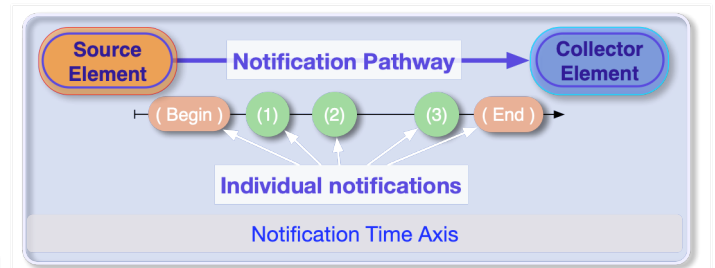


2.Reactive Fabric Concepts

1. The Concept Of The "Notification"

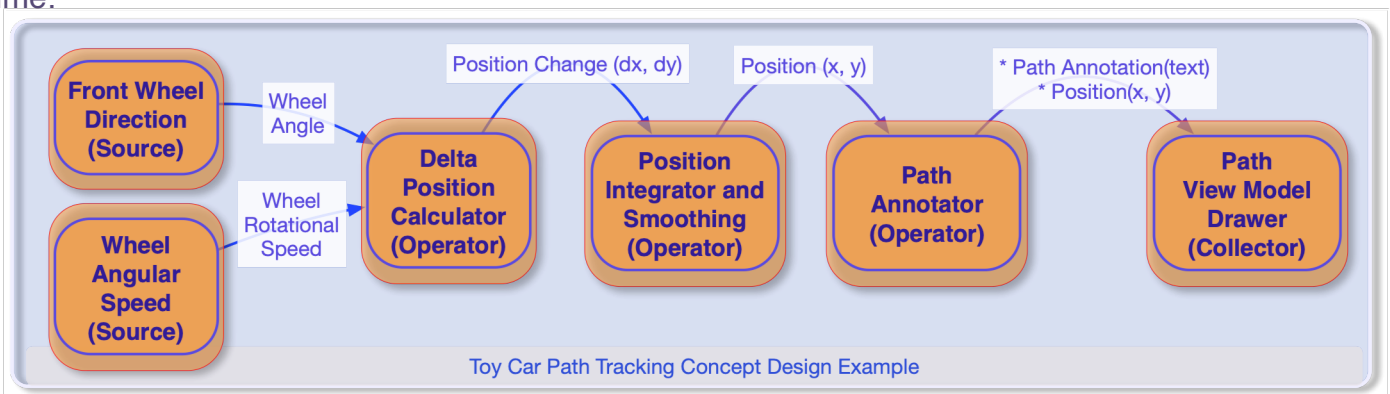
The "**Notification**" concept is an analogy of the *electrical impulse*, the *chemical messenger* or the *network data packet*. In abstract terms, it is a transportable data entity which requires a transport path for traversal between **Elements**. The **Notification** encapsulates the information that is to be messaged between particular elements and over time, collectively constitutes a **Data Stream of Events** with their associated event-data encapsulated in the notification payload.

Producers of notifications are termed **Source Elements** and consumers are termed **Collector Elements**. The diagram to the right depicts, a **Source** element notifying a **Collector**. There are control notifications (i.e. the **Begin** and **End** stream notifications) and there are data notifications with an integer data-type (values: 1 .. 3).



As **Source** elements merely produce notifications, **Collector** elements only consume, **Operator** elements on the other hand both consume and produce. **Operators** derive their output notification from the corresponding input notification data. Thus a systems **behaviour** and **state** is described by notifications flowing through the whole system in the same way that a physical organic system changes its state as a response to stimuli. All this interaction-activity ultimately constitutes the **whole system behaviour**.

