Reactive Fabric Technology

For Desktop and Mobile

SDK For iOS, OSX (and coming for Linux)

Product Application Brief

March 2019

Originware
Techology

Draft

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

Web: *www.originware.com*

Version: 1.0

1. Introduction

1. Document Content and Context.

This document discusses Reactive Fabric Technology in reference and applicability to the Desktop and Mobile domains. The content delves deeper into the core design principles of Reactive Fabric. The reader will require a grasp of the underlying Reactive Fabric concepts as outlined in the companion document: *Reactive Fabric White-paper*. The subject matter in this document is structured as a series of disparate discussions on various topics associated with software design.

The Mobile platform has grown up. Desktop systems traditionally have had greater application and complexity than its younger Mobile cousin but with advances in mobile CPU and GPU performance that complexity differential has narrowed. As a result, the expectation on Mobile has grown while the expectation on Desktop has kept more or less stable. Expectation equates to application/app complexity and Reactive Fabric addresses the propagating complexity wave by seeking to simplify design and the design process.

2. Reactive Fabric Design Theory.

Reactive Fabric design methodology has a core focus on:

1. The authentic representation of operational process as a design.
2. To promote simple and elegant design through visual graphing of the design.
3. Design decomposition by subsystem and Levels of Detail (LOD).
4. Design evolution through Design Iteration Cycles.
5. To couple design together with implementation so that Design and Implementation mirror each other.

What is the authentic representation of an operational process ? Well, first design is still an art. There are no real correct designs but some designs are more effective than others, more definitive (that indicate what is performed rather than how it is done). Part of the reason is that they reflect the essential nature of the operational process. Essential nature here refers to essential characteristics such as system variances and invariances (variances are the parameters and phases of the system, that modify the system behaviour, invariances on the other hand do not affect the system). It also refers to how transforms on those variances affect the design reflect on how those transforms effect the operational process. For example a "co-ordinate space" used in a graphing system can be a variance and rotating the co-ordinate space (mapping to a new coordinate space) could be a transform on that variance. So the effect of rotating the coordinate space in the design should be reflective of doing that in the actual system.

Simple design follows on to simple implementation, the ease of debugging and testing. In nature, many complex processes may have the appearance of complexity but have an actual underlying simplicity (and elegance). It is the interplay of simple, separate processes that gives the semblance of intricacy and complexity. The magic in the art of design is to synthesize the core, simple sub-processes. That simplicity is expressed in the elegance of the visual graph of the design. Reversing that around, graphing a design is also a process that supports simplification of design. It immediately highlights what is in the way of the design. De-complexifying a design is a process of organising the arrangement of elements that best suits their pathways and then decomposing into separate RfSubsystems. Each RfSubsystem then represents a different Level of Detail (LOD) of the design set.

Designing is a process in itself. Reactive Fabric partitions the process into a series of Design Iterations that evolve the design. The initial iterations drive the concepts behind the system. Those concept designs then drive the practical designs which are backed by implementations. The design process is discussed in more detail later.

Unification is a key aspect of Reactive Fabric. Unification simplifies the subject by not injecting boundaries that then need to be traversed. Reactive Fabric couples Implementation with Design so each mirrors the other. Their separation is reduced, complexity is reduced and there are fewer undulations in the topography to identify and navigate.

1. On The Practical Side: The Reactive Fabric Scoping Mechanism.

Scoping relates to a variance in a system that represents phases or states of the system and that the design RfElements then need to reflect (in order to be authentic). The lifecycle of those system phases also need to be reflected in the associated RfElements in the design.

