

eHF – Integration Platform

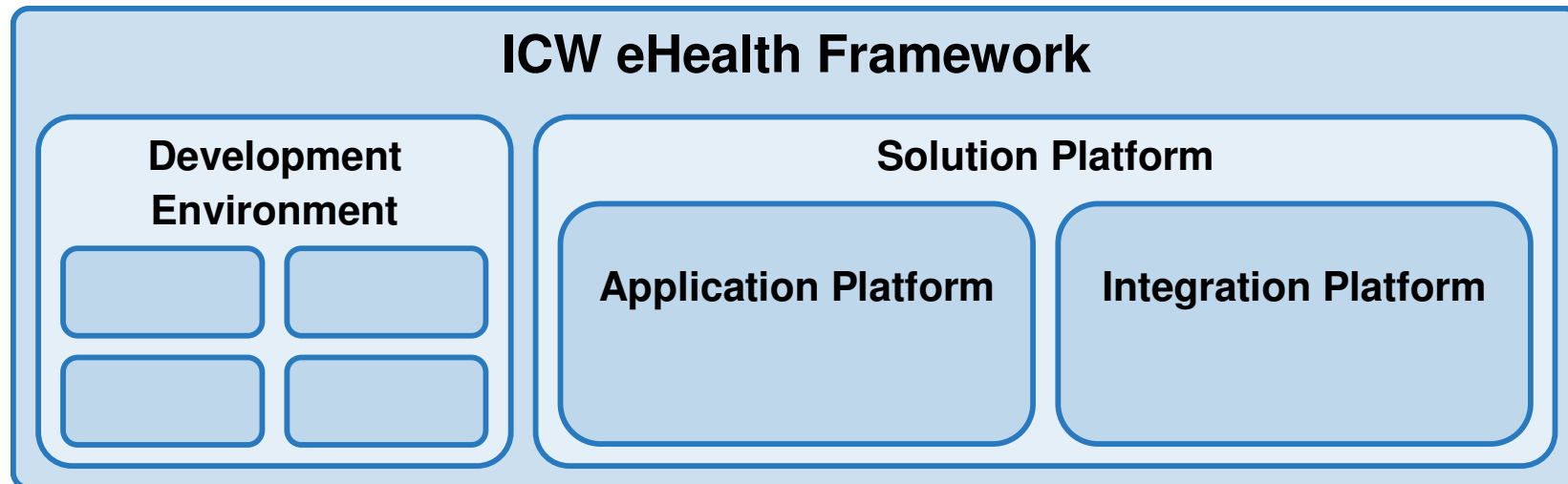
<Audience>

<Presenter> / <Date>

Agenda

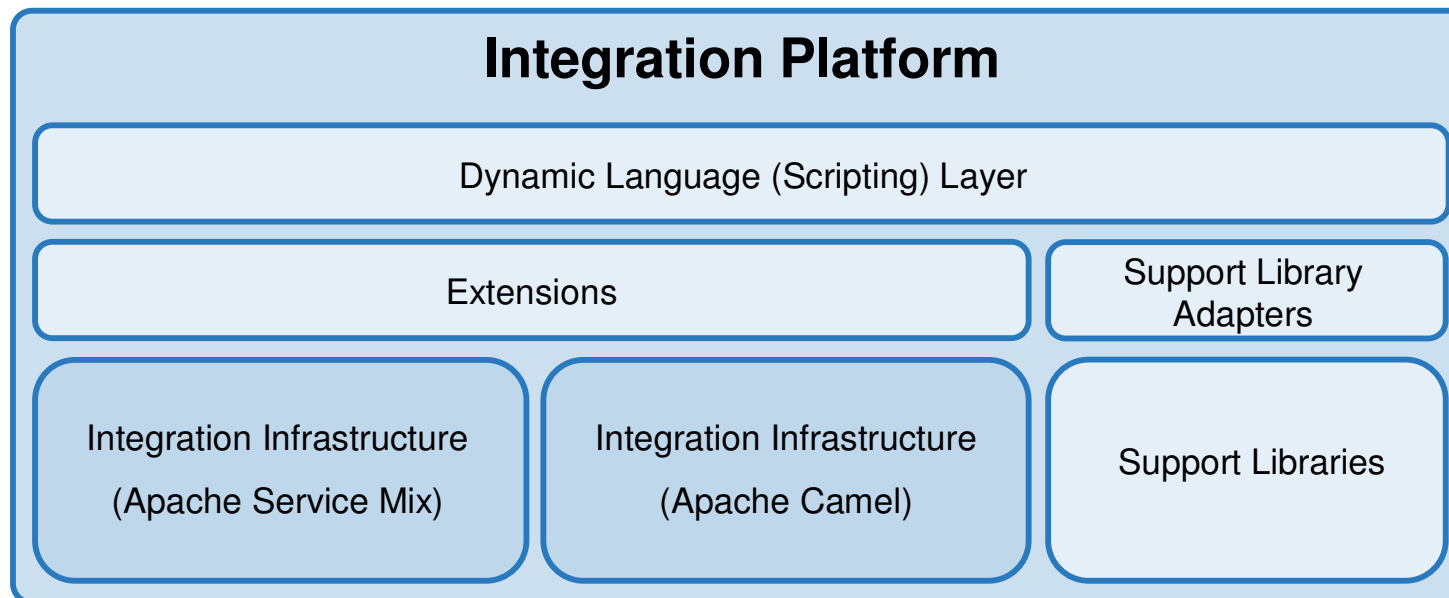
- Introduction
 - Apache Camel
 - Open eHealth Foundation – Integration Platform
- IPF inside eHF
- Route Design
- Modularization extended toward Integration
- Integration Opportunity
- Outlook
- Conceptual Conclusion

Introduction Overview



Introduction

Open eHealth Foundation – Integration Platform



Introduction

Need for Integration

- Business applications rarely live in isolation
- Expectation of instant access to all business functions in an enterprise, regardless of which system the functionality resides in
- Connecting disparate applications into an integrated solution
- Often happens using some form of „middleware“ or „integration platform“ or „enterprise service bus“ (ESB)
- Data integration → Information integration
- Merge information from disparate sources despite differing conceptual or contextual representations

