Reactive Fabric Technology

White Paper Part 1/3 - Benefits Of The Technology

SDK For iOS, OSX (and coming for Linux)

Product White Paper

October 2019

Originware Techology

Author: Terry Stillone (*terry@originware.com*)

Web: *originware.com*

Version: 1.0

1. Introduction

1. Why Read This White Paper ?

If you work in the Software Development industry, you probably understand just how complex software systems have become. You probably also appreciate the relentless drive to continually encompass more and more product capability. This situation sets up a developmental syndrome, which inevitably leads into software complexity escalation. As a reflection of this trend, I take it that a part of your role is to manage the impact of complexity creep and you have an active interest in solutions.

If your role encompasses product management, then you probably have concerns over product complexity and the impact on balancing **schedules**, project **resources**, product **quality** and product **confidence**. If your work envolves implementation, you are probably constantly trading off between meeting development schedules and the level of **introduced complexity** together with its associated **technical debt**. You probably also schedule cycles to address complexity overreach only when it reaches critical mass (that is the agile method). Unfortunately, this strategy does not directly address the core issue. You knew from the outset of the project, that complexity was going to be a concern, so why didn't incorporate processes to handle it from the beginning of the project?

Software language hasn't solved the complexity problem: new evolving software languages are supplying more flexibility and greater conciseness of expression but language has been traditionally focused on **micro-capability** and given little priority to managing **macro-complexity**. While language-technology has been evolving, pure software design technology has not. **Design** should be managing the **macro-complexity**, leaving language (code) to concisely express **micro-behaviour**.

Nature designs systems that appear complex in behaviour but in actuality are comprised of simple independent micro-processes which interact as a whole organism to perform complex interaction. **Reactive Fabric** employs those same design principles to incrementally design complex software systems while retaining the simplicity of the individual micro-processes. As such, **Reactive Fabric** is fit for purpose in complex systems but it is also applicable as a unifying agent for projects with multiple (vertical) platform tiers that require a unified design-process across the various platforms and disparate components.

Reactive Fabric Technology is a "**Design for interaction**" *software architecture* / *design methodology* / *design notation* / *design process* / *design collaboration* / *implementation SDK* that supports all members of the project and reaches into all aspects of the of the development process.

Put simply, **Reactive Fabric** addresses complexity by visually modelling interaction as:

Structured Inter-Processing-Element Notification.

2. This White Paper is Factored Into Three Parts.

The **Reactive Fabric Software Technology White Paper** is a separated into three parts:

- **Part 1** - The Benefits of Reactive Fabric (relating to the whole development process).
- **Part 2** - An Informal Introduction to Reactive Fabric Technology (expressed in non-technical concepts).
- **Part 3** - Reactive Fabric Formalised (covers architectural specifics and technical formalisation).

This part focuses on the **benefits** of the technology. **Part 2** is an informal discussion on background concepts that can be read by anyone. It then follows onto technical formalisation in **Part 3**. The outlining of **Reactive Fabric's** "**what**" (such as background concepts), is deferred to **Part 2**. If you are the type of person who wants the "**what**" first, it may assist to read **Part 2** first and then recommence here.

3. Applicability of Reactive Fabric.

Reactive Fabric has applicable:

- **As a Software Technology,** it can be applied through the vertical stack (**Server**, **Desktop**, **Mobile App** to **SOC/IOT**). It is perfectly suited for multi-platform services and multi-device systems.