To put this in a more concrete context, lets take and example: The area of Application Supervision. The hosting process of a mobile app has a number of phases and states:

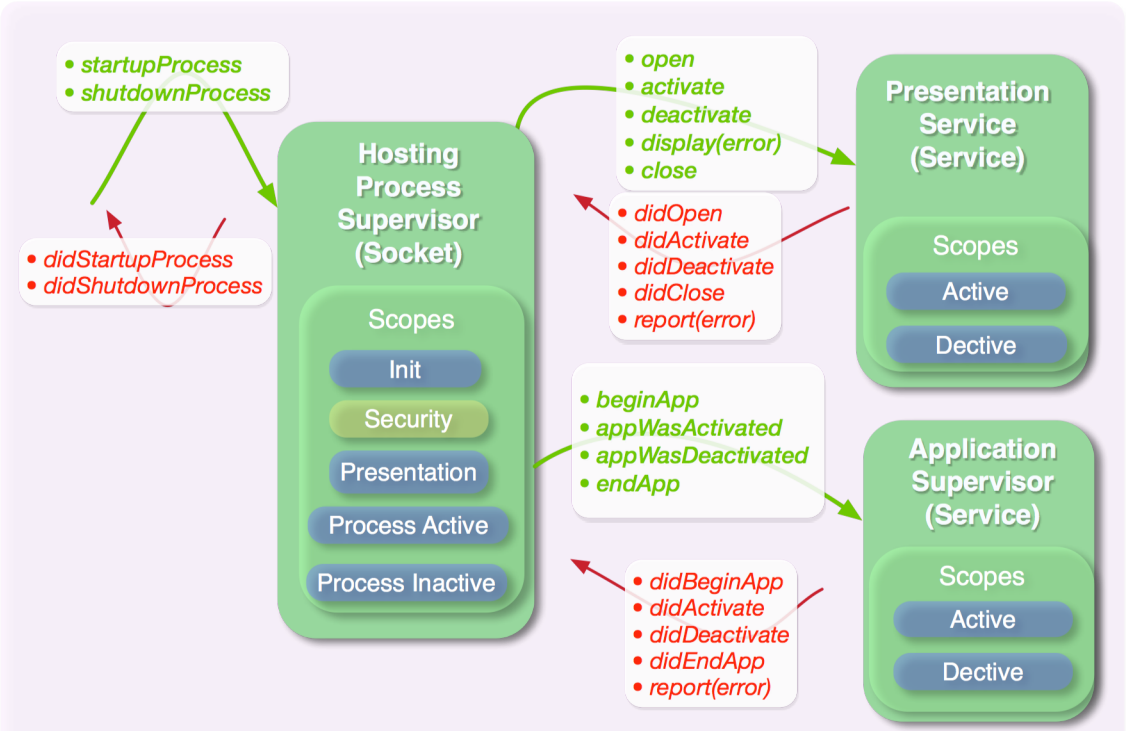
1. Application initialization.
2. Security verification, Permissioning and OS Services availability checking (Optional).
3. OS Presentation System Initialization (View loading).
4. Application Operation (application active)
5. Application Backgrounding (application inactive).

with the associated phase transitions:

- (1) -> (2) -> (3) -> (4), (4) -> (3) -> (2) -> (1)
- (4) -> (5), (5) -> (4)

Thus the design of the Application Supervisor that manages the Application phases over the application lifecycle needs to include those phases. Below is an example of a design that expresses those phases as scopes.

On a side note: In order to reflect the essential nature of the operational process of application supervision the design is decomposed into a *Hosting Process Supervisor*, an *Application Supervisor* and a *Presentation Service*. Part of the reason for the inclusion of the Presentation Supervisor is that it is inter-associated by the active/inactive scopes (phases) that it may need to react to.



Process, Presentation and Application Supervision

1. The Scoping Mechanism (cont.).

Describing the operation of this design:

The Hosting Process Supervisor (RfSocket) contains a scope stack and it also contains a handler for each of its scoping levels. Each scope handler can handle its associated receivable notifications and can run its own role code on initiation. So on initiation: The security scope handler checks Security, permissioning, and required OS Services. The presentation scope starts the Presentation Supervisor by emitting an open notification to the Supervisor and so on. As each scope is pushed and executed, the next scope is pushed and so forth until the Process Active scope is made current. The active scope represents Application execution.

The Process Active scope when initiated, notifies the other two supervisors to enter their active scopes. It also monitors the process backgrounding event and on the event, pushes the Process Inactive scope. Similarly the Process Inactive scope when initiated, notifies the other supervisors to enter their inactive scopes and also monitors the process active event. When monitoring signals an active event, it pops off to the top Inactive scope to make the Active Scope current and notifies the other supervisors. (The sequencing diagram two pages ahead will explain this more succinctly).