Lets look at a more direct application of the design principle. Say we have a toy remote control vehicle and we are to design a mobile **Reactive Fabric** app that displays telemetry from the vehicle. The telemetry includes the toy's wheel rotation speed (in revolutions per sec) and front wheel direction (in degrees). Our **App** is required to perform realtime analysis, to compute the path of the vehicle and draw it on the device's screen. Our design (given below) describes the reception of telemetry as two source elements, one for the front wheel angular speed (**Rotational Speed** notification) and one for wheel direction (**Angle** notification). We utilise an **operator** to perform a convolution of those two source element notifications to calculate the **delta positional change** (as a velocity vector: **(dx, dy)**) and then feed that into an **integration operator** to calculate the travelled position of the toy **(x, y)** over time:



Going into more detailed **App** requirements, when we are drawing the motion of the toy (on the device display), we want to display a smooth path on the screen (not a jagged one). So lets add smoothing capability to the **Position Integrator** operator. We also need to put some text up on the screen to describe the speed of the toy, (i.e. indicate when stopped and when in motion with a speed level). So lets add an additional **Annotator Operator** element which takes the position from the **Position Integrator**, analyses the position at various time increments and emits a resultant textual summary of what the toy is doing. The annotator then emits textual annotations when the toy is stopped, in motion (with a speed level), when reversing, etc.

The results of the **Annotator Operator** and the toy positional information are fed into the element that becomes the presentation **View Model**. This element has the role of drawing the toys path on the


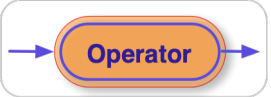





🌀 3.Reactive Fabric Element Patterns

1. Reactive Fabric Patterns

Reactive Fabric abstracts common **Element** roles into a set of **Reactive Fabric Patterns** that are reflective of their general use.

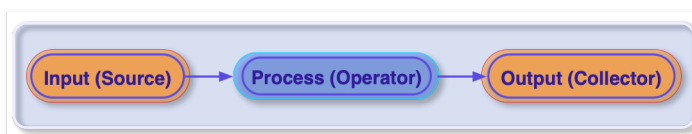
- 🌀 In the **technical-design** domain, these patterns differentiate the various element **topologies** (i.e. whether they have inputs, outputs, reply pathways, etc).
- 🌀 Where as in the **socio-communicative** domain, they form the **base model vocabulary** and are used during discussion to describe the facets of the model.

The table below lists the base design patterns together with their respective topologies.

Pattern	Description	Topology
Source	Solely generates Notifications, does not receive notifications or back-replies. Produces to one-way notification path.	
Operator	Operates on input Notifications to derive output Notifications. Consumes and produces on one-way notification paths.	
Collector	Collects Notifications and makes them available for consumption. Consumes from a one-way notification path.	
Controller	A Controller of elements. Acts as a source but also can receive back-notifications (a bi-way variant of the Source).	
Service	Provides a notification Service, receiving request notifications and back-replying results.	
Socket	A bi-way variant of the Operator.	
SubSystem	A Subsystem manages a sub-collective of Elements. The Subsystem input and output notification streams are made available to the sub-collective elements for internal processing. Their resultant notification stream is then forwarded to the output stream of the Subsystem.	

We now follow on to how a real system is decomposed into an **Element** collective and function as a real system. An **Element** collective is called a "**Fabric**".

All systems, have inputs, outputs and some intermediate processing mechanism that connects those inputs to outputs. This is typified by this very simple design:



But real systems typically have a myriad of inputs and outputs and require complex processing. A calculator for example, has an input for each button, an output for the main display, then output indicator labels for the various calculator modes and there are register stores and so on. The processing **Fabric** for a calculator would need to not only need to perform calculations but conversions depending on calculator modes. Let's go on to build a calculator design (conceptual design).

4. How Does A Reactive Fabric Perform

1. Example Designs

In the calculator design given to the right, inputs are on the left side of the diagram and outputs are on the right. Each button input and calculator mode switch is represented as a **Source** element and each output calculator display-element is represented as a **Collector**.

The digit buttons feed into the element that describes the **Register Stack** of the calculator, which stores the current calculation values. The **Register Stack (Service)** also drives the **Result Display** to show the top value of the stack. So when digits are being entered they are emitted individually to the **Register Stack** which then appends them into the top stack register entry. Those changes are then emitted to the **Result Display**, so that the display reflects the entered digits as they are pressed.