3. Applicability of Reactive Fabric (Cont).

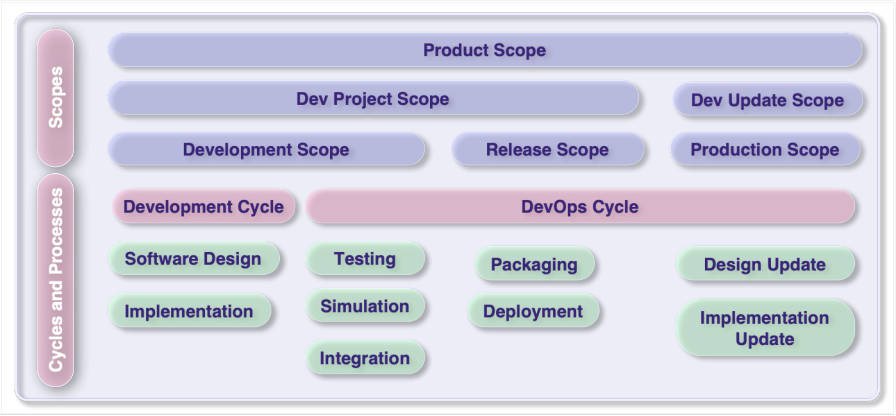
- **As a Developmental Methodology**, it naturally brings people together to collaborate, to share and exchange ideas. It supports **Agile** across the board: incremental and iterative design-development-testing-simulation.
- **As a Software Design notation and vocabulary**, it provides a concise, formalised design documentation system that can be shared to the required level of detail. **Intellectual Property** can be protected while sharing designs by selectively publishing-out only high level macro interactions as it is the micro-level interaction-designs that really detail the operational "how".

Reactive Fabric is not applicable for strict real time control as it cannot guarantee strict timing.

1. Benefits: The Developmental Organisation Unit.

Every organisation is an ecosystem in itself. While the variations in these ecosystems can be quite broad, here is a simplified representation of the vertical role stack in a typical developmental organisational unit:

Using this organisational graph, lets look at the benefits **Reactive Fabric** provides for each scope-domain.



2. Benefits In The Product Domain.

Product and high level **Project Management** are traditionally distant from technical details and so lack visibility into what the product is actually doing under the hood. **Reactive Fabric** allows designs to be available and comprehendible to all project levels. Product management no-longer have to accept themselves as being foreign or peripheral to the (internal) operation of the product. Product management can have **up to date visibility** of the design space on the level of detail relevant to them and have a greater understanding of the challenges facing technical groups. The incremental design aspect also offers a window into developmental progress.

Designs provide a medium (both as a visual and vocabulary base) to describe operational process and the medium can be used to share designs with other interested parties. The medium also allows technical staff to more fully expound on problems and solutions to management.

3. Benefits In The Project Domain.

There are benefits derived from having the software project setup with:

- A "**Design Space**" which owned by the group**,**
- **Produced** as a team collaboration**,**
- and **Made** open to authorised parties for feedback.

Design becomes a **shared**, **team experience**, which challenges and **feed-forwards** designers, keeps peripheral **parties-of-interest** informed of progress and associated problems but more importantly, these benefits all flow back into the individual members to better self **actualize their potential** (and to grow in their respective fields).

As an **iterative** design process, **Reactive Fabric** brings in a greater **agility** to the project. It synchronises cycles of design with cycles of implementation and testing. It integrates into the fabric and **heartbeat** of the development process to become a cycle of:

Design ⇒ ***Implement*** ⇒ ***Test*** ⇒ ***Assess*** and ***Repeat***

3. Benefits In The Project Domain (Cont).

Reactive Fabric broadens the scope of expertise of development team members, fosters a deeper appreciation of the **whole system** and developers no longer have to read bulk code to understand the backing operational design. It speeds system comprehension, allowing incoming developers to come up to speed more quickly and easily. Pre-design ahead of implementation allows ancillary groups to give pre-implementation feedback and allows them to better synchronize with the project.

In **Reactive Fabric**, the implementation becomes a **projection of the design** in the **Design Space** onto the **Code Space**. Some of the issues typically handled in the code space can be pre-envisioned in the design space and handled there instead.

Designs can be shared with clients of the product so that they can better understand the process behind the product Visual or API interface.

4. Benefits In The Development Process Domain.

Working with **Reactive Fabric** designs at the developmental level requires a mind/paradigm shift. Additional, high level operational representations come into play. Rather than thinking of in terms of functions/object-classes/properties/traits/protocols you work with interactions. You visualise operational process in terms of "**Notification Data Types**" and their flow through a **Fabric** of "**Notification Processors**". You essentially "**Design for Interaction"** through the application of primitive interactions and as these primitives are organised into **patterns**, in effect you design in terms of "**Interactional Patterns**".

