How ION uses Virgo

Charles Vuijst

Disclaimer

This document reflects the direction Infor may take with regard to the specific product(s) described in this document, all of which is subject to change by Infor in its sole discretion, with or without notice to you. This document is not a commitment to you in any way and you should not rely on this document or any of its content in making any decision. Infor is not committing to develop or deliver anything described herein.

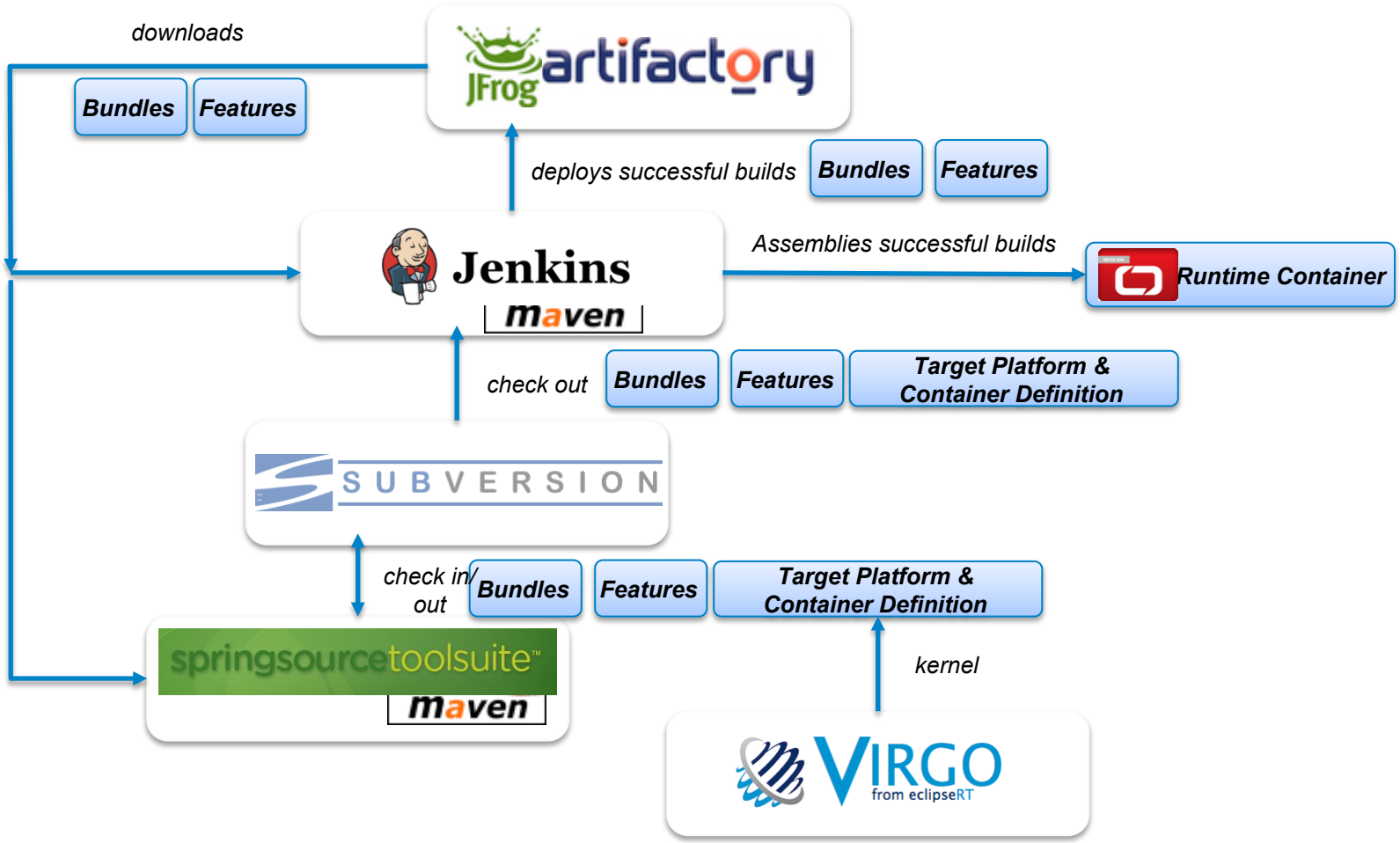
Overview

- Virgo 2.1.0.RELEASE Adoption
- ION Runtime Development – The Numbers
- ION Runtime Development Cycle
- Stack Comparison Virgo 2.1.0.RELEASE vs. ION Runtime
- How we use OSGi
- How we use Eclipse
- The Infor ION Case
- Architecture
- Future Thoughts, Feedback, Wishes

