Reactive Fabric Technology

For IOT

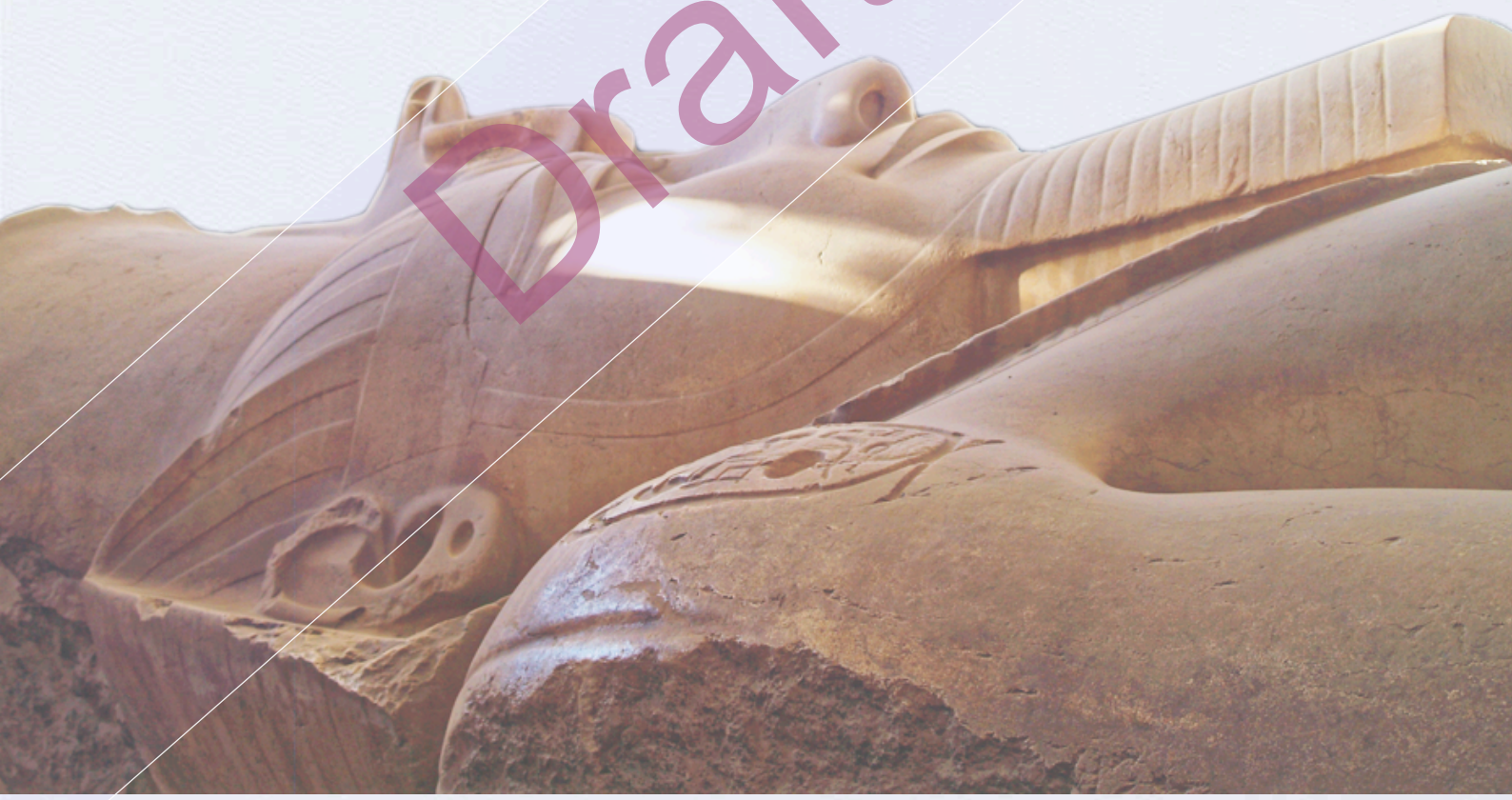
SDK For iOS, OSX (and coming for Linux)

Product Application Brief

February 2019

Originware Techology

Draft



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

Web: *www.originware.com*

Version: 1.0

1. Introduction

1. Required Background Reading.

This document discusses Reactive Fabric Technology in reference and applicability to the IOT domain. You do need to understand the underlying concepts of Reactive Fabric and so you are advised to read the companion *Reactive Fabric White-paper* first.