The **Reactive Fabric** development process is collaborative by nature as it operates on the "**design for interaction**" principle. It forces developer/designers to collaborate on the evolution of focus interactions. It provides a **visual notation** and a **base vocabulary** for people to describe and discuss mutual interaction in their designs.

Interactional design opens to a wider scope of possibilities, such as:

- Pre-adaption to target environments (i.e. flexible configuration).
- Dynamic adaptive behaviour, reacting to changing environments.
- Interactional modelling of swarm (device) interactions.
- Modelling of exchanges, such as protocols and communication network interactions.

Reactive Fabric brings the developer/designer into a richer realm of expression which is ultimately more engaging and rewarding.

5. Benefits In The Testing and Simulation Domain

Reactive Fabric naturally supports and moulds into testing and simulation. As designs are decomposed into primitive interactions (of notifications), notifications become the prime mechanism for test-data, test-data delivery and test-data validation. This is discussed more fully in Part 3.

Again, designs give operational insight to test-staff as to what they are in fact testing. It provides a platform for how it is to be tested. It allows the test team to participate more deeply in the design of the test process and the design of the testing framework.

2. Benefits (Cont)

6. Benefits On App Life Cycle Management and The DevOps Level.

Reactive Fabric provides for system behaviour adaptation, which in turn can provide solutions for:

- Fail-over behaviour.
- Graceful handling of low internal and external resources.
- Performance thread load balancing.
- App energy management through strategies of notification throttling.

As systems become more interactionally complex, those teams that test and operate facilities require more visibility into the product system operation. The **Reactive Fabric** design process assists **DevOps** to discern system dependencies and understand system-operation cycles (such configuration, adaptation, etc).

3. Example Conceptual Design

1. Example Design Introduction.

To convey what a design does actually look like, lets look at an example of:

A **Conceptual, Macro-Level** Design of a **Mobile App.**

The App's role is to allow a user to view their submitted **Jobs** on a hosted **Net Server** (here, the term of **"Job**" is taken to be an abstract concept). The components of the design are comprised of:

- **Notifications** (i.e. messages that represent interaction).
- **Elements** (processors of those Notifications that denote the effecting of those interactions).

The design (given on the next page) conveys the essential character of the **App's** interaction of:

User ⇔ Job ⇔ Portal

This character, reflects the intrinsic interaction that:

- The **User** initiates interaction.
- Which in turn engages **Job** requests.
- That triggers **Portal** exchanges.
- and those generate back replies along the chain to reflect their results.

These interactions are modelled as **Notifications** between respective **Elements** and these **Elements** perform their assigned processing roles on those **Notifications**.