When a function button is pressed to perform an arithmetic operation, the **Supervisor Operator** retrieves the value(s) from the **Register Stack**, forwards them to the **Arithmetic Operator**, which performs the calculation and channels the result back to the **Register Stack**, replacing the top-value(s) used in the calculation with the result. The **Register Stack** then updates the **Result Display** with the new result. The **Supervisor** also retrieves the **Calculator Modes** before performing a calculation and includes them with the values to the **Arithmetic Operator** so that the calculations are appropriately converted.

Not all systems have multiple inputs as the calculator example, some systems have only one Input/Output Element (**IO Element**) and the **Fabric topology** of a design will naturally reflect the character of the systems input and output (**IO**).

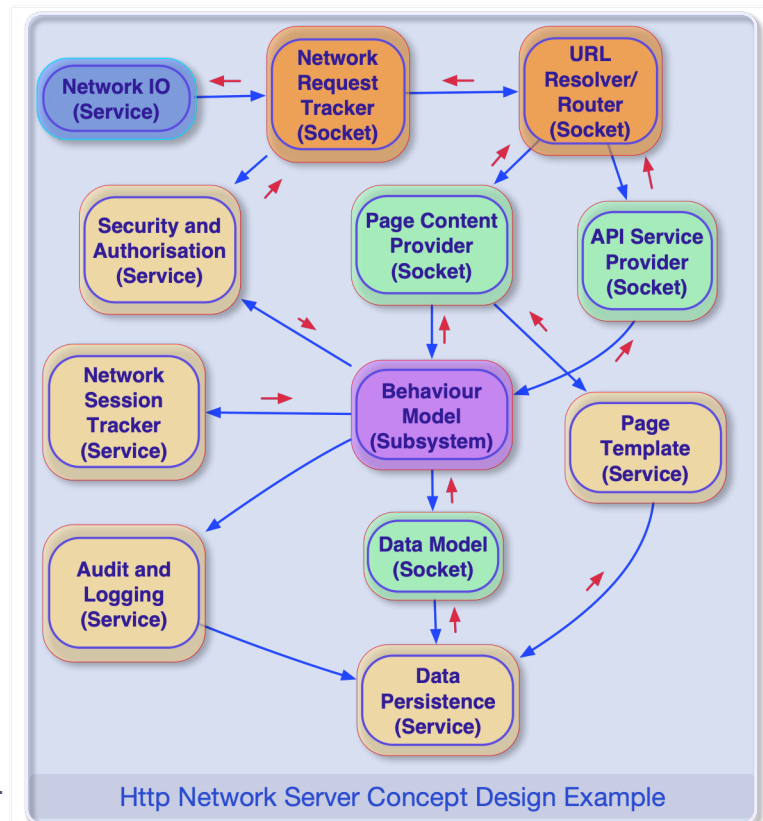
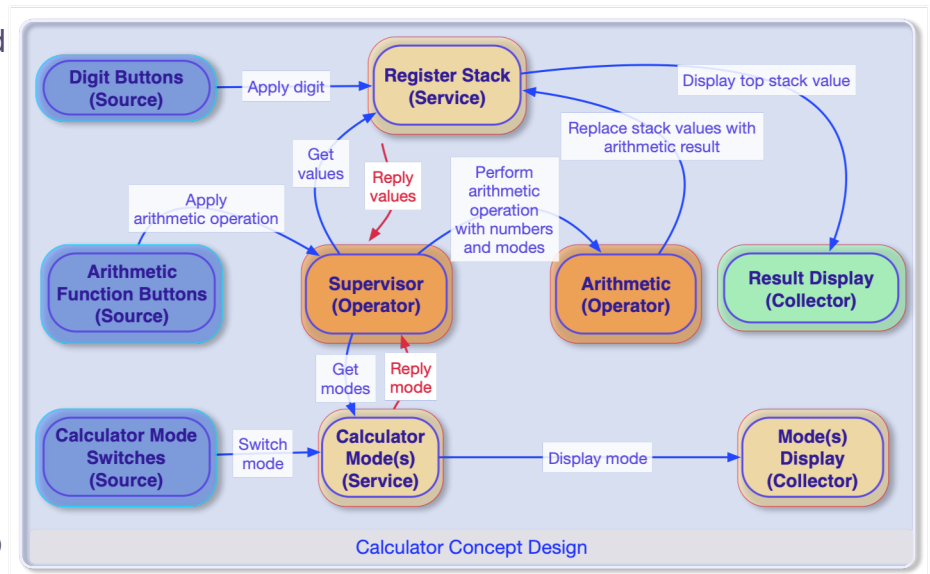
Here is an example of a **HTTP Web Page Server** with a **Network IO** element which functions as the only **IO Element** for the **Fabric**:

Focusing on the internal elements of both of these designs, let's discuss their general character.

