



Graham Charters | IBM

SOA Flexibility with OSGi Remote Services and the Service Component Architecture

SOA **Flexibility** with **OSGi Remote Services** and the **Service Component Architecture**

...including a demo...or two

SOA Design Principles

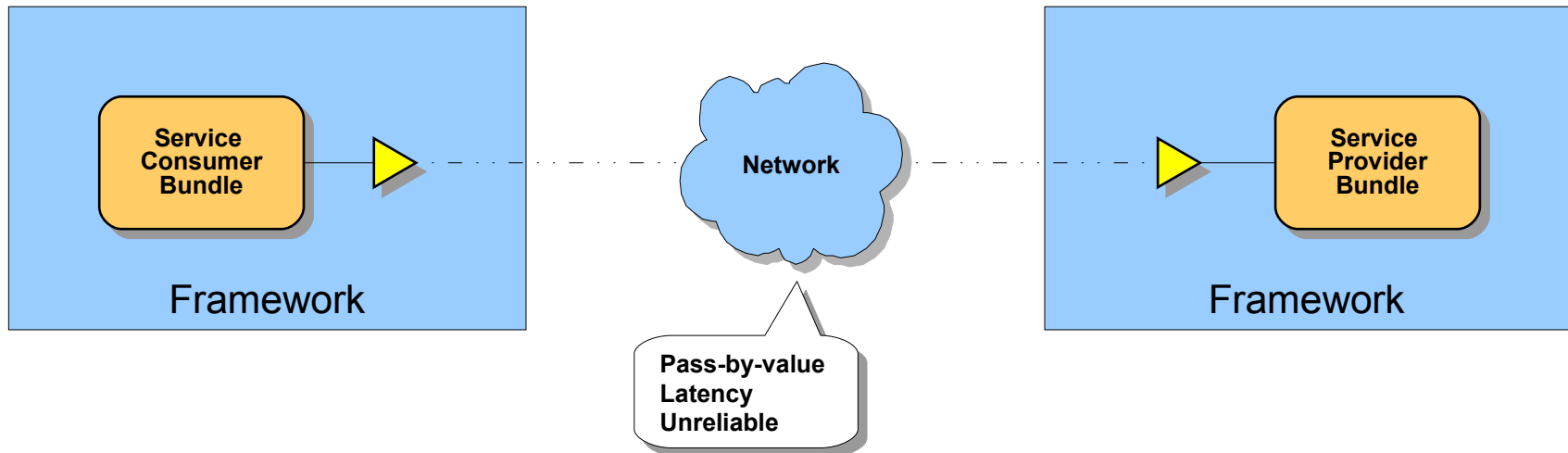
- **Service-based** for simple interaction patterns
- **Loose Coupling** so components need not know implementation details and can be bound late
- **Separation of Concerns** so developers can focus on immediate task
- **Composition** and re-composition of assets
- **Heterogeneity** to re-use what we already have

Flexibility

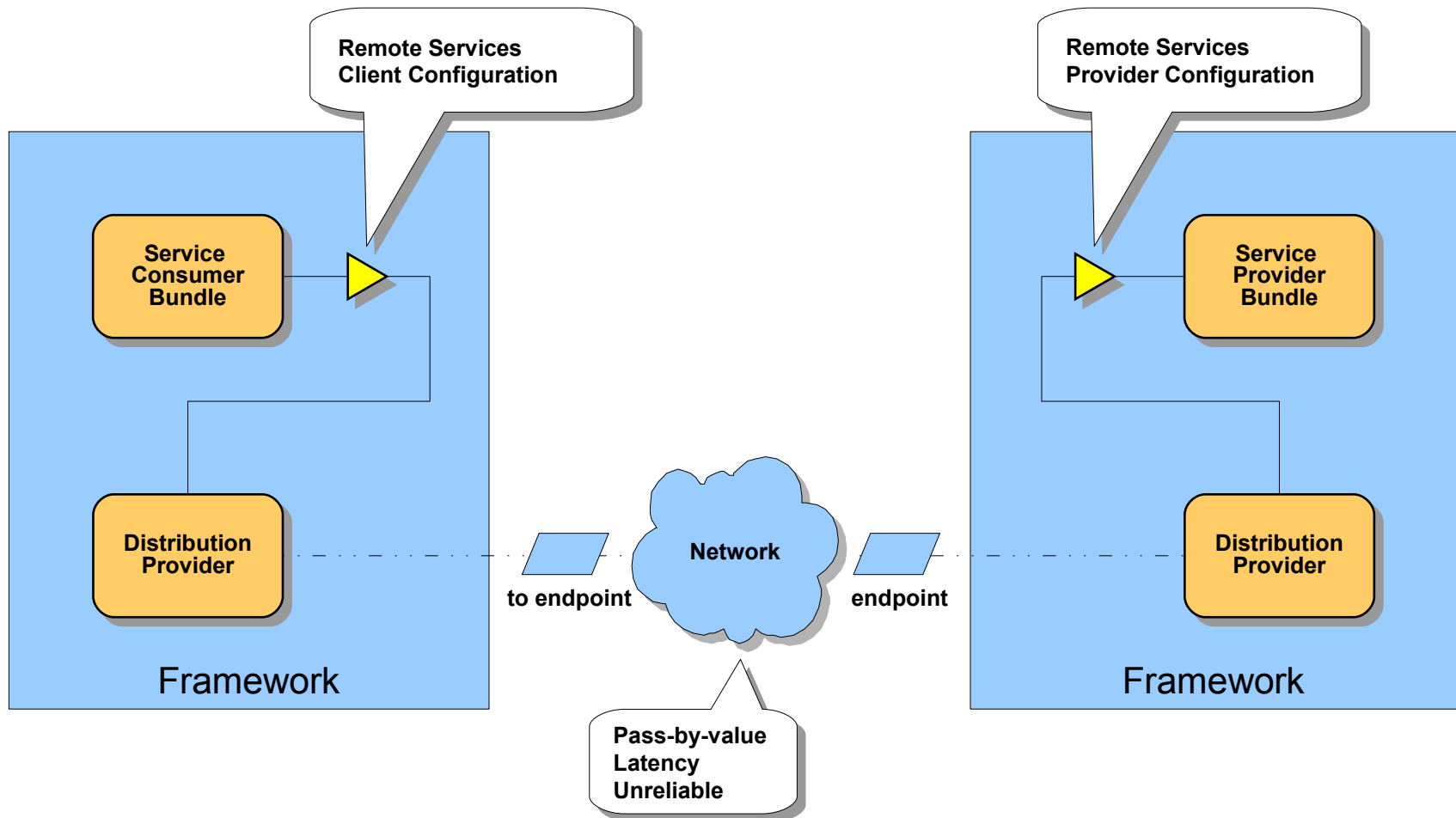
- What might we want to be flexible?
- Choice of Service Provider?
- Choice of Binding?
- Qualities of Service?
- All of the above