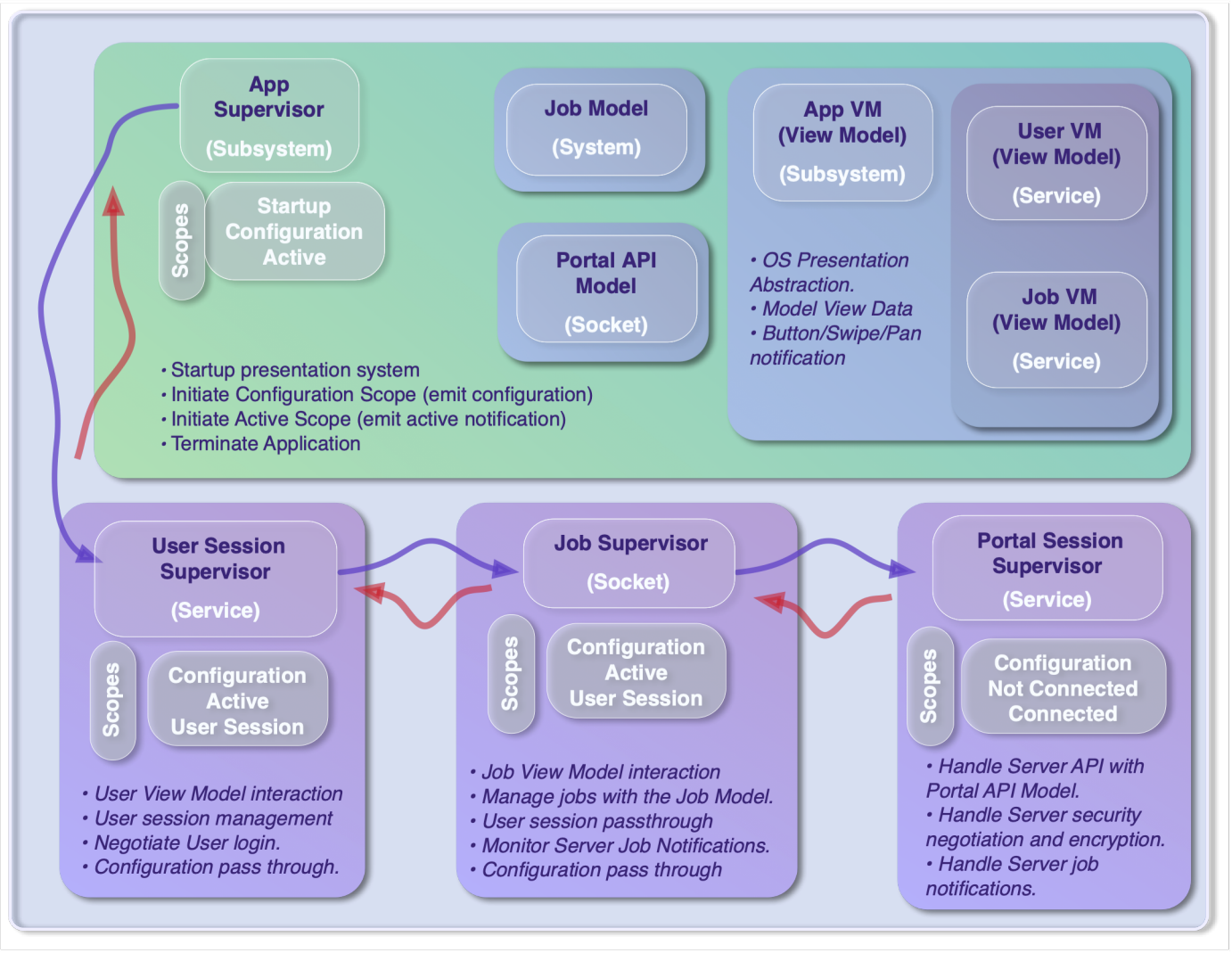
Note

Conceptual Designs are intended to convey **essential**, **core interaction** and withhold detail. **Functional Designs** on the other hand, provide full detail are used by technical staff to author implementations. So this particular **Conceptual** design below is devoid of detail about **Notifications**, it just conveys the essential interactional character.

3. Example Conceptual Design (Cont)

2. Example Design Description.

In this particular design, as well as exhibiting the essential **operational** character just discussed, the design also overlays some **lifecycle** characteristics, for a configuration cycle. During the configuration cycle, specific models (such as the **Job Model** and the **Portal API Model** are deployed to their respective hosting **Elements** for their execution).



Explaining the operation of this design, the **App Supervisor** element manages the whole system and so is the causal initiator of system. It has a number of operational states (termed **Scopes**) for:

- **App Start Up**.
- **App Configuration.**
- **Active Operation**.

The system starts in the **App Start Up** scope during which the supervisor performs enviromental identification (and possibly enviromental checks) and then auto-transitions to the **Configuration** scope. In the **Configuration** scope, the **App Supervisor** configures the subordinate **Elements** and then auto-transitions to the **Active** scope. It is here in this scope, that the **App** attains its full functional operational state.