The Application and Presentation Supervisors similarly handle their own scope stacks and handlers.

Reactive Fabric uses sequence marble diagrams are used to depict notification sequencing. The sequence diagram for this system is given on the next two pages.

This design goes further, it also compasses error handling for all the Supervisors.

The Application and Presentation Services can report their observed errors to the Hosting Process Supervisor using the report(error) reply. The Hosting Process Supervisor determines the severity of the error and in turn performs the appropriate action:

- For fatal errors it coordinates shutdown by issuing an endApp notification to the Application Supervisor then, popping down to the presentation scope and emitting a close notification to the Presentation Supervisor and then popping down to its Init scope and emitting a didShutdownProcess reply.

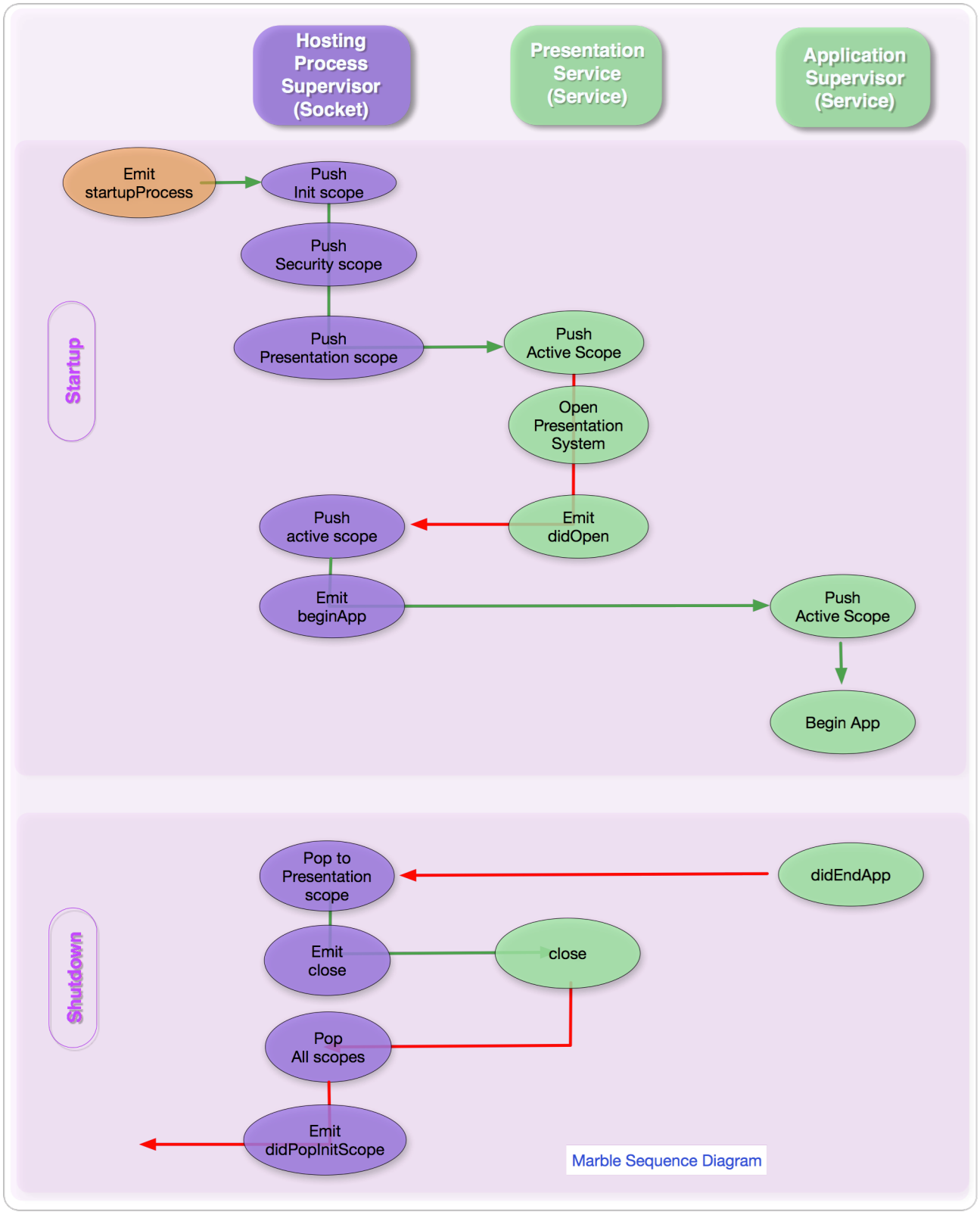
- For non-fatal errors, it can trigger the emission of a display(error) notification to the Presentation Supervisor for the presentation of the error to the user.