2. The Reactive Fabric Technology Platform For IOT

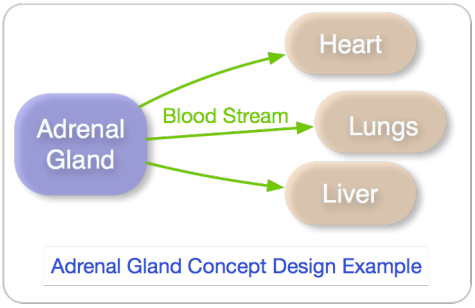
IOT has realised an expansive growth phase and as a follow on consequence, requirements previously on the broad horizon have been drawn to the door step. More intricate peer interactions, deeper, more varied online dependencies, edge boundaries and correspondingly expansive operational complexity. Reactive Fabric brings advances in Software Technology normally available to the more sophisticated desktop base to the SOC platform, bringing enhanced capability while simplifying design and development.

The IOT concept is founded upon premise of the value of local "interchange" (interaction between devices). Reactive Fabric is all about the design of interaction and interchange. It structures design into a hierarchy of Levels Of Detail (LOD) that represent various levels of interchange. From macro LOD interaction across mass devices down to the micro LOD controlling a simple sensor. It organises the design of interaction into a manageable visual medium. As a common design process, it brings projects and groups together to collaborate, iteratively evolve and share their software design and design process. The Reactive Fabric design methodology brings people together, gives them a human language base to describe design and in turn, unburdens their (design) cognitive load, so as to ponder deeper possibilities.

Reactive Fabric unifies in the total dimensionality of the development process. As a common Software Architecture that applies to the whole vertical platform stack, It unifies the total software base from SOC to Mobile App, to Desktop program and even to the Server level. On a group process level, it brings members together to design together. It encourages general design and developmental interchange between all interested parties (not just group members).

3. Reactive Fabric as a Design Medium.

Nature employs messaging as a design principle: some component generates encapsulated message quanta, which in turn follows a transport path and ultimately is consumed by a receiving component. A good example of this are chemical messengers in organisms, sent through arterial pathways to be acted upon by target organs. Here is an Adrenal gland example in diagram form.



Adrenal Gland Concept Design Example

Reactive Fabric takes that natural design principle and applies it to software interaction. Here RfElements produce, consume and operate on message quanta (called RfNotifications). In the diagram analog, an RfElement (of type RfSource) corresponds to the Adrenal gland secreting Adrenaline (the analog of the RfNotification data quanta) which then is transported to separate target RfElements (of type RfCollector) in this case the heart, lungs and liver.

An RfNotification is an event data-quanta. Examples include: a network data packet with the packet data, a heart rate reading with the reading timestamp and the sensor ID that took the measurement, a peer discovery event with the peer identity, a peer Bluetooth Descriptor value change event with the descriptor identity together with the before and after values.