Introduction

Types of Data/Information Integration

- **Technical Integration.** Provide connectivity to other systems over the network via transport protocol adapters.
 - Example: File → SOAP Web Services, MLLP (HL7) → HTTP
- **Semantic Integration.** Provide transformers to handle structural and semantic differences in messages that applications use to communicate with each other.
 - Example: CSV file dump → HL7 messages
- **Process coordination.** Provide routing and coordination logic to support complex message exchanges between services of a distributed system.
 - Example: ECG observation data analysis

Introduction

Apache Camel - Overview

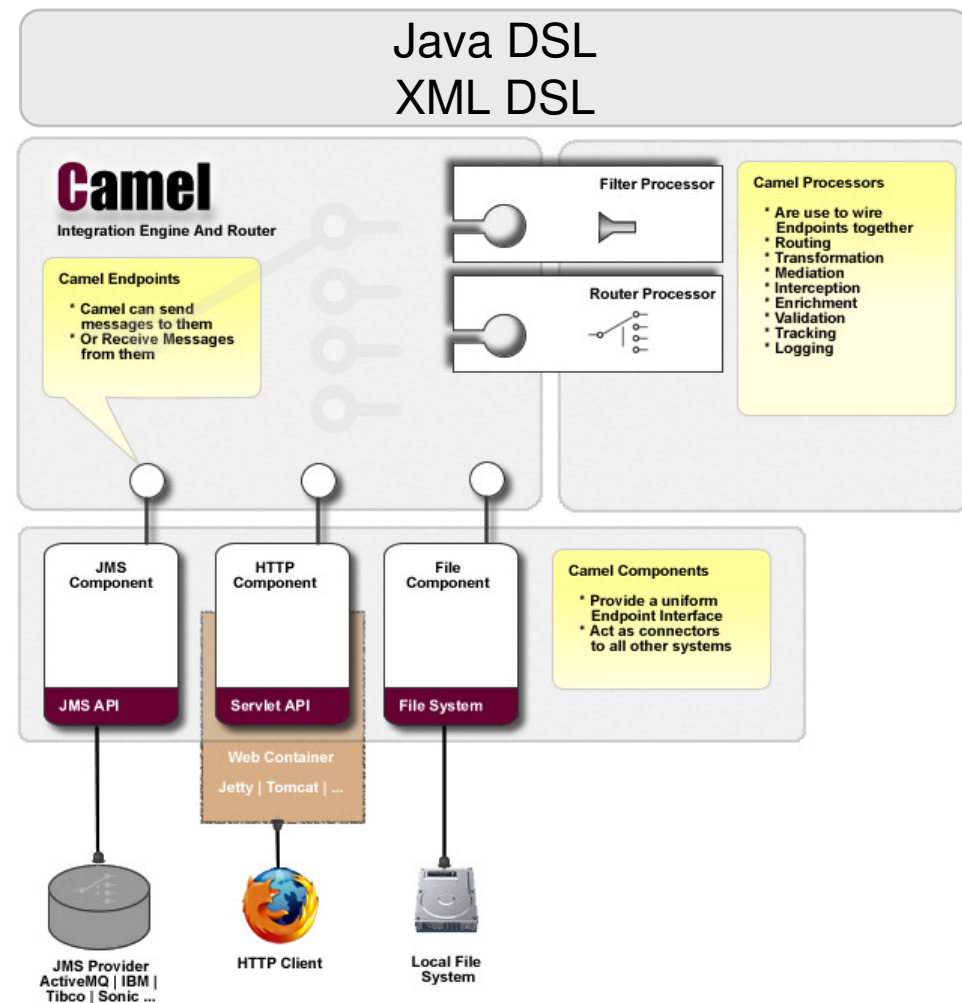


- Spring-based integration framework implementing Enterprise Integration Patterns, routing and mediation engine
 - CamelContext, Bean Integration (POJO approach), XML routes
- Allows you to configure integration logic in either a Java DSL (Fluent API) or via XML configuration files
- Uses URIs to work with any kind of transport or messaging model
 - Example URIs: *jms:topic:SomeTopic*, *bean:service?method=do*, *https://ehf/record/v2-5-0/service*
- Provides a lot of integration components out of the box
 - JMS, HTTP, Jetty, MINA, Spring Bean,
- Integrates with other open source projects
 - ActiveMQ (message broker), CXF (web service), ServiceMix (ESB)

Introduction

Apache Camel - Architecture

- Component: A Component acts as connector to other systems or middleware and is essentially a factory for Camel Endpoints
- Endpoint: An Endpoint represents a channel to which messages can be sent and from which messages can be received
- Processor: Processors are used to wire Endpoints together to Routes following the Pipes-and-Filters pattern. A Camel processor processes a message, e.g. transformation, validation, routing...



Example Camel route:

```
from("jetty:http://localhost:8080/myService")
    .process(new MyServiceProcessor())
    .to("bean:serviceBean?methodName=do")
```

Introduction

IPF - Overview

“The Open eHealth Integration platform (IPF) is an integration platform based on Apache Camel with special support for the eHealth domain”

- Integration modules
 - Adapters for platform-independent message processing libraries (validator, aggregator, transformer) into platform-specific message processing routes
 - Reusability of integration modules, platform dependent integration routes for wiring
- Groovy Scripting Layer
 - Extending Camel’s native DSL to define custom extension (custom language elements) in Groovy
 - HL7 message processing DSL
- Flow Management
 - Service to monitor, replay, query and audit message flows
- Quality of Service
 - Failure Recovery, Transactional messaging, Clustering
- Custom integration components
 - Message stream processing, rule and process engine support, large binary support

IPF inside eHF

eHF Servlet Component

- **Usage:** `from("service:<relative-path>") ...`
- Binds the endpoint to a generic servlet that dispatches request based on the relative path of the URL
 - <https://localhost/<product>/service/<relative-path>>
- Interpretation of the request is subject to the technical route
- Motivation: use to expose arbitrary procedural services from inside a servlet container.