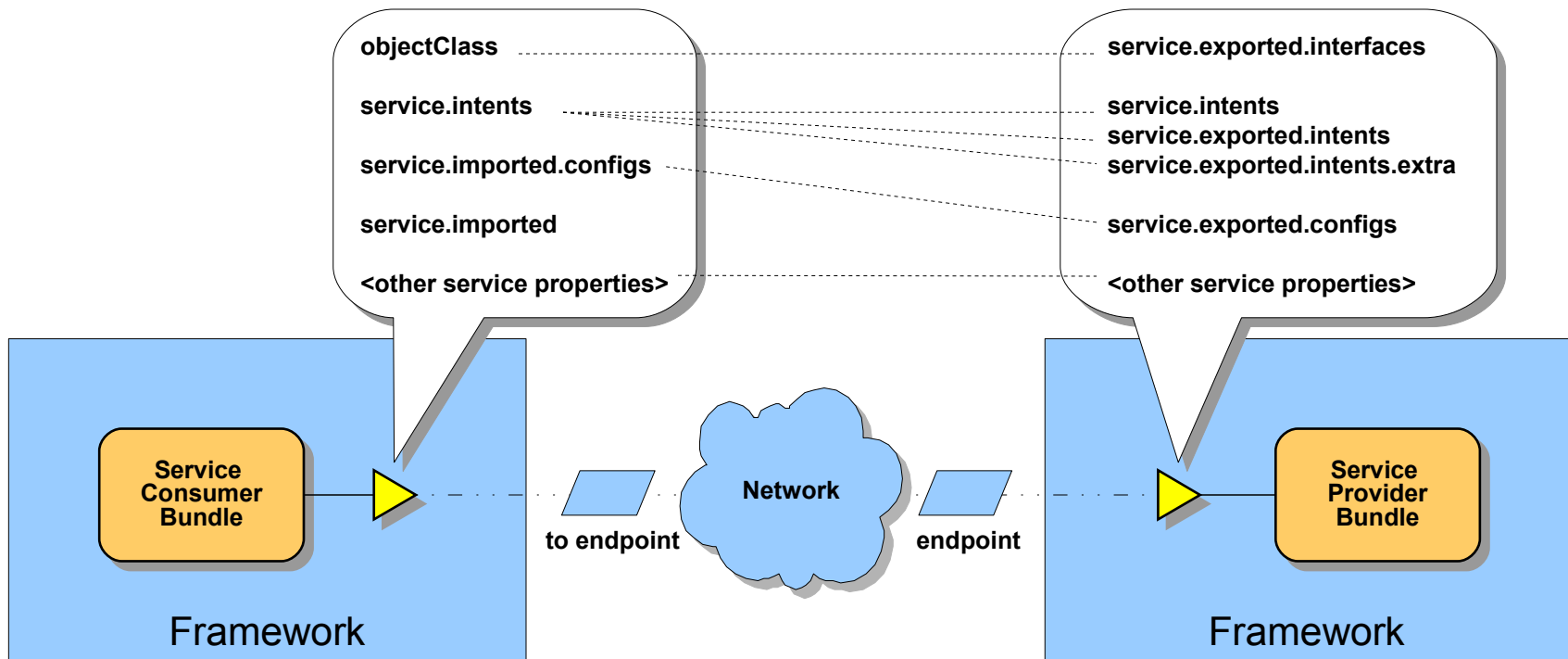
Remote Services



Distribution Provider



Services Properties



Intents

- Describe an abstract requirement or capability
 - Separation of concerns – requirement/capability independent of implementation
 - Implementation is a deployment choice
- Three properties
 - service.intents – provided by the service (local and remote)
 - service.exported.intents – statically configured for remote
 - service.exported.intents.extra – admin configured for remote
- Inspired by SCA intents, examples include
 - confidentiality, integrity, atLeastOnce, atMostOnce, SOAP.v1_1, etc.