Some simpler data-processing expressions have a **one-way** notification pathway character where as more complex interactions will tend to require **bi-way** notification interchange.

Simple data-processing is typically performed by **Operators**. Shared services (within the fabric) will tend to be **Service** elements. Complex expression pipelines will tend to be represented by **Socket** chains.

Where element collectives co-perform a joint role within the system, they can be encapsulated into a **Subsystem** element. Complex **Service** elements may also be internally composed of element collectives.



5. The Reactive Fabric Development Process

1. Reactive Fabric Design as a Group Collaborative Process.

Reactive Fabric engages the whole team in the design process, from management, to software design & development, testing, documentation and possibly even affecting non-technical facets such as marketing. This cross connective-engagement is precipitated by providing the design models to all parties and most importantly it makes them:

- ✳ Available at their own level of engagement.
- ✳ Available at their own level of appropriate detail.
- ✳ and this is realised through the use of model LOB (Level Of Behaviour).

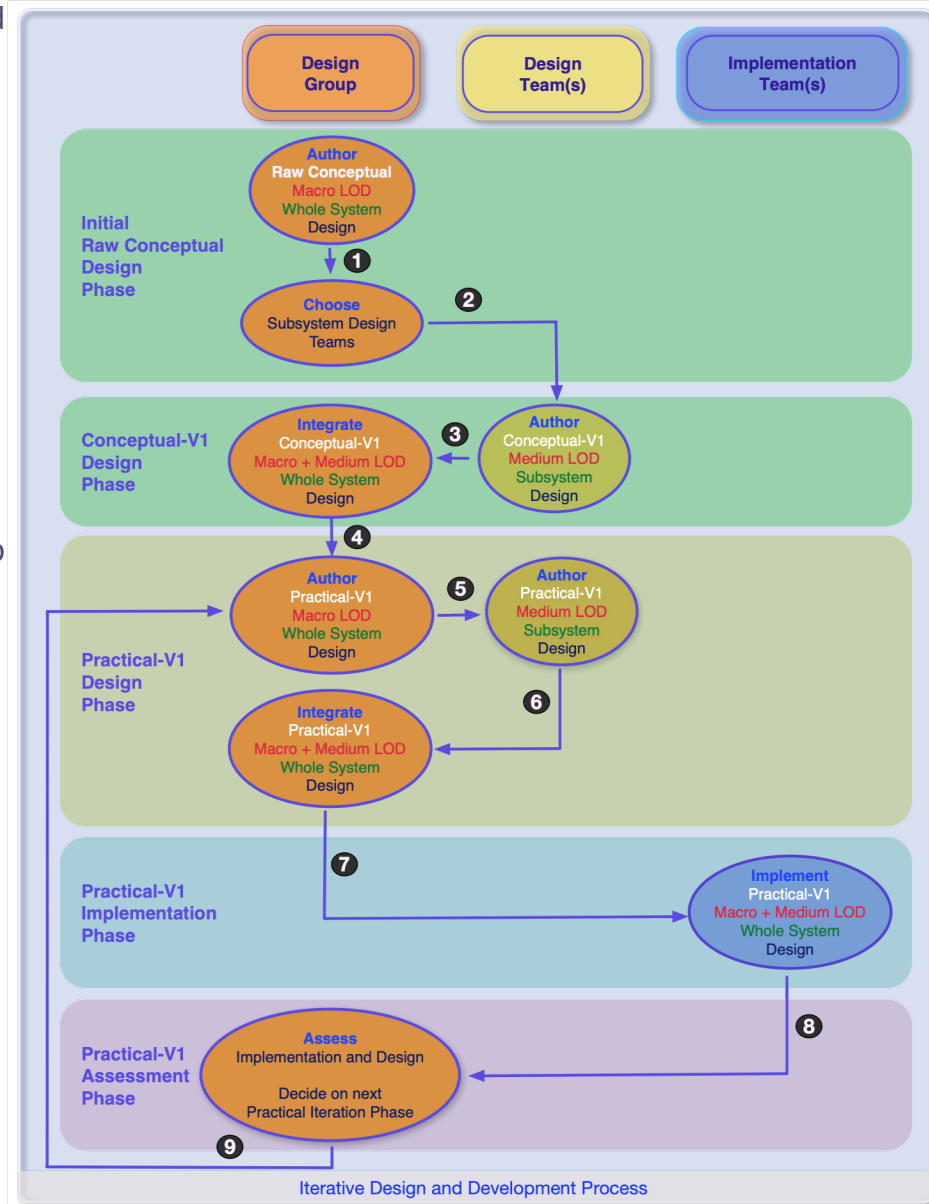
Functional-Practical Designs will tend to be handled by technical staff and less detailed, more **Conceptual Designs** are typically made available for consumption by non-technical associates.