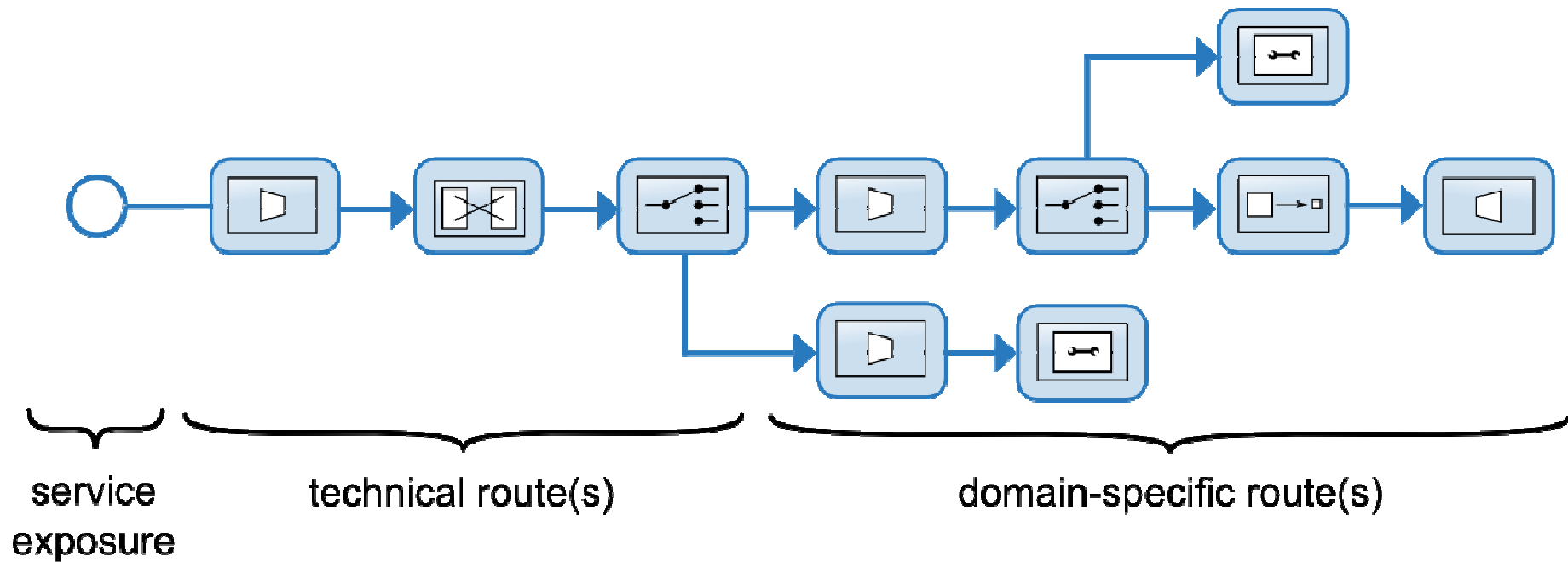
IPF inside eHF

eHF Resource Component

- **Usage:** `from("resource:<uri-template>") ...`
- Binds the endpoint to a restlet infrastructure based on the uri-template according to JAX-RS (JSR-311).
(The Noelios Restlet Engine provides a servlet connector which acts as an adapter between Servlet API and Restlet API)
 - <https://localhost/<product>/resource/<uri-template-match>>
- Supports the standard HTTP methods GET, POST, PUT, DELETE, HEAD, ...
- Motivation: use to expose ReST-style data-centric services from inside a servlet container.

Route Design

eHF Best Practices

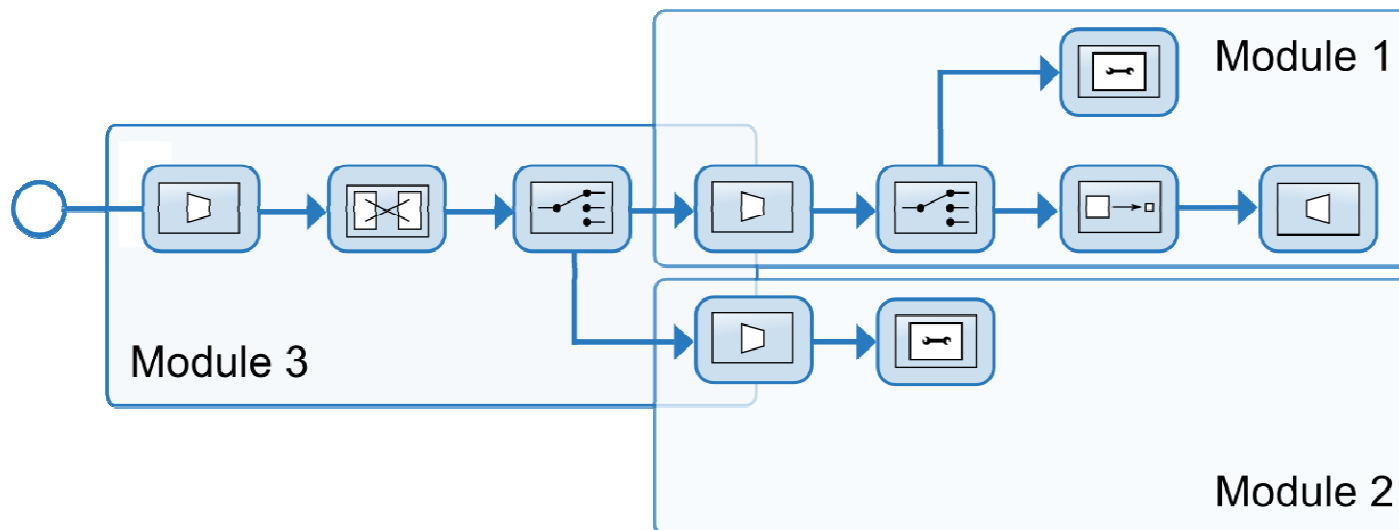


What is a Service now? Endpoint / Service Terminology

- A service is the exposure of one or many endpoints.
- The term **Endpoint** is used in the area of routes and route definitions. A route starts with an incoming endpoint and may use several outward facing endpoints consuming either external functions or further internal routes.
- We speak of **Service** once one or more endpoints have a manifestation towards the outside of an application (the endpoints are exposed).