Scoping is also extensively used in network protocol stack design. The scope reflects the protocol state. For Bluetooth protocol it can represent the phases of peripheral discovery, service discovery and session. For TCP/IP depicts closed and open connection session states.

Scoping is generally based on a bounded discrete data space and represents a phase or phase level variance but there is no reason why it could not be unbounded or a continuous data space.

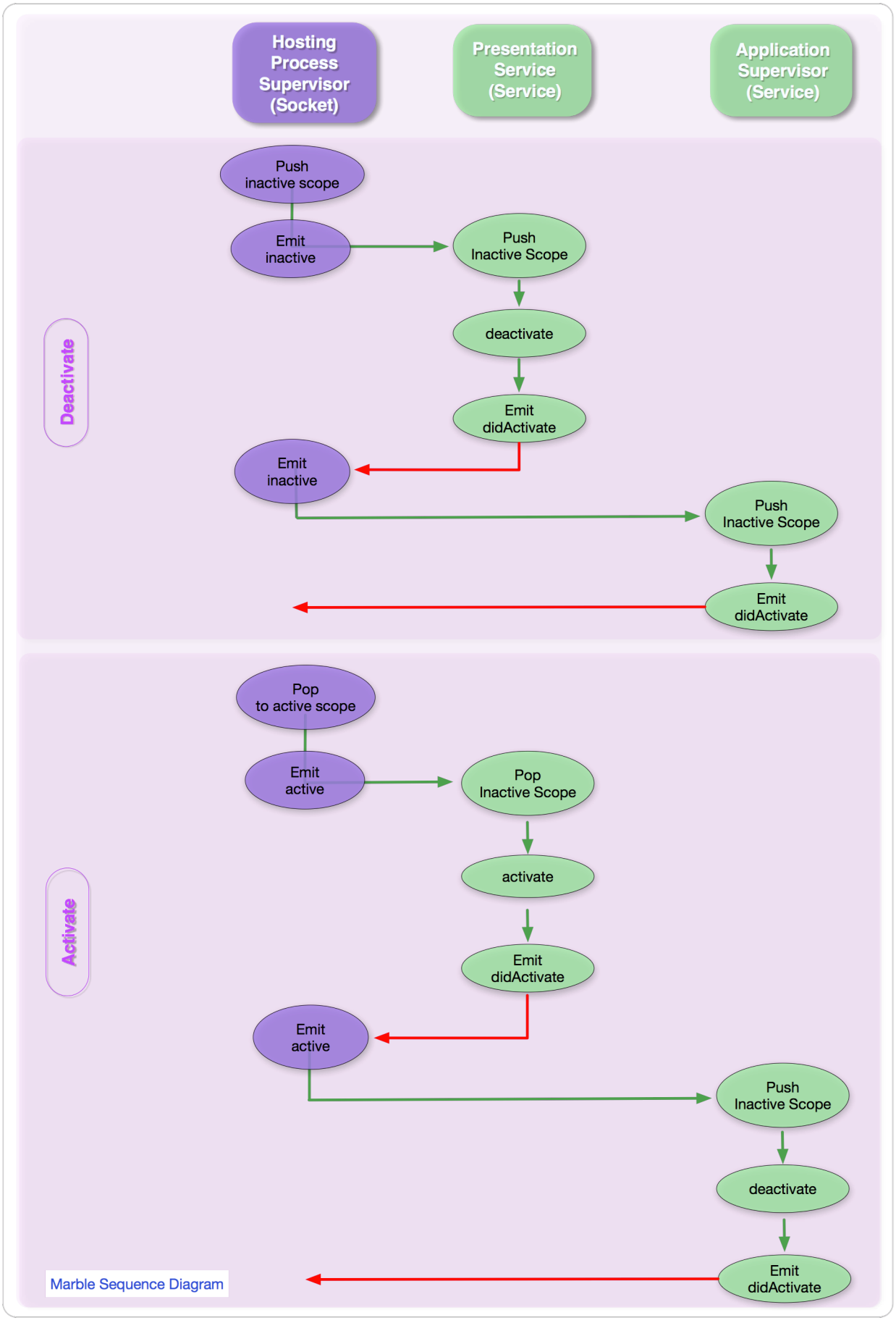
1. The Scoping Mechanism (Cont.)