As the technology **principle** is based on **Design for Interaction**, the process of developing these designs reflect in the team group interaction. The quality, authenticity and openness in the team interactions back-reflect to the quality, authenticity and openness of the design product.

The development process is performed in a number of design and implementation cycles. The first cycle is to produce **operational design models**. These initial models are on the **conceptual** level, not **practical-technical** (meaning they are not implemented). The conceptual models form a base from which practical designs are derived and then these practicals go on to be implemented. This development cycle-process is outlined in the sequence diagram to the right:

The whole process begins with a group meeting to look at project **use-cases** and form the first **conceptual macro-operational model** (which is very much a raw macro-design-product). This conceptual design naturally defines **functional boundaries**, which are used to synthesize **teams** and **team sizes**. Those teams then individually over time, work on their assigned raw concept models to detail and mature them for integration back into the original macro-design. The original design group is then reconvened to back-integrate the team models. The group eventually agrees upon the first conceptual model, tagged as: "**Iteration 1 Conceptual Operation Design**".

Once the conceptual design is accepted, the practical design phase begins to produce **Iteration 1 Practical Design Model** (which follows a similar development process). **Practical** designs are more piecemeal and are iterated over, to incrementally build up functionality and each is used as a backing for an implementation (which reflect their design incremental functionality). The practical iteration process continues until full functionality is realised.