Configuration Types

- Identifies a type of Distribution Provider specific configuration
- Use reverse domain name scheme to avoid conflict
- Property naming convention used for extra config

service.exported.configs = **com.acme.config**

com.acme.config.host = ...

com.acme.config.port = ...

OSGi Remote Service Summary

- Standardizes basic metadata for distribution
 - Remote interfaces
 - Default invocation semantics
 - Place to hang QoS requirements (intents)
 - Place to hang distribution provider configuration
- However...
 - Distribution configuration details left to distribution provider
 - Encryption, signing, protocols, ...
 - Full configuration not portable
 - No standard way to configure for interoperability

...enter the SCA Configuration Type

- Standardizes detailed distribution provider configuration
 - Leveraging SCA Bindings, Policy specifications
- Fully Portable
- Interoperable through interoperable bindings and policy (e.g. WS-*)

Service Configuration

- Remote Service properties unchanged

service.exported.interfaces = ...

service.exported.intents = ...

etc...

- Defines a configuration type following Remote Services spec

service.exported.configs = [org.osgi.sca](#)

[org.osgi.sca.bindings](#) = <bindings to use>

SCA Bindings

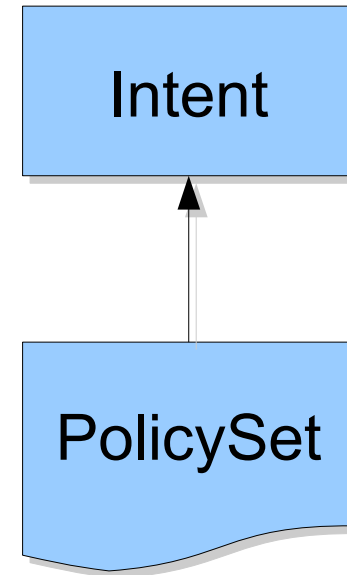
- Configuration for protocols and technologies
- Standard bindings (OASIS)
 - Web services, JMS, JCA

```
<binding.ws name="OrderServiceBinding"  
uri="http://localhost:8086/OrderService" />
```

- Extensible (e.g. Apache Tuscan)
 - JSON, JSON-RPC, Atom, RMI, EJB, ...