Here is the marble sequence diagram for the major Application supervision interactions. (Error handling is not included):



1. The Scoping Mechanism (Cont.)

Here is the marble sequence diagram for active and deactive scope interaction:



3. The Design Process.

1. Synthesizing Design RfElements.

There is a basic outline for the process for synthesising the RfElements and RfNotifications:

- Identify boundary elements, their associated notifications and associated data-types.
- Extend post processing for input boundary elements with new RfElements.
- Extend pre-processing for output boundary elements with new RfElements.
- Identify required infrastructure and synthesize their RfElements. Infrastructure includes:
  - System management.
  - Other OS Services the system will require.
  - Pure internal RfServices that will benefit system decomposition.
- Identify the pathways and processing required for input notifications and how they trigger output notifications. This forms the application processing logic.

Boundary Elements can be categorised into:

- Simple one way inputs (input text, mouse position, touch/clicks, location etc)
- Simple one way outputs (graphic element drawing, graphic text, sounds)
- Two-way interactions (such a networking, peer interactions, etc)

Generally, one-way inputs are expressed as RfSouces, one-way outputs are expressed as RfCollectors and two-way interactions as RfServices. Once boundary elements are synthesized, their secondaries are identified by asking questions as to what post processing of inputs and pre-processing for outputs are required.

Infrastructure is important to be considered, but may not be expressed in initial designs. Application logic is the more important characteristic to be captured in initial rounds.

2. Design Iterations.

The first design iteration delivers a purely conceptual version of the system. It captures the essential nature of the application logic. Practicality is not a focus in this initial version. Rather, it is intended to be a spring board for the later practical designs and a reference base to allocate work groups and who works on what design sections.

To get to the resultant bare conceptual design you may need to create a first pass design with over detail which is then pruned down to a bare conceptual form. The over-design may include all the inputs and outputs which is fine, as long as they are pared down for the final conceptual version.

The second iteration design forms the first practical design. A backing implementation will most likely be authored for this version (and probably not for the first iteration design). Brain storming design sessions are a very useful exercise. Some designs end up being non-practical explorations which then add knowledge context to the group. Graphing the design is important to exhibit what is in the way of the design and shines the light to simpler alternatives. Graphing assists in partitioning large designs into separate Subsystems and LODs.

From the first iteration conceptual design working groups can be formed to work on subsystems/ sections of the design. Those groups then work on their respective areas and derive their second iteration practical designs (for their assigned sections). They then come back to the group design sessions with their proposals for integration into the whole design.

LODs are typically sublimated in the second iteration and then clarified in successive iterations.

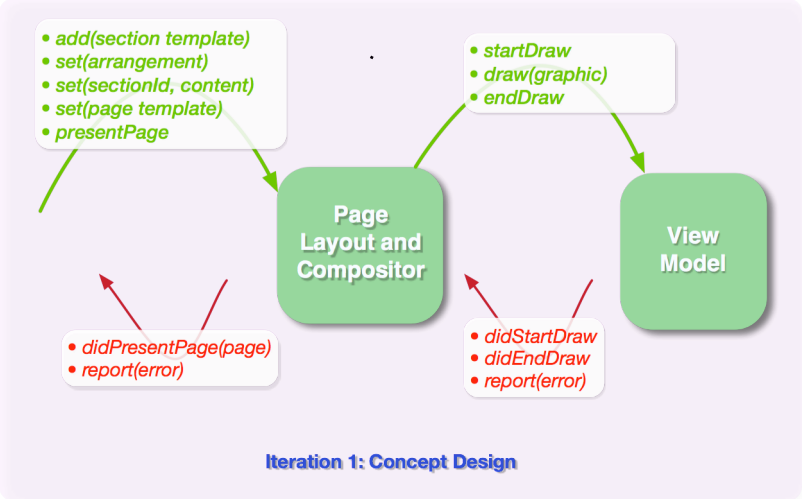
4. An Example of a Design Iteration Series.