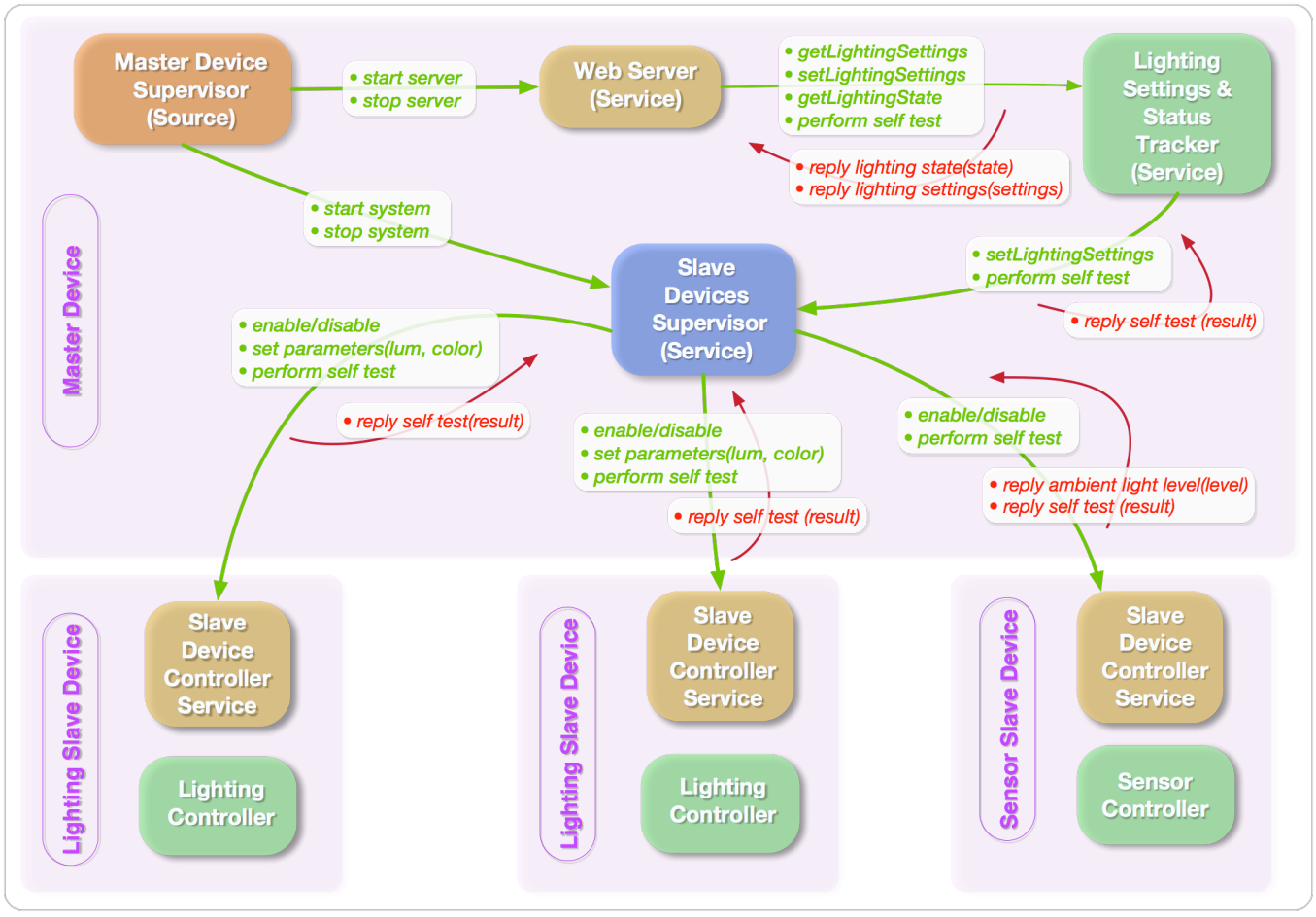
Reactive Fabric design describes operation in terms of processing elements (i.e. RfElements), RfNotifications (the messengers) and their transport paths between those elements. The connected graph of RfElements forms the topology of the interactions and this structured organisation embodies the "Fabric" which encapsulates the whole system operation.

1. Reactive Fabric Design and Implementation Iteration Cycles.

Designs describe interaction in the abstract. The space of the design is not confined. It can focus down to an LOD of simple controller of a embedded device, it can go to the group interaction of a device/peer/server collective, it can depict a whole network, it can go beyond. The LOD of designs are relative.

The design, also only needs to depict what is relevant for that LOD. High level LODs tend to be conceptual in the beginning and then evolve to more practicality. A Design must balance detail with relevance. Too much detail, too much complexity reflect an immature design. Simplicity and elegance are the goals.

Below is a concept, high level LOD design of a lighting system with a master device and small slave devices. The master device includes a mini web-server to act as a user interface. The design implies wireless connectivity between the master device and slave devices but there is not relevant for a high level LOD and would be depicted in lower level designs. The Web Server also has much more internal detail and this would be depicted in the low level LOD designs.