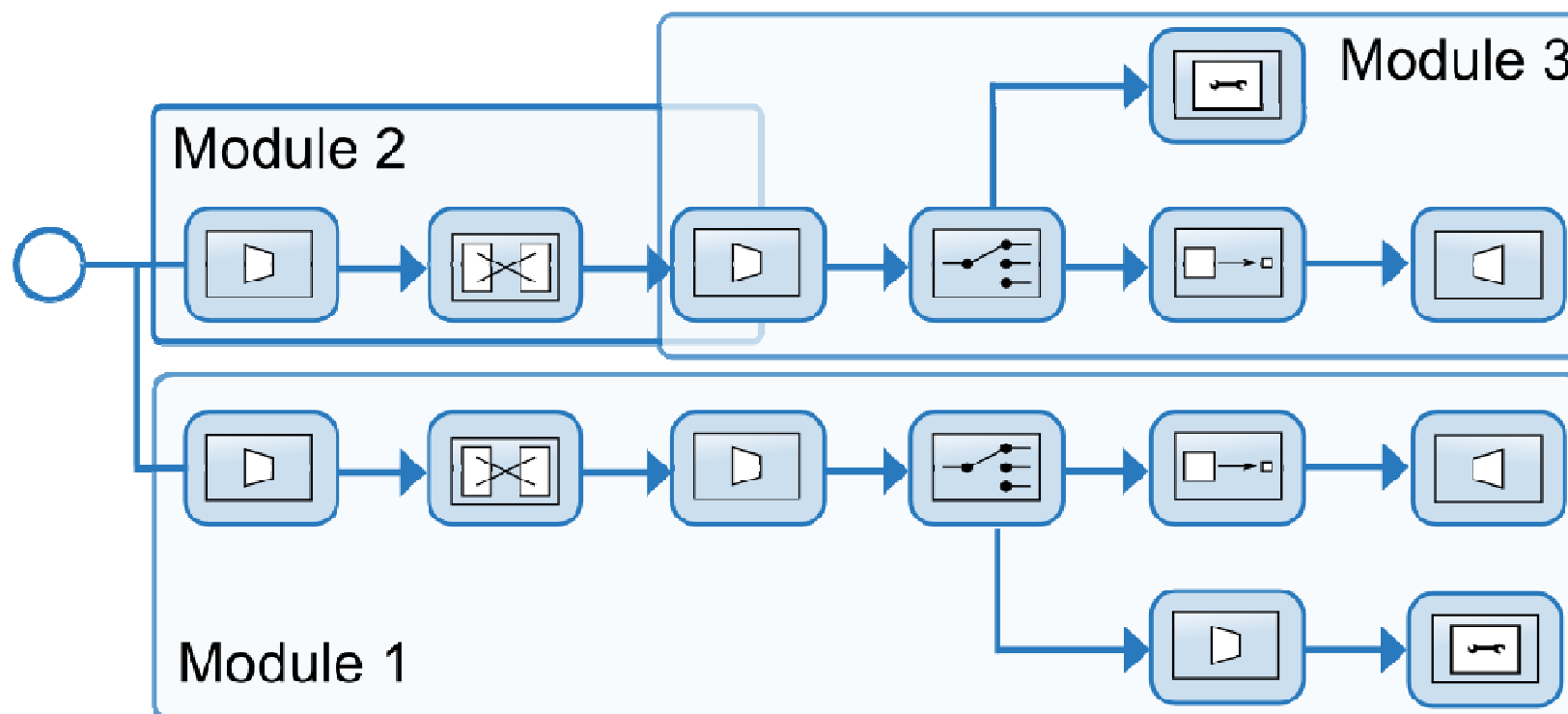
Modularization extended toward Integration

- Modules can contribute processors as well as routes
- They can contribute a technical route to a 'modular' service
- They can contribute domain-specific routes to be reused in the context of different types (protocols, styles) of services.