Iteration 1: Concept Design

Example design description: A graphic layout expression which is given section templates which are to be populated with content, laid out according to a given section arrangement and then composited onto a given page template.

The concept design, here to the right depicts the essential information of:

- The notifications required to be supplied to the expression.
- The essential function is to bind content, arrange and composite sections onto a page template.
- The compositor drives a View Model which adapts application virtual drawing commands to OS drawing commands so that it is decoupled from any particular Drawing System.

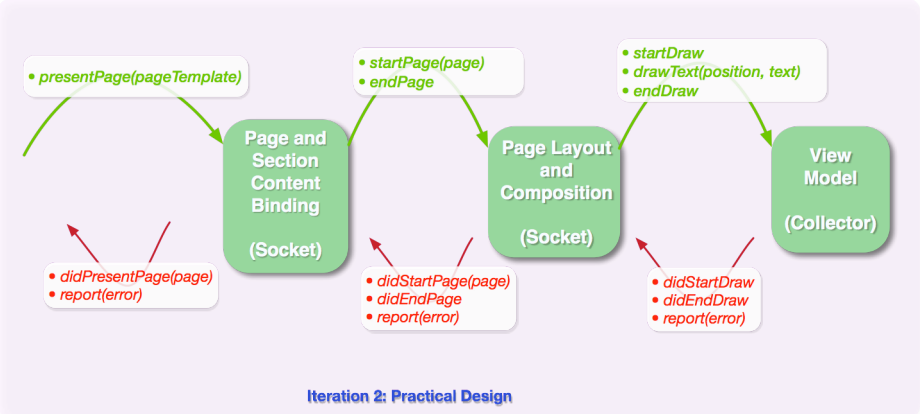


Iteration 1: Concept Design

Iteration 2: Minimal Practical Design.

The first practical design here to the right, only operates on a page template (and not on sections yet) and only generates textual content. The page generated from the page template is emitted to the view model.

It identifies that the page layout and compositing would be simpler if broken into two sequential operations. One for binding content to the pages and sections and another for layout and composting.



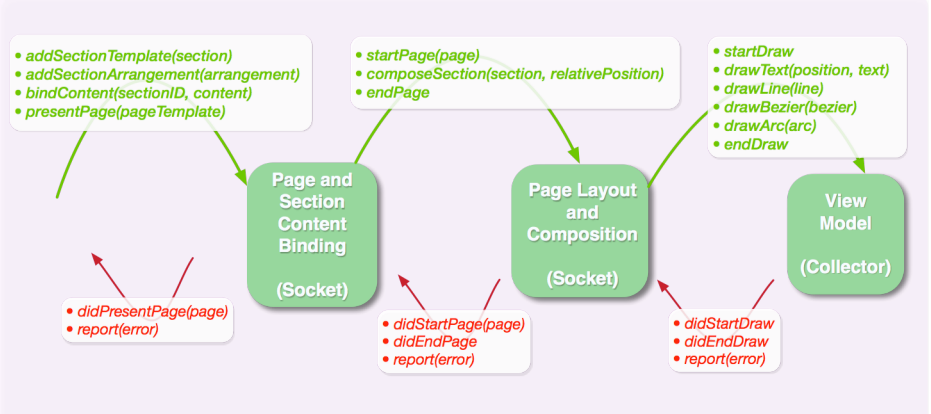
Iteration 2: Practical Design

The minimality in the design is chosen in order to get the backing implementation up and running and so that this expression can be plugged into the larger system and allow other dependant design implementations to begin functioning.

Iteration 3: Final Practical Design.

The final design adds more detail by adding section handling, section layout and compositing together and with the capability to generate all drawing operations.

This design adds more capability to the Iteration 2 implementation without structurally changing the topology of the design.