Designs evolve together with their co-dependent backing implementation. In the same way implementations go through iterations, so do designs. They grow, mature and refine along side their complimentary implementation. For example, the Web Server low level design would begin first with just a service initialisation/finalisation design, then incorporate net connectivity with HTTP GET protocol handling and then onto POST/PUT protocol handling and so on.

Low level LOD designs tend to detail more of the "how" whereas high level designs describe more of the "what" (of the interactions).

2. Simulation and Testing

1. General Testing and Simulation.

The interaction capability unlocked by Reactive Fabric also follows over onto greater simulation and testing capability. The multi-device nature of IOT adds even more complexity to interaction mix when device-collective testing and simulation becomes into play.

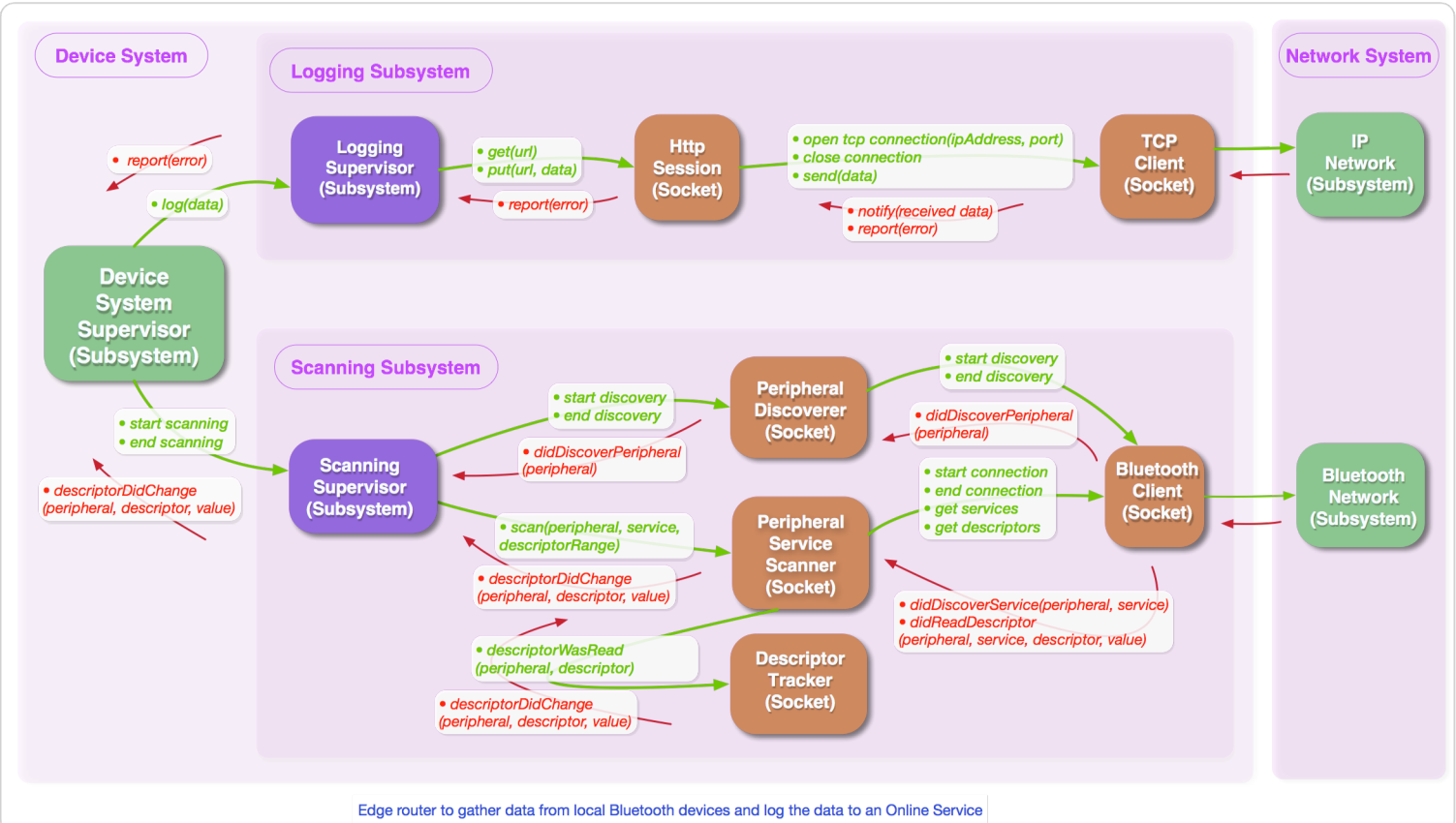
Reactive Fabric provides for testing at a number of scopes:

- Unit Testing of individual RfElements.
- Behavioural Testing of single systems (single devices).
- Collective Behavioural Testing (device collectives).

Typically, test rigs are authored to generate input RfNotifications to target testables and authored RfCollectors or RfServices to consume the resultant products (notifications) and verify their legitimacy.

Let's use an example design as a reference for testing. Below is a lower level (in device) design example for an edge router. It uses Bluetooth to scan for known peripherals in the locality, extracts their data (in the form of Bluetooth descriptors) and then logs the data to a net service when those values are found to change.

It comprises of two subsystems, one for scanning Bluetooth devices and the other for logging. The Supervisor in notified of changed descriptor values and then emits to the Logging Subsystem for the data logging.

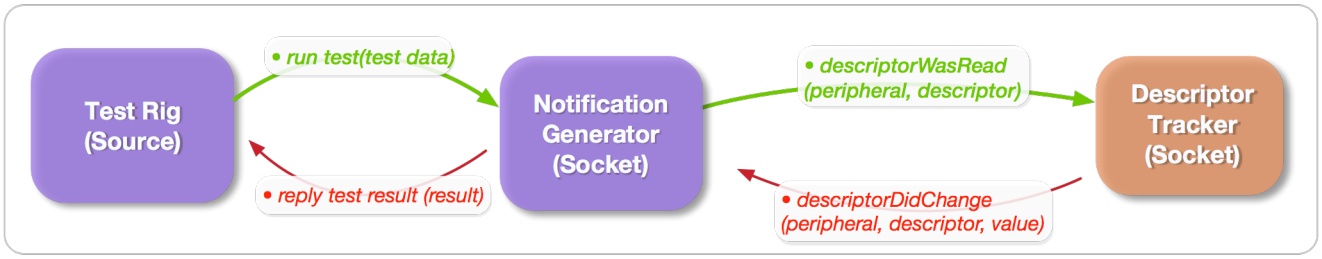


Edge router to gather data from local Bluetooth devices and log the data to an Online Service

2. Simulation and Testing

1. Unit Testing (Single RfElements).

On the simplest level, RfElements can be unit tested in a simple rig to organise the test data into test quanta (RfNotifications) and deliver that to a notification generator/matcher RfElement. Here is an example test rig for exercising the Descriptor tracker in our base design:

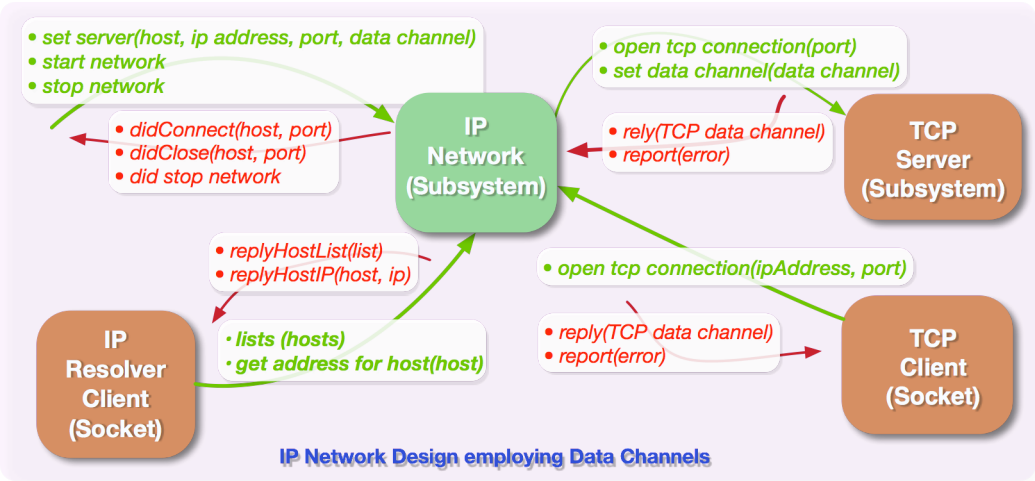


2. System Behavioural Testing.

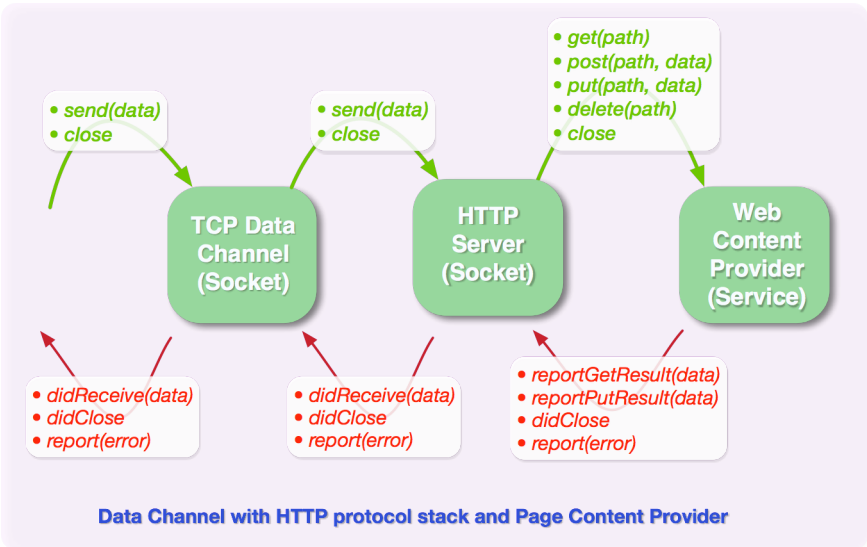
In this scenario we again have to generate input and verify output RfNotifications but it requires more coordination between more numerous generators and verifiers. In our base example the inputs and outputs are network interfaces so, there are two main avenues of test rig architectures:

1. Pure in-device testing architecture which would feed and verify to/from the client interfaces.
2. Multi-device testing architecture which would simulate the networks themselves and lead onto the capability of multi-device testing and simulation.

We will follow the more extensive operation option (2) and incorporate network design but only cover the IP Network. The IP Network design is given here and rather than running read and write requests through the whole Network we will use a Data Channel design. The Network open requests reply back with data channels which in turn perform data interchange.