The App design also incorporates configurational flexibility to cater for varying environments. It can configure for different **Portal APIs** by deploying the appropriate **Portal API Model** for the identifed environment. It also can adjust for differences in the content or layout of views by providing environment specific **View Model**(s). **The App Supervisor** is the actual store of these models and emits them as a number of **Configuration Notifications** during the **Configuration Scope**.

4. Technology Pros and Cons

1. Technology Pros and Cons.

The table below outlines the pros and cons of **Reactive Fabric Technology.**

Aspect	Pro-s	Con-s
Introducability to new projects.	Provides long term benefits, discused earlier in Part 1.	Larger learning and tooling cycle.
Introducability to existing projects	Flexible enough to be introduced in piece meal fashion.	Larger learning and tooling cycle.
Agility, Extensibility and Maintainability.	Supports incremental design. The inherent modularity of the **notification-element design** asists subsystem and expression insertion, replacement, enhancement and refinement.	Changes in system design incurs modifications to test models that test the system. These issues should also be addressed when re-designing.
Applicability	Applicable to the full vertical stack from Server, Application, App to SOC.	Does not support strict Realtime control.
Modularity	The **notification-element** design is intrinsically modular and decouples dependencies.	
Configurability and Adaptablity	Designs are open to perform system configuration through the Notification mechinism. Open to design adaptablity through changes to element behaviour and changes to topology.	
Scalability	Performance issues can be identified and localised by analysing notification and element performance with native SDK tracing, performance analysis tools and debuggers.	Notification passing incurs processing overhead. Options for handling these overheads are discussed in Part 3. If there are possible hard CPU resource walls that may be hit, then there should be initial simulations conducted to assess what is options are viable.
Testability	Naturally decouples subsystems and provides natural interfaces through the use of notifications-element design. Reactive Fabric Test Rigs bring structure and organisation to testing.	Testing will typically be more complex than a classical synchronous system, as the Technology caters for asynchronous applications.
Traceability and Debugability	Reactive Fabric supports both comprehensive and selective notification tracing. Tracing provides high level assessment of notification propagation and flow. Debuggers can be employed for manual debugging of notifications and element processing.	
Robustness, Reliability and Failure Handling.	Scope is available for graceful failure design, for adaption to changing environments and varying resource levels.	As with any system, reliability depends on testing. As Reactive Fabric addresses complex systems, more extensive and varied testing is most likely required.
Portability		Currently only the Swift language is supported in the SDK. The Kotlin language is on the roadmap.

5. Technology Application

1. Applicability Domains For Reactive Fabric.

Platform	System	Application (for Designing and Modelling)
Hosted Systems or Containers	Linux Server	Micro-Process behaviour. Process/thread behaviour. Inter-process protocols and interactions. Testing and simulation.
Software Application	Desktop or Mobile Application (OSX, iOS, Linux)	System behaviour. Network interaction. Attached device control & interaction (GPS, medical, etc). Testing and simulation.
IOT	Linux Device	Device and edge behaviour. Network and peer interaction. Attached sub-device control & interaction. Testing and simulation.
	Swarm	Collective swarm interaction. Testing and simulation.
Electronic Applicance or Toy	Linux	High-level system management. Directing input data (to AI processing nodes). Directing output results (from AI processing nodes). Co-ordination of separate disparate compute units.
AI Hybrid Systems	Linux	High-level system management. Directing input data (to AI processing nodes). Directing output results (from AI processing nodes). Co-ordination of separate disparate compute units.
Legacy Hybrid Systems		High-level system management. Managing and directing input data and output data. Co-ordination of separate disparate execution units.
Network Interaction		Protocol design and modelling. Network operation modelling. Protocol and network operation testing and simulation.

6. More Information

1. Further, White Paper Reading.

This **White Paper** continues onto **Part 2** (see links below), which goes onto to describe the fundamental concepts behind 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: *Reactive Fabric Sample Kit For iOS*
  (employs the **Reactive Fabric SDK**)

4. Contact.

Please direct questions and requests for more information to:

Terry Stillone (*terry@originware.com*)