Modularization extended toward Integration

Contributions Options

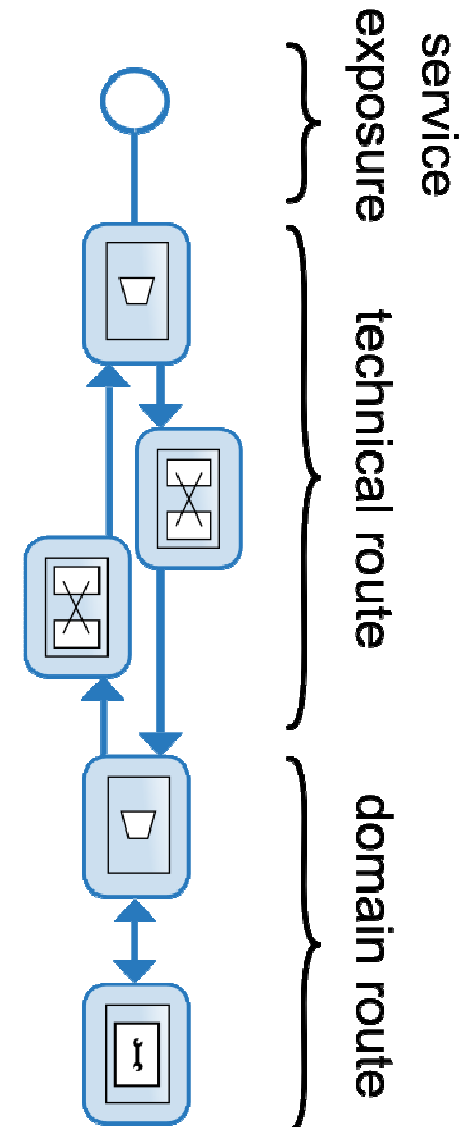


Example ReST Route In Pictos and Code

```

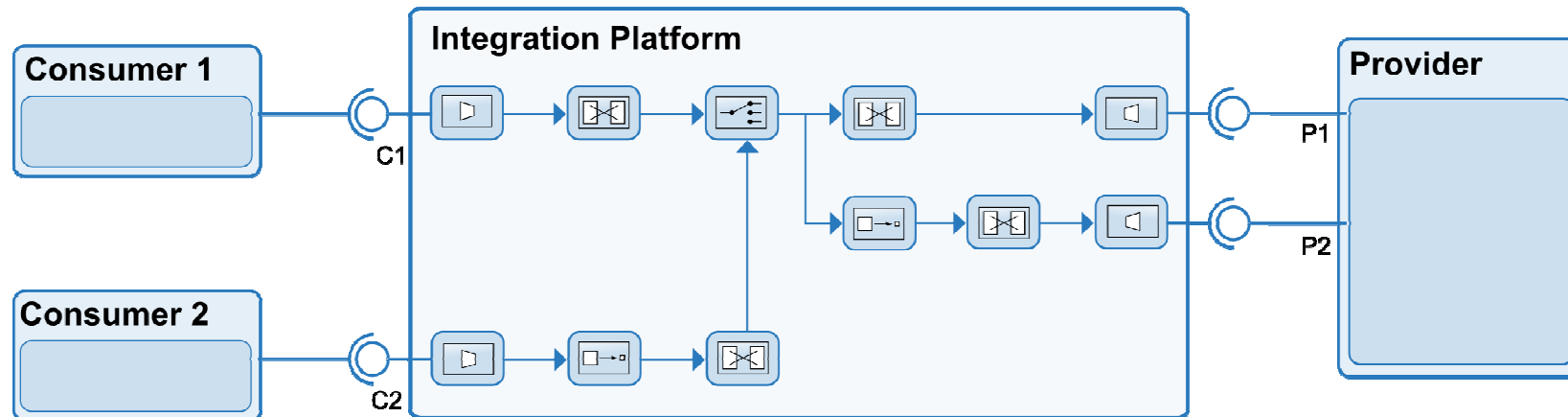
from("resource:cabinet?restletMethod=post")
    .process(cabinetUnmarshaller())
.to("direct:create-cabinet")
    .process(cabinetMarshaller());

from("direct:create-cabinet")
    .process(cabinetCrudProcessor.create());
    
```