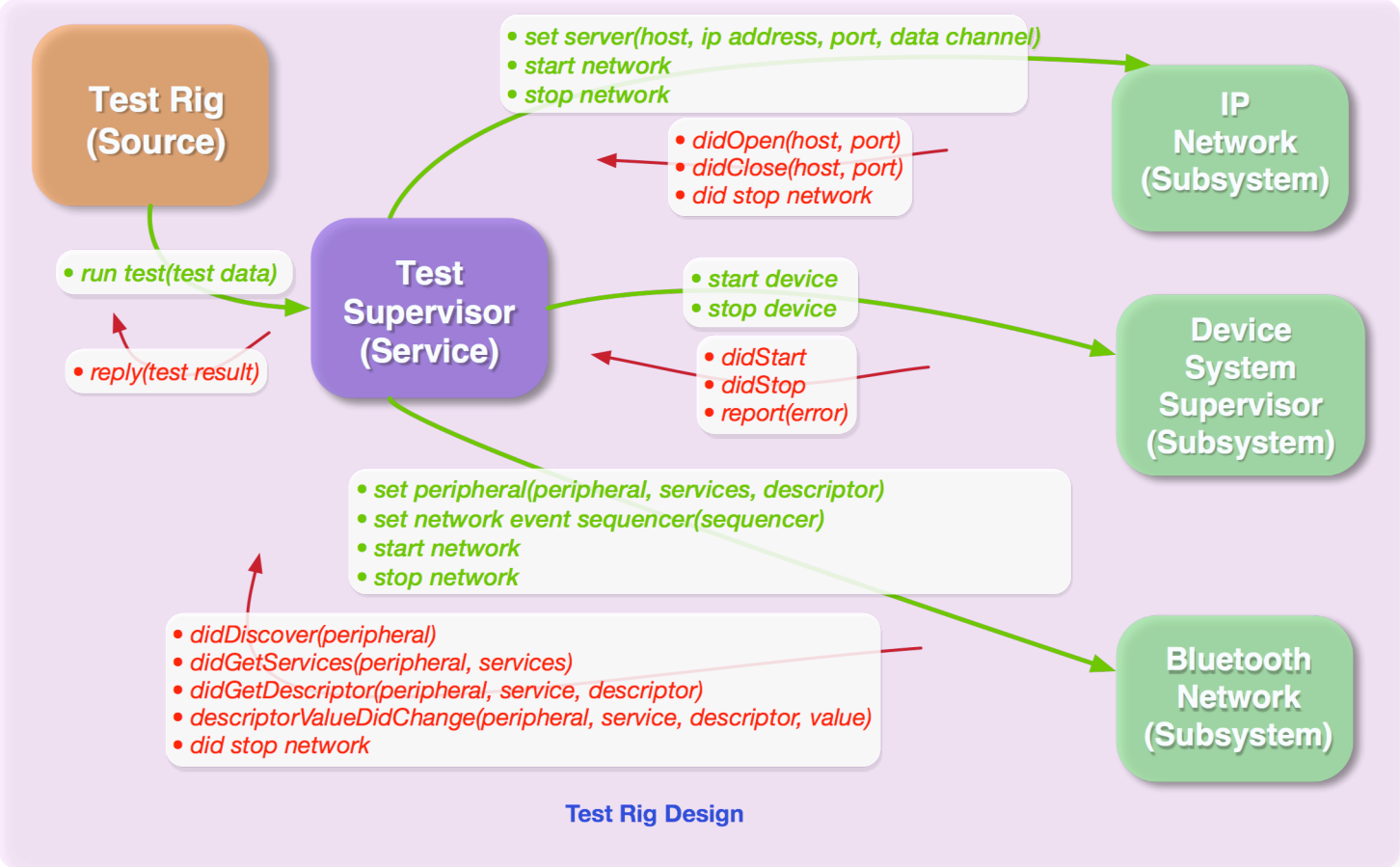


The Test Rig supervisor will have the responsibility of creating the data channels, populating Web Content within them and installing the data channel into the TCP Server RfElement.



2. System Behavioural Testing (cont).

Here is the Test Rig design:



Test Rig Design

The Test Rig emits (to the Test Supervisor) the test data comprising:

- The collection of TCP Servers with their data channels.
- The collection of Bluetooth peripherals with their descriptors.
- The Bluetooth network sequencer which drives the bluetooth events (discovery, descriptor change, etc).