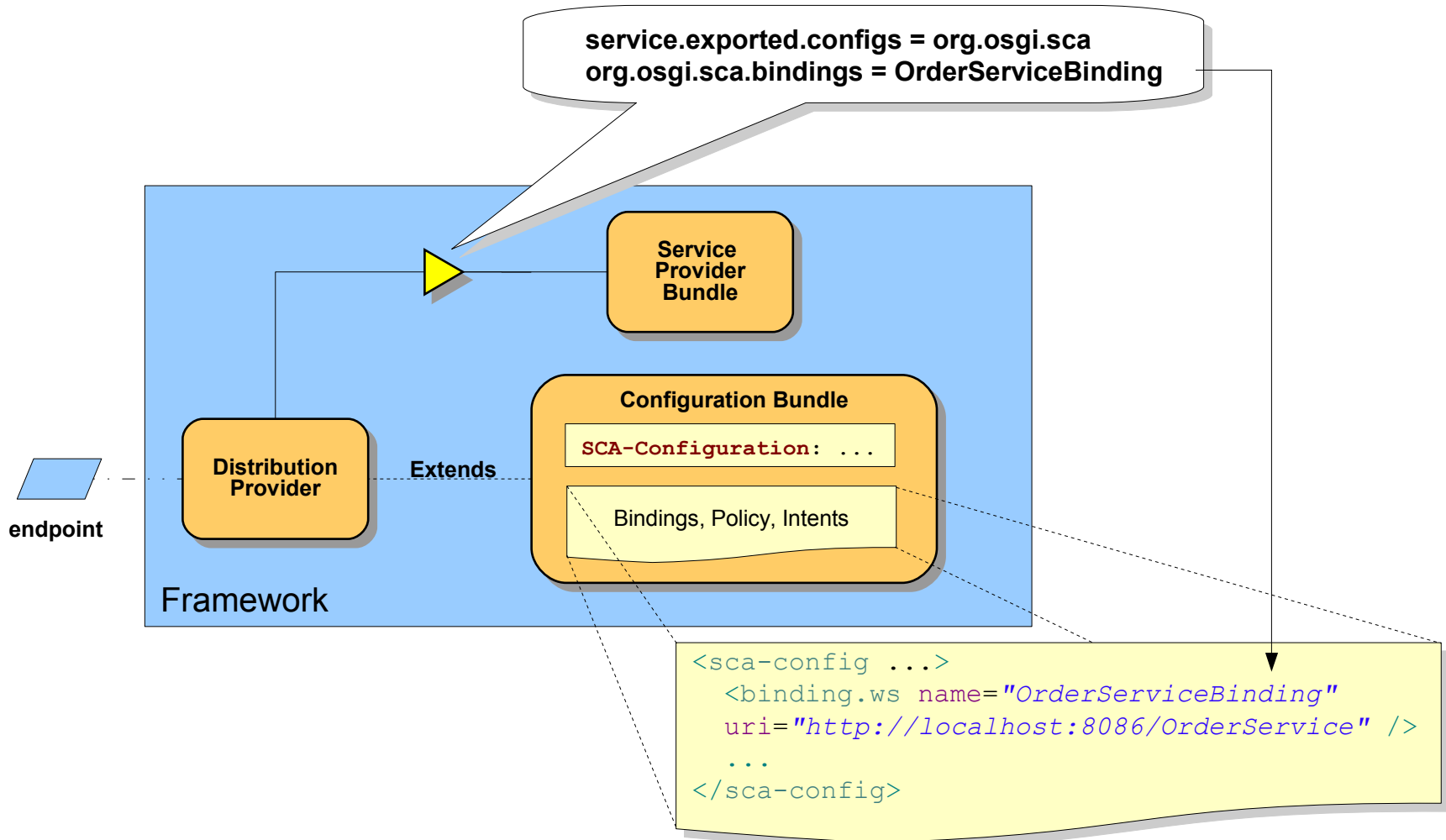
SCA Policy

- **Intents:** an abstract requirement or capability (sound familiar?)
- **PolicySet:** an implementation of an intent (using a policy language e.g. WS-Policy)
- Ideally the distribution provider is pre-configured so you never need to specify the details

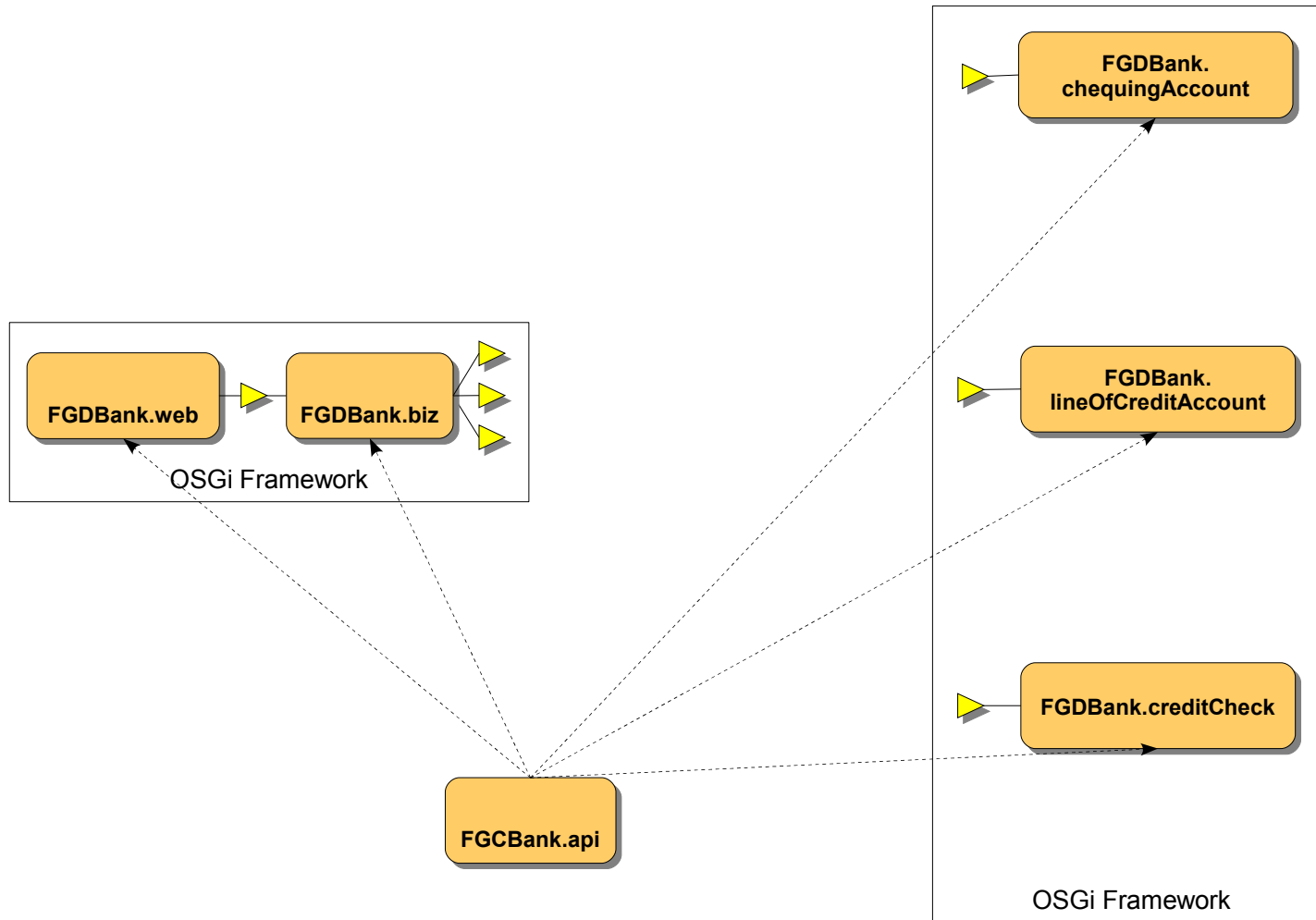


```
<sca:policySet name="Encrypted"
  provides="acme:protection"
  appliesTo="sca:binding.ws"
  xmlns:wsp="..." xmlns:sp="...">
  <wsp:Policy>
    <wsp:ExactlyOne>
      <sp:Basic256Rsa15 />
      <sp:TripleDesRsa15 />
    </wsp:ExactlyOne>
  </wsp:Policy>
</sca:policySet>
```