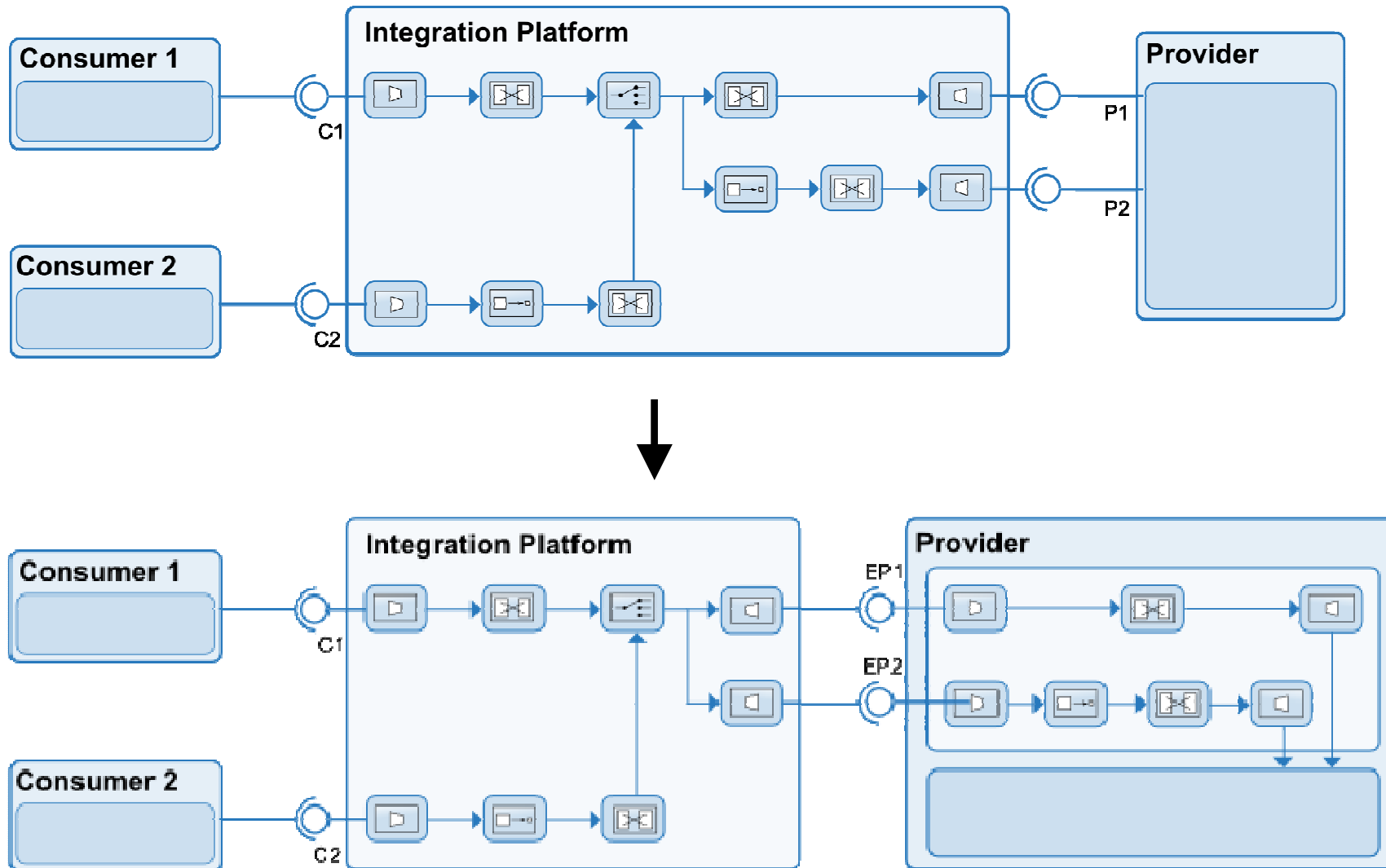
Integration Opportunity

Moving Integration closer to the Action



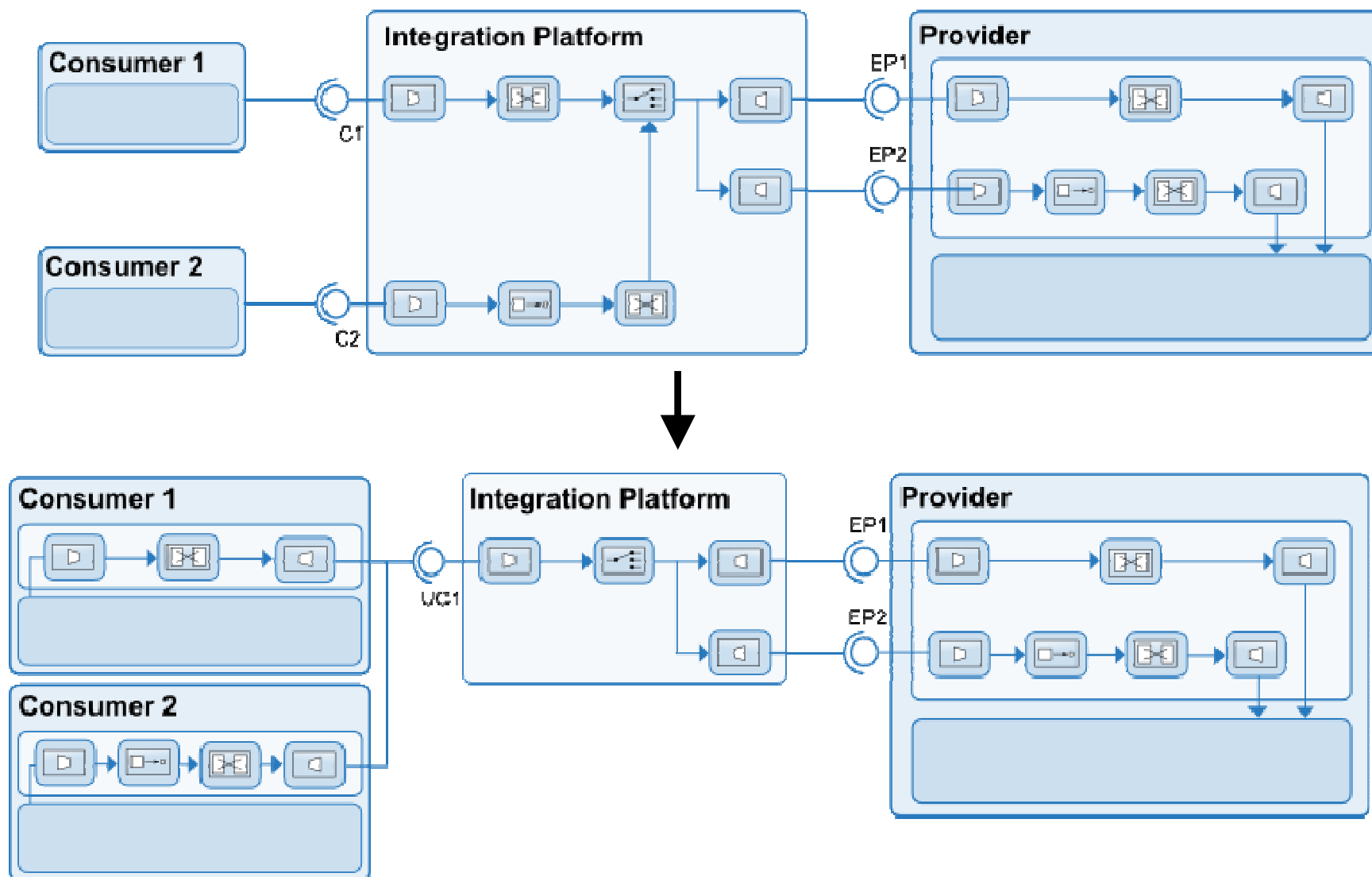
Integration Opportunity

Moving Integration closer to the Action



Integration Opportunity

Moving Integration closer to the Action



IPF DSL Extension

„Camel lets you create the Enterprise Integration Patterns to implement routing and mediation rules in a **Java based Domain Specific Language (Fluent API)**“

- **Java based DSL or Fluent API** is way of implementing an object oriented API in a way that provides more readable, fluent code
 - The term Fluent API was coined by Martin Fowler and Eric Evans
 - Fluent API is normally implemented by using method chaining to relay the instruction context of a subsequent call
 - Also known from Hibernate when building dynamic queries with Criteria API

Exampel 1: Camel Fluent API

```
from("resource:cabinet?restletMethod=post")
    .process(cabinetUnmarshaller())
    .to("direct:create-cabinet")
    .process(cabinetMarshaller());
```

Example 2:LiquidForm - Language Integrated Queries for ORM

```
select(p.getName())
    .from(Person.class)
    .as(p)
    .orderBy(p.getAge())
    .andThenBy(p.getSurname(), DESC);
```

- Limitation: Camel itself doesn't support DSL extensions on Java-level

IPF DSL Extension

- IPF scripting layer is an extension to the Camel DSL to provide more fluent code
 - Based on the meta-programming mechanism of Groovy
 - DSL extension mechanism and predefined extensions

Camel Route

```
public class CabinetResourceRouteBuilder extends RouteBuilder{  
  
    public void configure(){  
        from("resource:cabinet?restletMethod=post")  
            .process(cabinetUnmarshaller())  
            .to("direct:create-cabinet")  
            .process(cabinetMarshaller());  
    }  
}
```

Camel Route with IPF

```
public class CabinetResourceRouteBuilder implements RouteBuilderConfig {  
  
    void apply(RouteBuilder builder){  
        builder  
            .from("resource:cabinet?restletMethod=post")  
            .unmarshalCabinet()  
            .to("direct:create-cabinet")  
            .marshalCabinet();  
    }  
}
```

IPF's DSL Extension (cntd.)