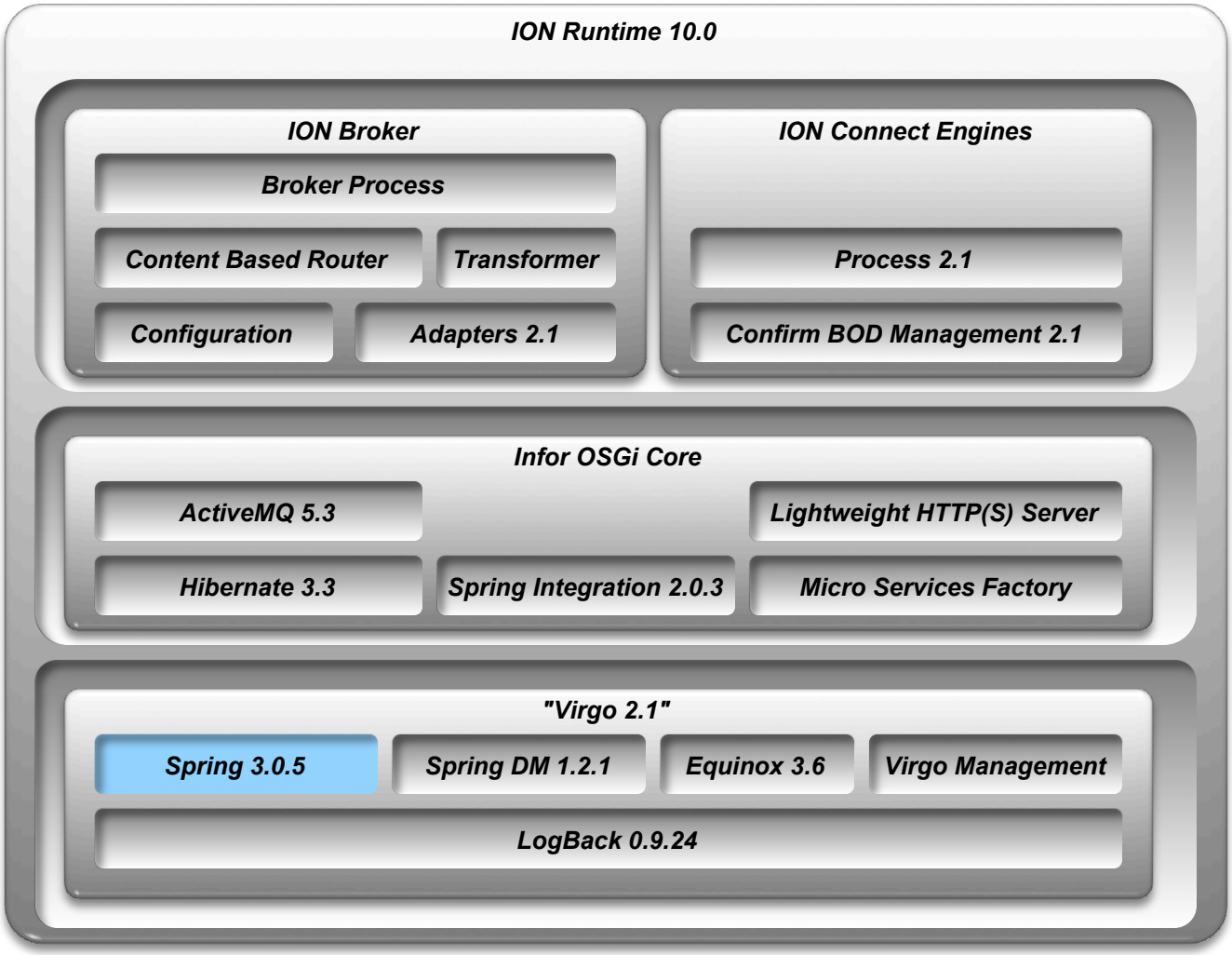
Virgo 2.1.0.RELEASE Adoption

- From Equinox 3.5 to Virgo 2.1.0.RELEASE using *Kernel* only
- Added a Lightweight (Secured) Web Server
- Added Windows Service / Linux Daemon for startup
- Added Spring Integration
  - Limited, due to inherent “static” context model
- Modified Virgo Kernel Stack
  - Replaced Spring 3.0.0 with 3.0.5 for Spring Integration
- Added a Watched Repository Folder
  - for Dynamic Database Vendor plugins with plan files
- Integrations with JDBC, Web Services (SOAP, REST), JMS, JCA, XMPP, File system