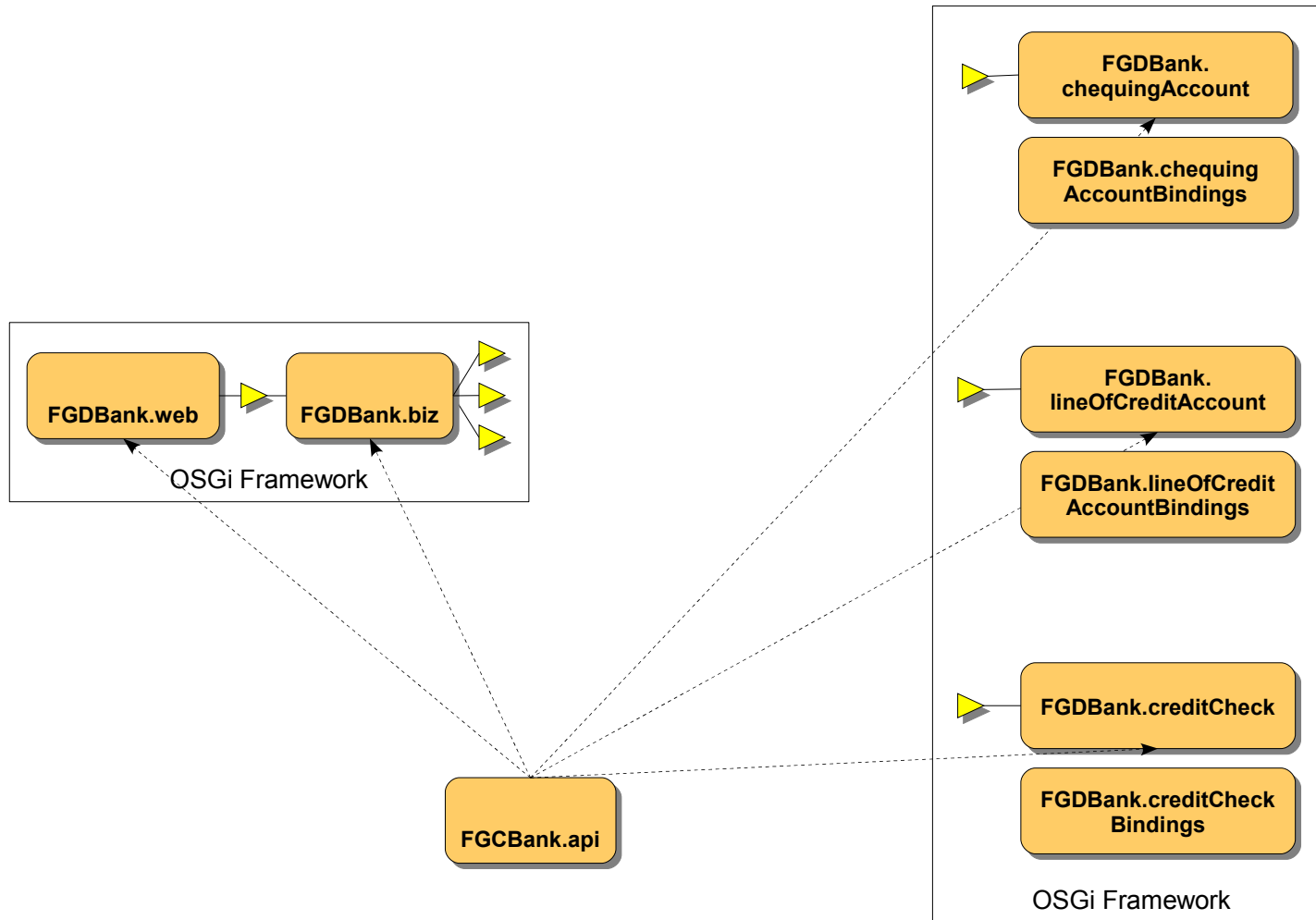
Configuration Bundle



Demo (Application Bundles)



Demo (adding Config Bundles)



Demo runtime – EquiTuscAries

- Head of Apache **Aries** + Head of Apache **Tuscany** on Eclipse **Equinox**
- Fingers cross this will work...