Iteration 3: Final Design

3. Fabric Expression Evaluation (i.e. Execution).

1. Theading and Processing Models (Synchronous and Asynchronous)

Reactive Fabric evaluation can be performed synchronously or asynchronously. In contrast, the pure RfElement/RfNotification design level that we have been discussing is essentially invariant to the processing model.

A Reactive Fabric expression is collection RfElements which have been composed (to define their pathways between each other) and then is called to evaluate. Evaluation is conducted in three possible modes:

- Inline evaluation (the calling thread is used for evaluation).
- Sync evaluation (a new thread is used for evaluation but the caller is blocked).
- Async evaluation (a new thread is used for evaluation and the caller is not blocked

So in a fabric, you can choose which parts (expressions) are operated asynchronously and which are synchronous. Generally, for production design you will just use async evaluation but for testing you may find circumstances where you want to block the test thread, run the evaluation and then verify results in the test thread when evaluation is complete.

Where production code is required to run in a particular system thread (for example in the UI thread to display) you will need the particular RfElement notification handler to dispatch off onto the target thread to run the target code.

2. Debugging Reactive Fabric in a Debugger.

Reactive Fabric has a different executional signature than typical programs, it is more stack heavy. This arises because notification emission is represented as a function call, so an evaluation of an expression is a series of function calls. As most expressions are run async (in a separate thread), the thread stack for an expression will be length of the expression but when expressions are synchronously coupled they can extend thread stack depth.

There are a variety of debugging techniques, but debugging with a debugger is the first option for immediate observation. This entails setting a break point in an element notification handler and then observing the stack to peruse the call chain of the propagated notification. Setting the breakpoint in the ending element of an expression will yield a stack trace going through the whole expression. This ends up being quite convenient as all captured variables are available throughout the call chain in the debugger.

3. Other Debugging Techniques.

For observing a long chain of events (that the debugger may be less effective at) the Reactive Fabric SDK is instrumented for tracing. Notification activity is trace-able to a log-able interface such as a system logging service or a print function. Trace enabling is as simple as calling a trace() method on an RfExpression. Tracing can generate complete expression notification activity or specific element activity.

The final debugging option is to temporarily place dummy observer elements in a target expression and observe notification activity directly in code form. This is generally the heavy weight option but useful for custom observation, processing and reporting.

5.Benefits of Reactive Fabric Architecture

1. Benefits to Development.

Reactive Fabric re-scopes capability in the development process. It re-scopes what is possible and how that is attained. It precipitates through greater group and personal process to visualise design, express in design, collaborate through design and explore with design. Project members grow in themselves in the design space it provides.

The flow on, permeates into the individual and group development process, the quality, flexibility and testability of the code base and assurance in the operation of the code. More specifically:

- The individual development process is enhanced with scope to do more at less personal cognitive burden (i.e. the individual does not need to pour over tens of thousands of lines of code in order to understand system operation). The individual is given greater scope to express and demonstrate the trade offs of one design over another.

- The group development process is sped up with more effective, more frequent and tighter collaboration cycles.

Reactive Fabric relies on and comes at the cost of the group shifting to a different mind set, shifting into the design space, looking at the problem in a different way, letting go of old safety blankets (e.g. "*I write large complex systems, design just slows me down and I don't have time for it.").* It does take acclamation to start breathing, thinking and working in the design space. The net result is benefit but people may require time to integrate it into themselves, into their being where their creation springs from.

2. Benefits to Testing and Simulation.

Good design should also equate to testability and testability flows back into good design. The easier it is to design testing systems the more assurance there is in the design.

Reactive Fabric brings the design space into the testing and simulation domain, so that development and testing are codependent threads. The benefits of Reactive Fabric to development also become the benefits to testing. Testing engineers get see the design of what they are testing. They couple into the design of test systems.

3. For More Information.

Please check the:

- Originware site: *www.originware.com*
- The Reactive Fabric web page: *www.originware.com/reactivefabric.html*
- The Reactive Fabric white paper: *www.originware.com/doc/Reactive%20Fabric%20Whitepaper.pdf*

For direct questions and information contact: Terry Stillone (*terry@originware.com*)