ION Runtime Development Cycle



Stack Comparison
Virgo 2.1.0.RELEASE vs. ION Runtime

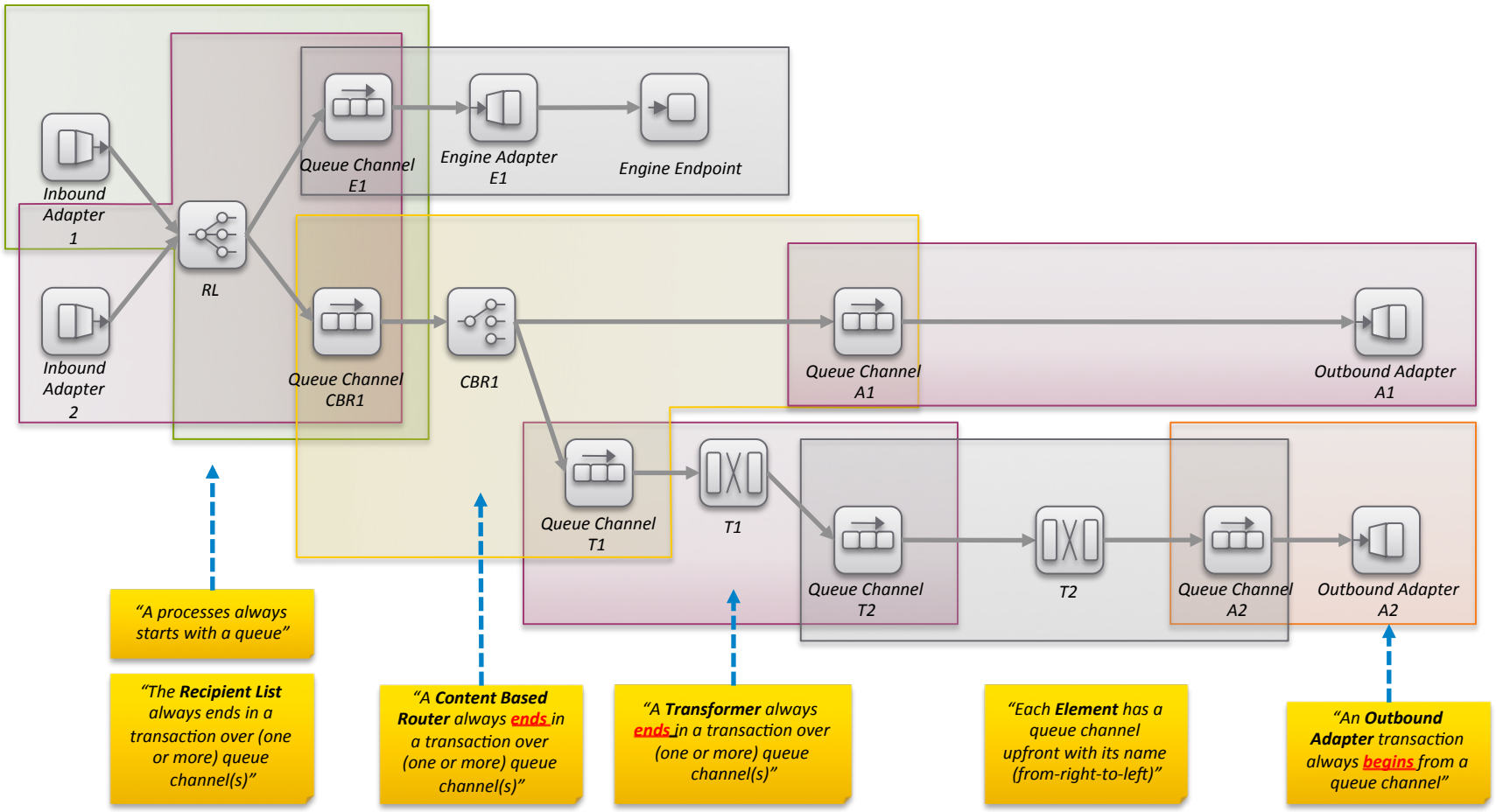


OSGi

- Spring DM bundles preference
  - *"Let Spring do the repeating work for you"*
- Whiteboard Pattern
  - *"Everyting is a service"*
- Micro Services
  - *"Static vs. Dynamic Spring Contexts: Singletons vs Prototypes"*
  - Extracted Configuration object (also prototype)
  - Prototypes currently require container restart
  - Note on OSGi bundle replacement...
- Maven vs. Bundles & Features
  - Hybrid development model: Bundles & Features are Maven projects
    - Compile & generate "OSGi-ish" artifacts using transitive dependencies
  - Build on central build server/slaves and local development systems
  - Using Maven plugin extension model to generate Bundles & Features