SCA Configuration Type Summary

- Standardizes fully-portable distribution provider configuration
- Based on SCA Bindings and SCA Policy
- Configuration for interoperability through interoperable binding

So where's the Flexibility?

- Configuration Bundle lifecycle determines
 - Endpoint availability
 - Protocol/technology choice
 - Quality of Service configuration
- Service properties determine
 - Quality of Service choice
 - Binding choice
- All configurable without changing the service client or provider code



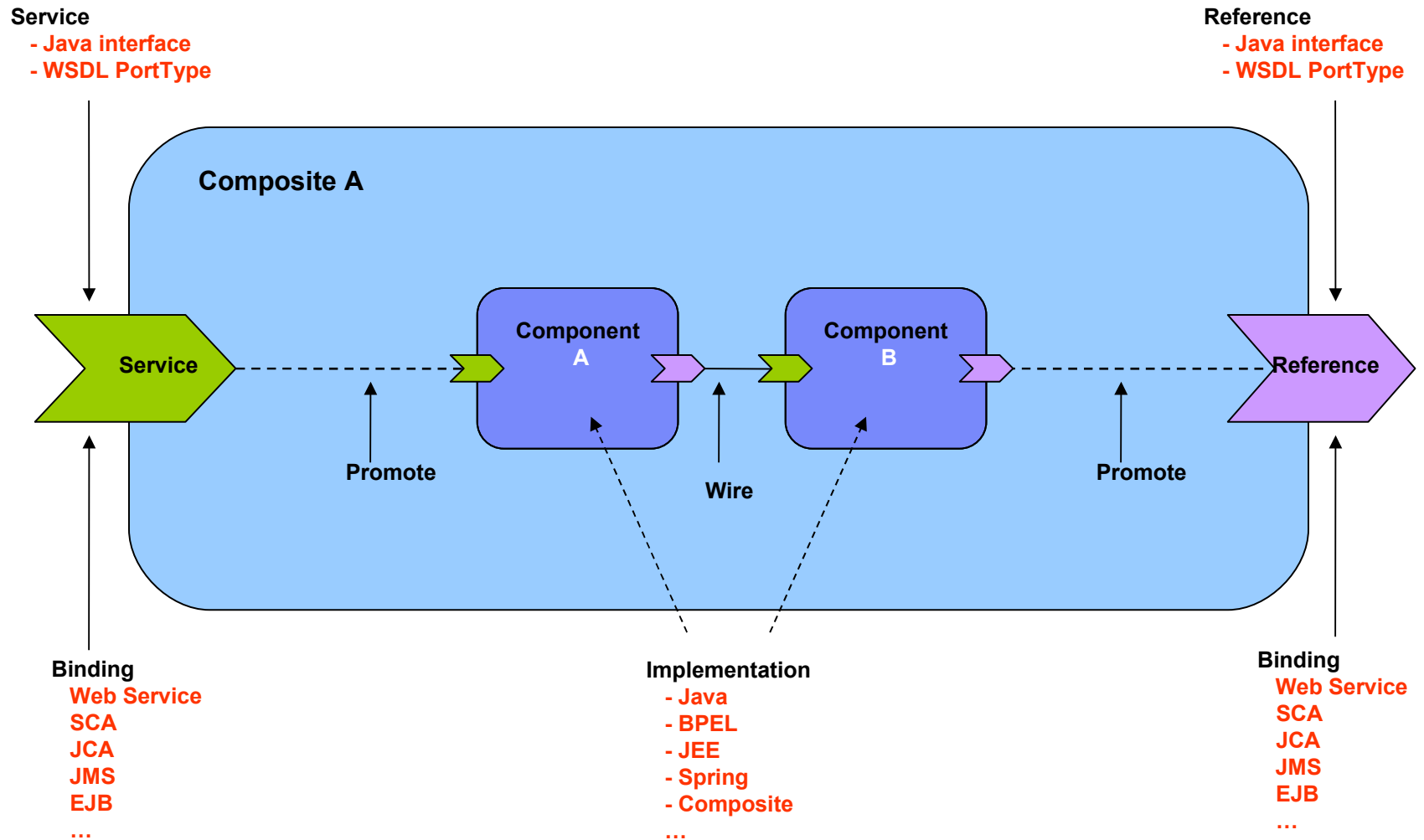
**But there's more to SCA than
bindings and policy...**

What is SCA?

- Heterogeneous component assembly
- Synchronous, asynchronous, event processing
- Pluggable communications protocols