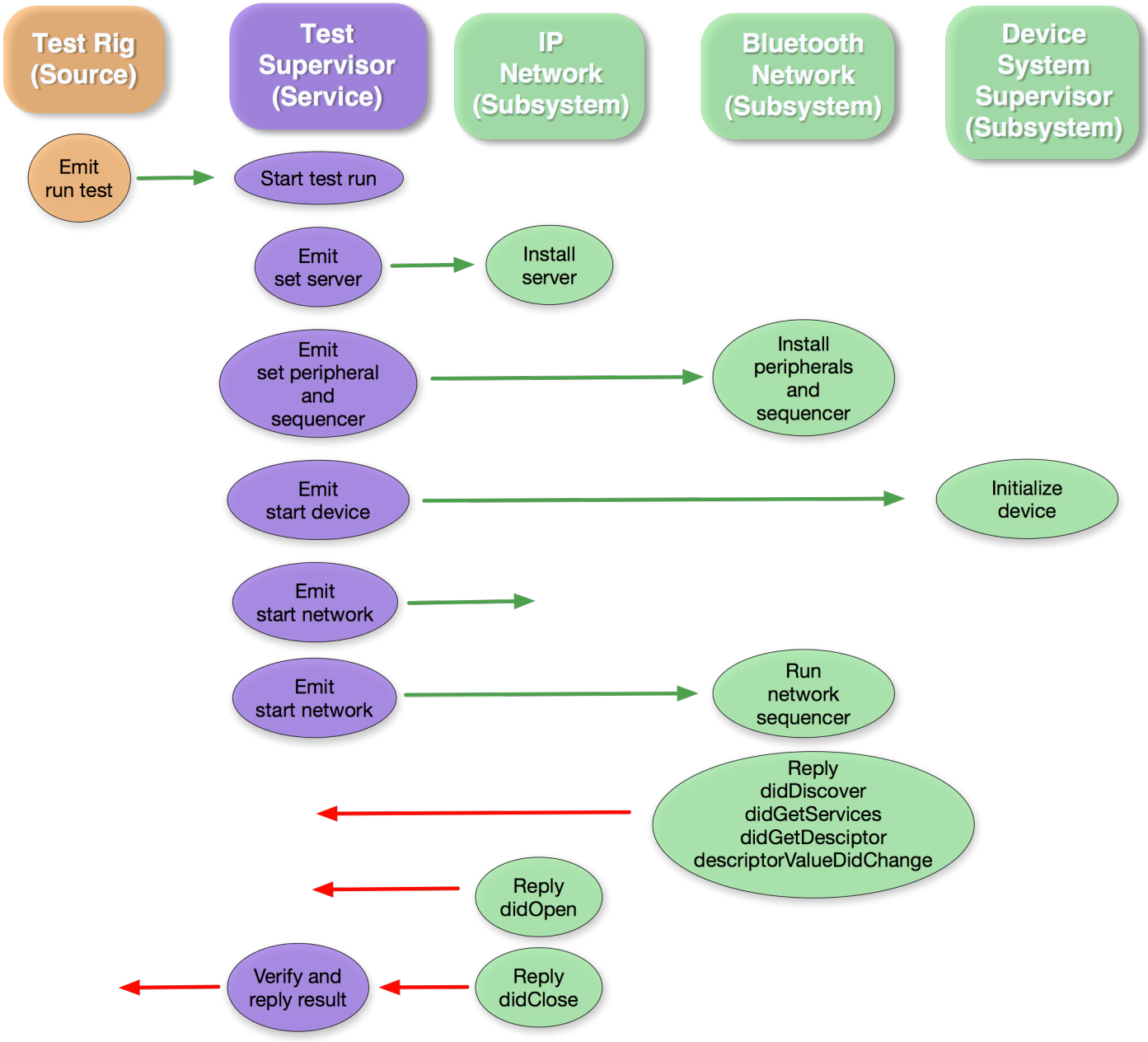
The Test Supervisor in turn configures the networks from the supplied test data. The test is run, the network events are recorded and the cycle completes when the all the IP hosts close (as observed through back replies from the networks). The sequence of recorded events are verified and the data channels are verified. The test result is emitted back to the Test Rig.

For more fine grained testing, the data channels could record their event activity and verification could then use that rather than just their resultant state. For testing temporal variations, the bluetooth event sequencer can be perturbed and it would then be expected to generate the expected whole system events used for verification.

This is a single test target design, the Test Supervisor creates a single the target Router Device. For a multi-device test, the Test Rig test data would include the target devices and the Test Supervisor would in turn construct all of them.

3. Sequence Diagramming

Sequence diagrams (also called marble diagrams) are useful for conveying the sequential order of notifications and replies in designs. Here is the sequence for the previous Test Rig:



Marble Sequence Diagram

Of course, green arrows are request notifications and replies are in red.

3.Benefits of Reactive Fabric

1. Benefits to Development.

Reactive Fabric re-scopes capability in the development process. It re-scopes what is possible and how that is done. 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 space it provides.

The flow on permeates into the individuals development process, the 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 (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 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*)