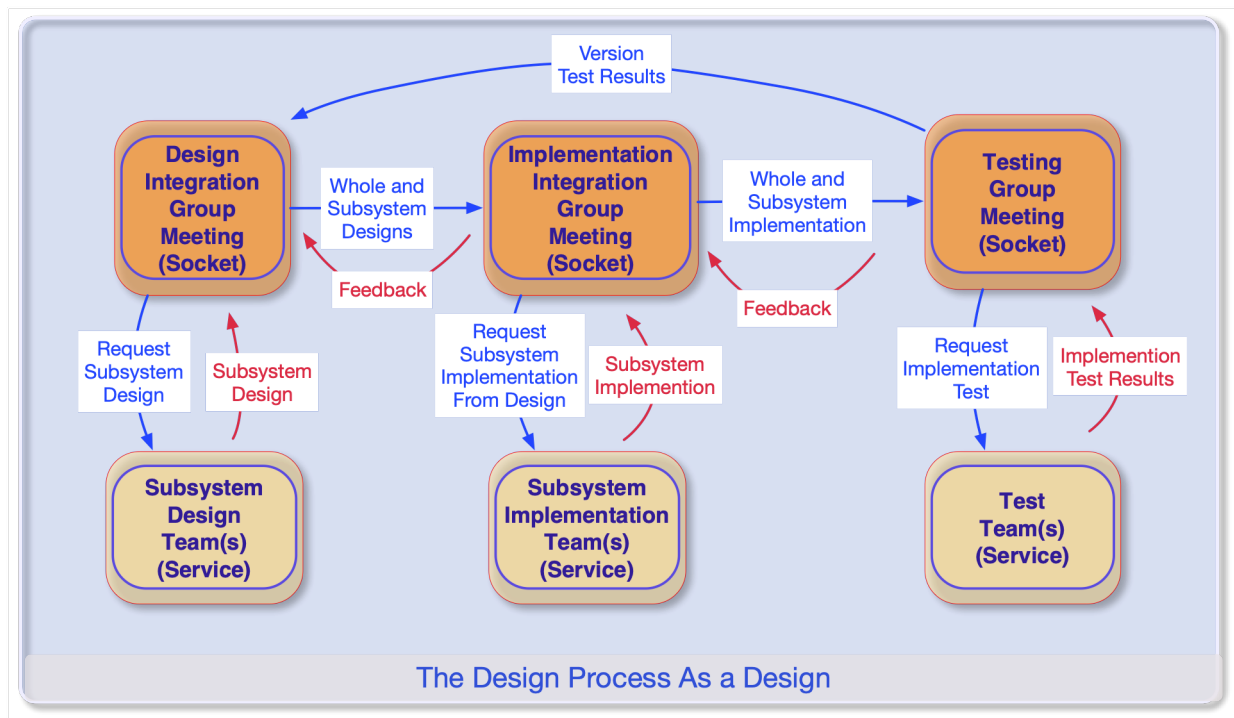


⑤ 5. The Reactive Fabric Development Process (Cont)

2. The Reactive Fabric Design Process as a Design In Itself.

In fact, the whole **Reactive Fabric Design-Process** can be modelled as a design in itself, reflecting team interaction. The main characteristic here to note is the **pipeline** character of the topology which expresses the fundamental process of:

Design ⇒ Implement ⇒ Test



Also, note that the upper row of **Sockets** represent the integration group meetings while the lower row of **Subsystem** elements represent the individual work-teams.

The teams denoted in the diagram may not be composed of all separate individuals. Teams will probably be sharing members as they operate at different cycles. The **Design Integration Group**, may come together to integrate separate sub-system designs and then break up to form the individual **Subsystem Design Teams**. For smaller projects, the implementation team members may also comprise of the design team members, again, coming together in an implementation cycle and then dispersing to re-form design teams. Also, sprint cycles can be arranged into the design-implement-access phases so as to have design sprints followed by implementation sprints and so on.

The **design** of the **development process** needs to be addressed before or at outset of the development cycle. There is huge flexibility as how and what the **development process** design ends up being. The design must cater for team size, expertise, project priorities, project and product complexity, timelines etc. All the trade offs that come into play when designing software also come into play when designing the project **development process**. There is even scope for iterative design of the project **development process**. The final project cycle may require a different **development process design** to the initial project cycle and situations may arise that preempt re-addressing the **development process** design mid-project, such as loss of key members, changes to product goals and requirements.

© 6. More Information

1. Further Reading

This **White Paper** continues onto **Part 3** (see links below), which formalises the technology.

2. For More Information.

❄ Reactive Fabric White Papers and Briefs:

- The Reactive Fabric White Paper, in three parts:

- [Reactive Fabric White Paper Part 1 of 3.pdf](#)
- [Reactive Fabric White Paper Part 2 of 3.pdf](#)
- [Reactive Fabric White Paper Part 3 of 3.pdf](#)

- For Desktop and Mobile Apps:

- [Reactive Fabric For Desktop and Mobile Product Application.pdf](#)

- For IOT:

- [Reactive Fabric For IOT Product Application.pdf](#)

❄ Website pages:

- Originware site: originware.com
- The Reactive Fabric web page: originware.com/reactivefabric.html

3. Reactive Fabric Samples.

Apps designed on Reactive Fabric principles:

- ❄ The **Dronenaut** iOS App on the App Store
(employs the older **Reactive Patterns SDK**)

[The Dronenaut App](#)

- ❄ The Reactive Fabric Sample App:
(employs the **Reactive Fabric SDK**)

[Reactive Fabric Sample Kit For iOS](#)

4. Contact.

Please direct questions and requests for more information to:

Terry Stillone (terry@originware.com)