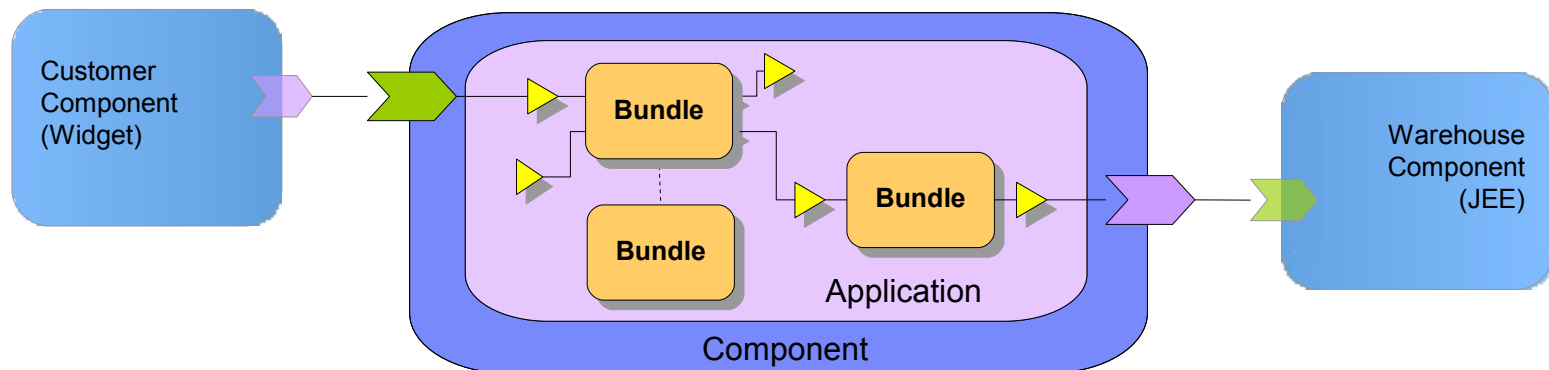
- Think Blueprint Service or Spring Framework only with bigger components implemented using different stuff and able to talk to other different stuff...

SCA Overview

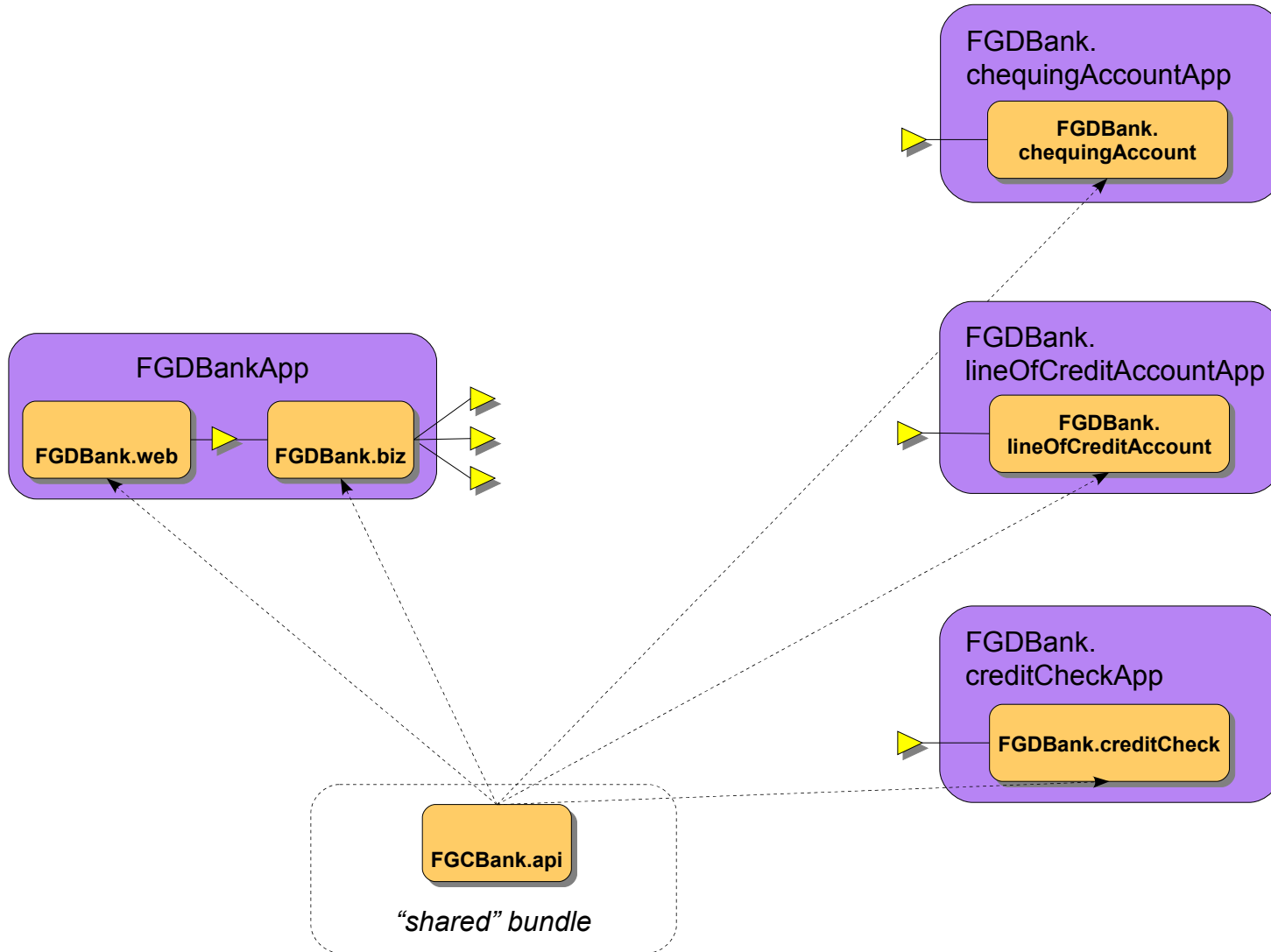


Where does OSGi fit?