Camel Route with IPF

```
public class CabinetResourceRouteBuilder implements RouteBuilderConfig {  
  
void apply(RouteBuilder builder){  
builder  
    .from("resource:cabinet?restletMethod=post")  
    .unmarshalCabinet()  
    .to("direct:create-cabinet")  
    .marshalCabinet();  
}
```

Groovy extension – based on Groovy's ExpandoMetaClass

```
public class CabinetResourceRouteBuilderExtension {  
  
def routeBuilder  
  
def extensions = {  
    ProcessorType.metaClass.marshalCabinet = {  
        delegate.process(routeBuilder.transmogriifier(cabinetMarshaller))  
    }  
  
    ProcessorType.metaClass.unmarshalCabinet = {  
        delegate.process(routeBuilder.transmogriifier(cabinetUnmarshaller))  
    }  
}  
}
```

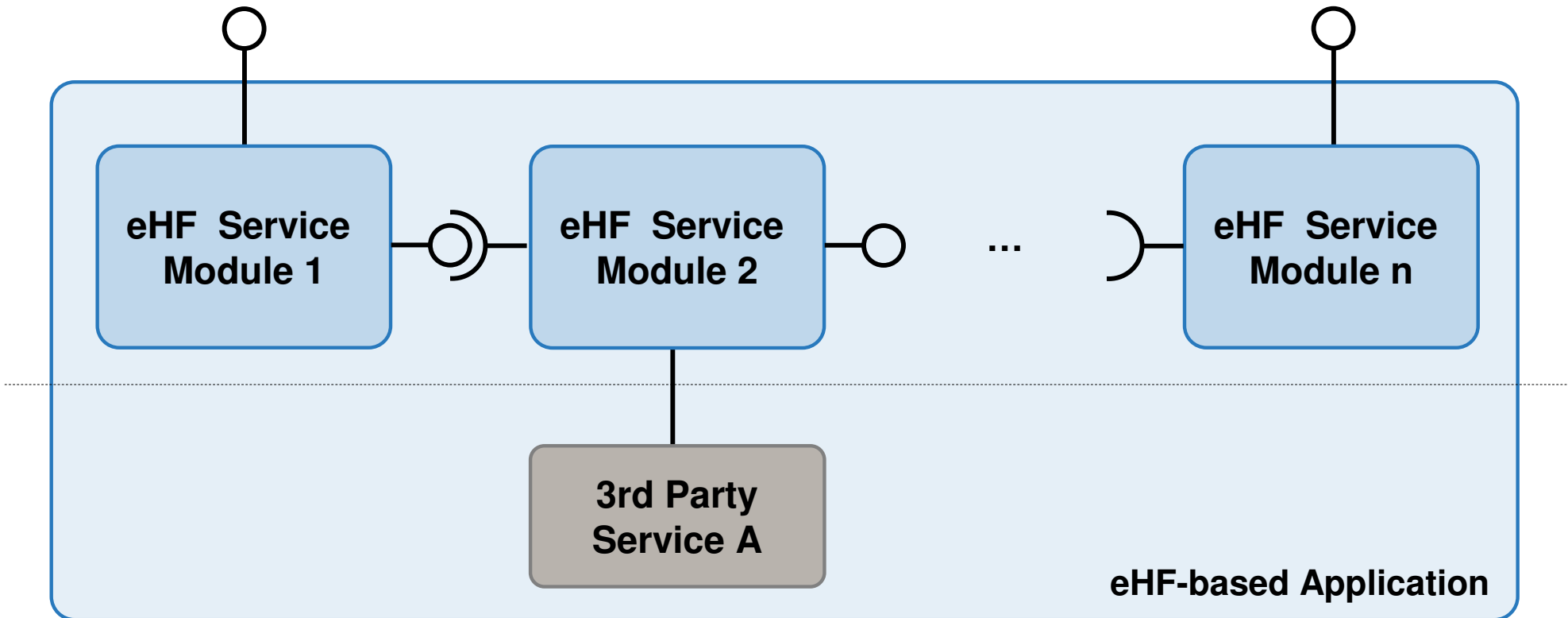
Outlook

- ReSTful Record Service
- IHE Actor Implementation based on IPF and eHF
 - DSL Extensions supporting IHE
 - IPF inside for bridging IHE specified interface and modules
- Evaluate opportunities towards Portals