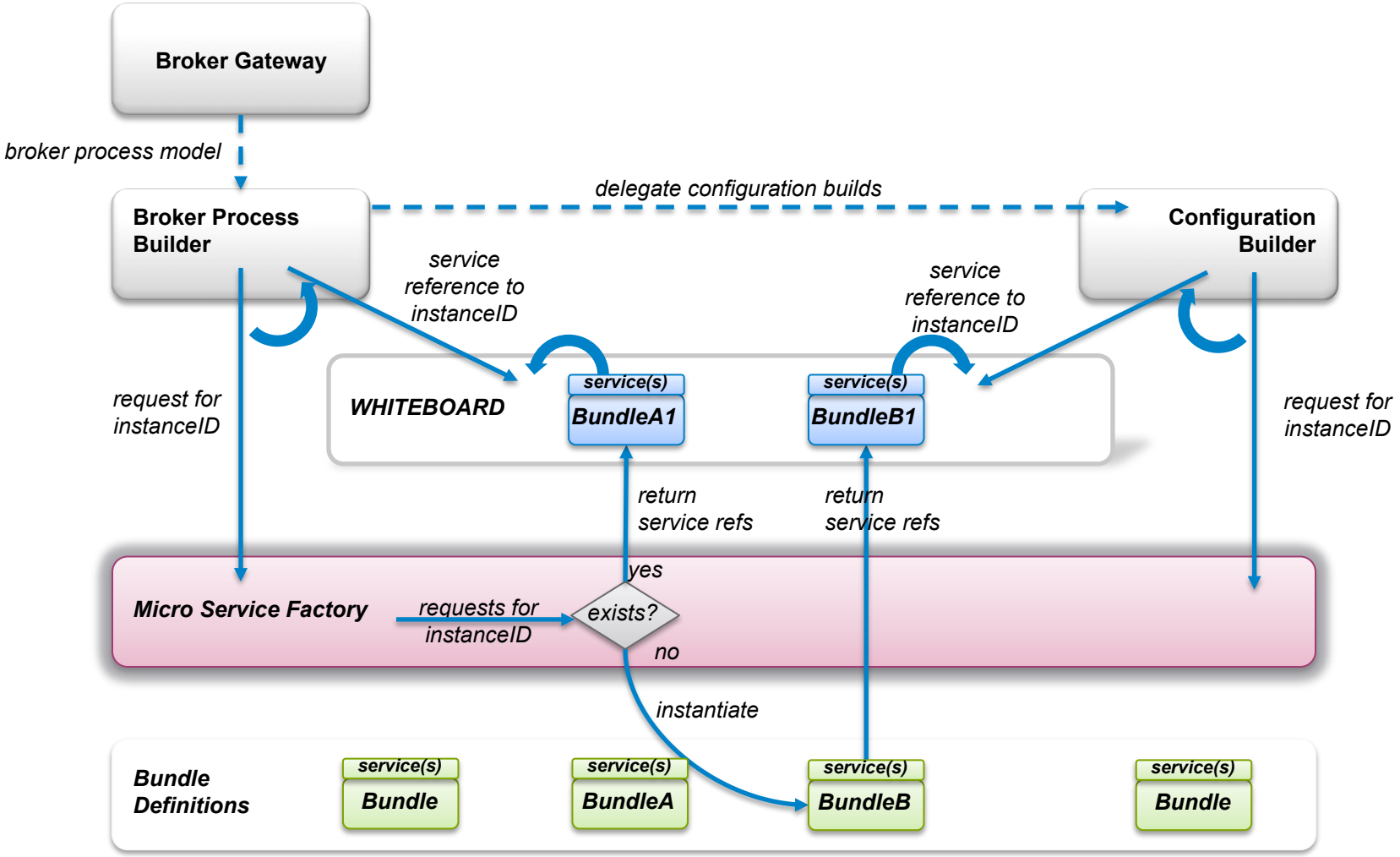
The Infor ION Case
Broker Process Model

"From Singletons to Prototypes with Transactions"