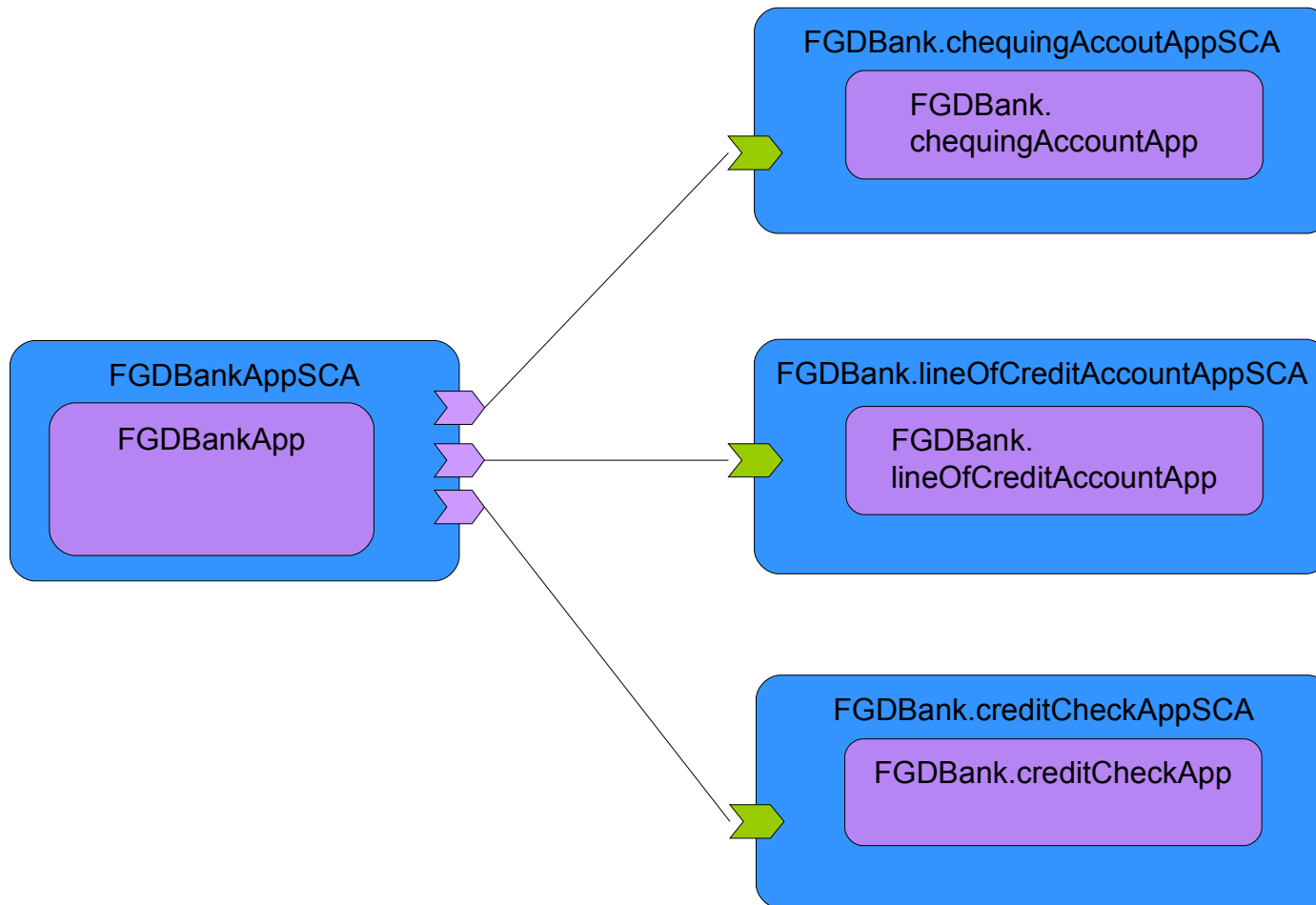
- Apache Aries and WebSphere Application Server define an OSGi Application Concept
- OSGi Applications fit well as an SCA Component implementation
- Re-use OSGi Remote Services metadata or have SCA provide configuration externally for even greater flexibility
 - SCA looks like a Distribution Provider to OSGi



Demo (OSGi Application View)



Demo (SCA Assembly View)



Summary

- OSGi Remote Services provides basic metadata for Distributing OSGi services
- SCA Configuration Type provides detailed portable distribution configuration
- SCA leverages OSGi Remote Service metadata to integrate OSGi applications
 - SCA acts as an OSGi Distribution Provider
- Distribution is non-invasive to client and service provider code and therefore enables flexible assembly and re-use

Useful Links

- Enterprise OSGi R4 V4.2 Specification
 - <http://www.osgi.org/Download/Release4V42>
- Apache Aries
 - <http://incubator.apache.org/aries/>
- Apache Tuscany
 - <http://tuscany.apache.org/>
- WebSphere Application Server OSGi Applications
 - <http://www-01.ibm.com/software/webservers/appserv/was/featurepacks/osgi/>