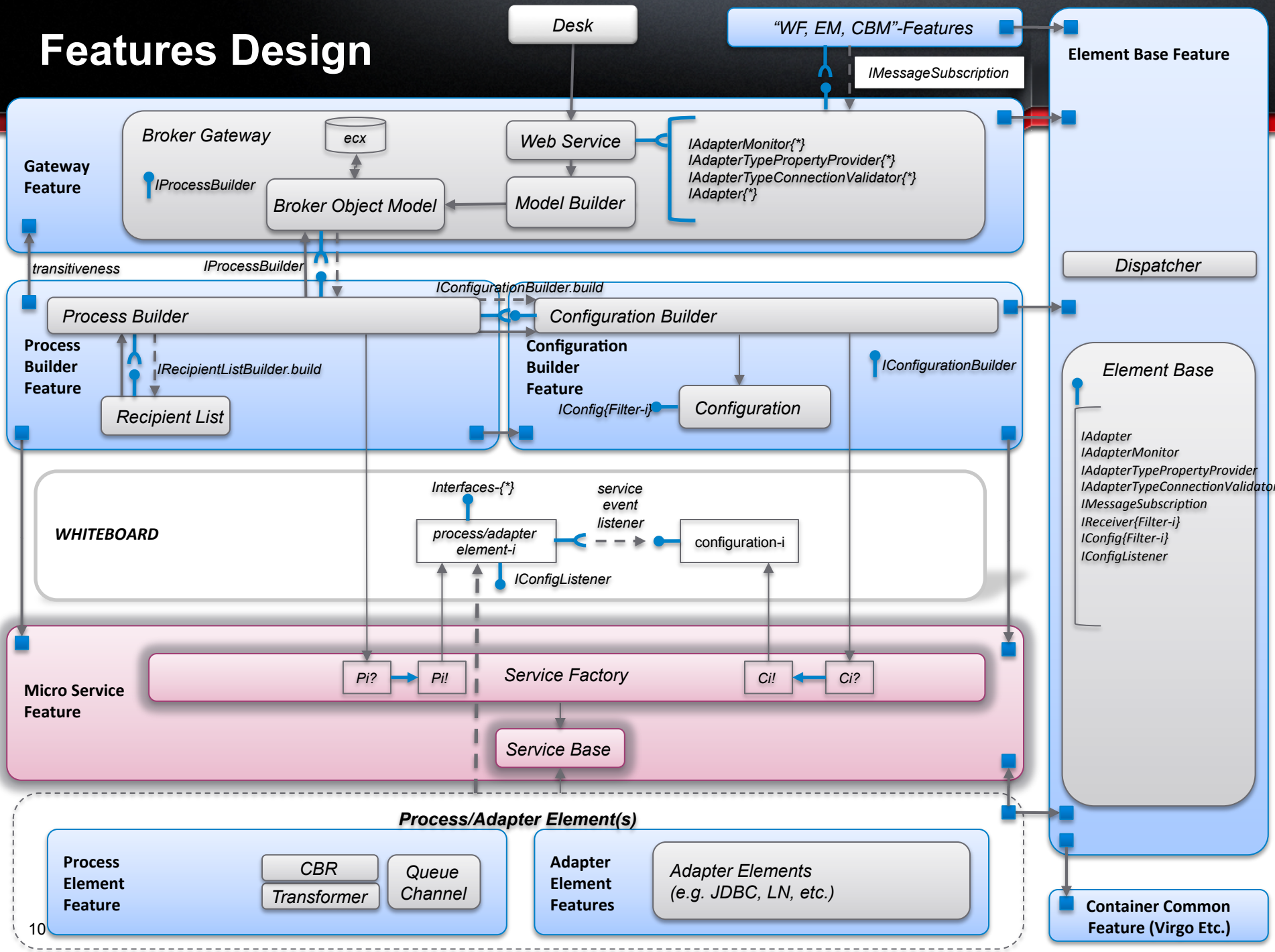


Micro Service Factory

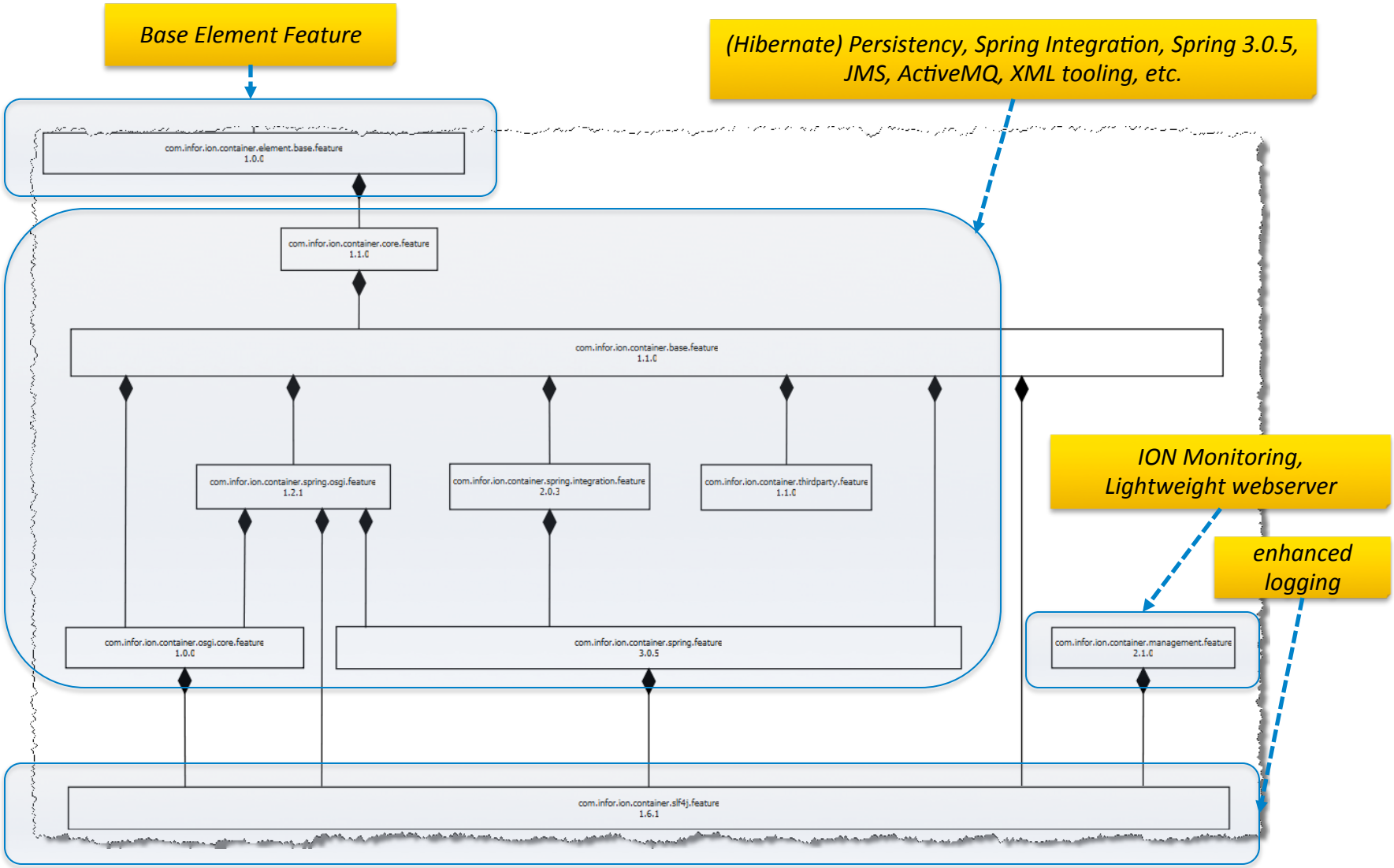
"Prototypes as Services"



Features Design



Remainder of ION Runtime Features



Lightweight (Secured) Web Server

- Web server that doesn't involve application descriptors, servlet container, etc.
- Based on JRE's internal HTTP(S) Server
- Wrapped as Spring DM Bundle
- Exposing JAX-WS based web services through WSDL
- Using self signed SSL certificates: only pipe security
- Seamless integration with .NET WCF
- Using Mutual Authentication based on SSL certificates
- Silent securing .NET 2 OSGi container
- Independent of platform (Windows/Linux) for Virgo

Eclipse

- Using Spring Source Toolsuite (STS) latest version
  - Run/Test/Debug Virgo Server internally in Eclipse environment
- Using "External Launcher" with Maven
  - All builds require to run & test headlessly on Jenkins as well
- Maven Builds
  - Features
  - Spring DM (OSGi) Bundles
  - Custom Maven plugins generates feature.xml,
  - Custom Maven plugin generates Manifest (using BND tool)
  - Target Platform, fully feature based (update site ready)
  - Container Assembly, merging Virgo essentials with Target Platform
- No M2Eclipse
  - Need to run on Jenkins

Feedback

- Steep learning curve
- Headless Integration Test framework required
- Automatic detection of deadlocks rocks!
- Using modified User Region: Virgo Plan Pipeline Resolver is too slow
  - "Any hints"?

Whishes

- User Regions like Plan Files: Version ranges configured
- Plan file Loading sequence instead of alphabetically
- Plan files still required: "Import Bundle" statement
- Dependency tooling would be great to generate overview pictures
- Eclipse + Maven3.0/Tycho + Jenkins to work with Artifactory/Nexus

INFOR™

How ION use Virgo