Conceptual Conclusion eHF-Based Application

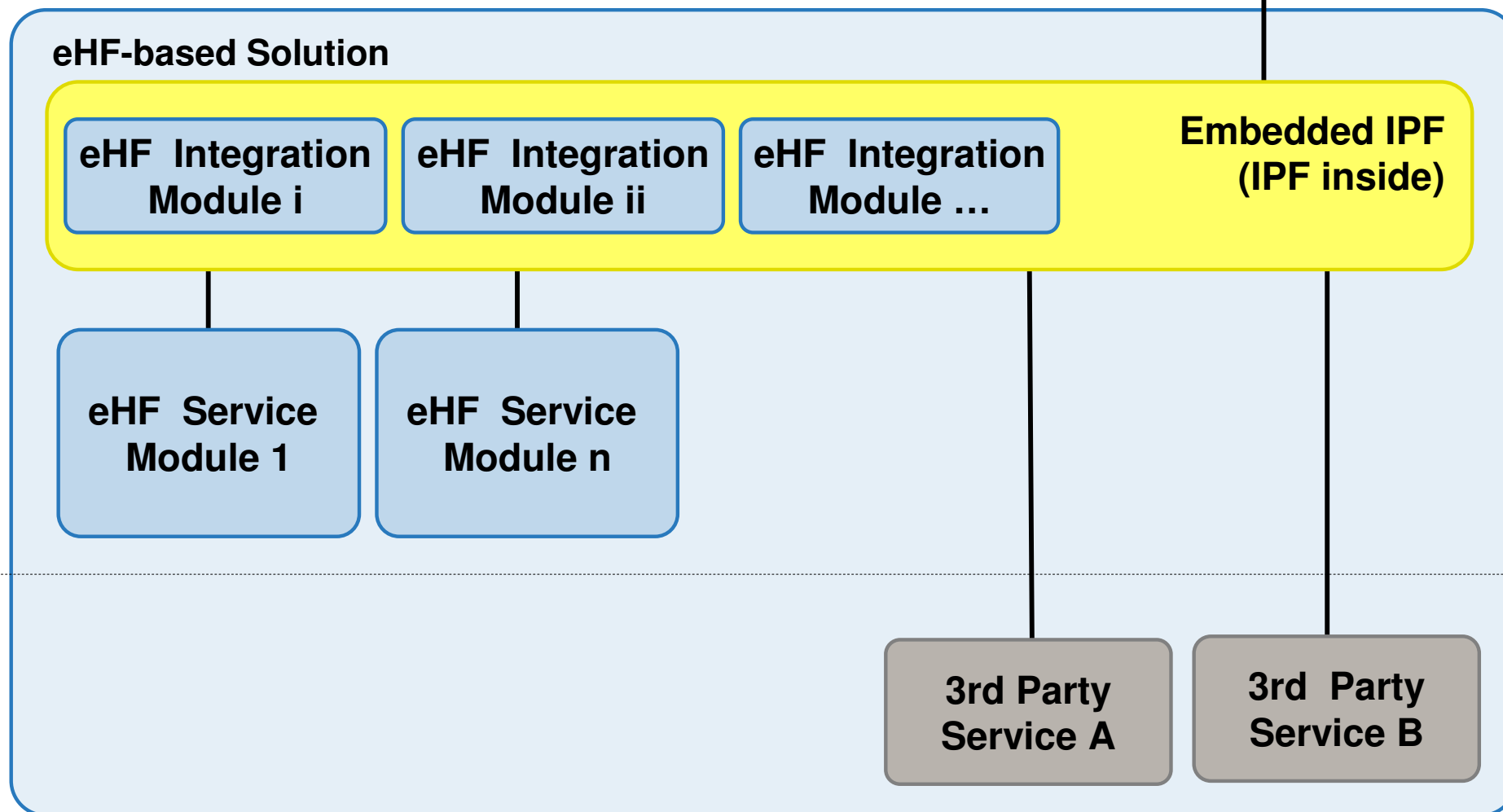
Module-specific Service

Application-specific Service

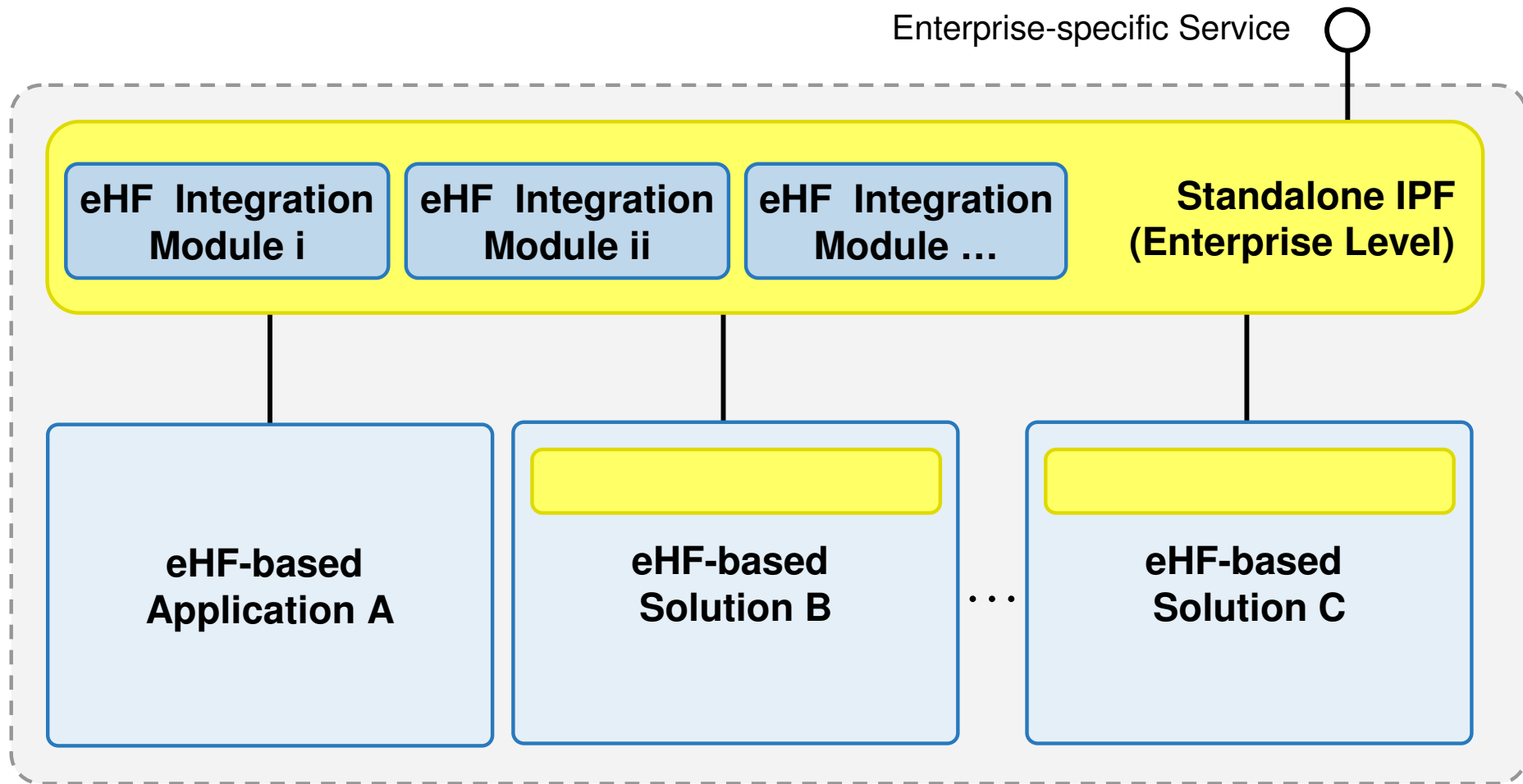


Conceptual Conclusion eHF-Based Solution

Solution-specific Service



Conceptual Conclusion eHF in the Enterprise

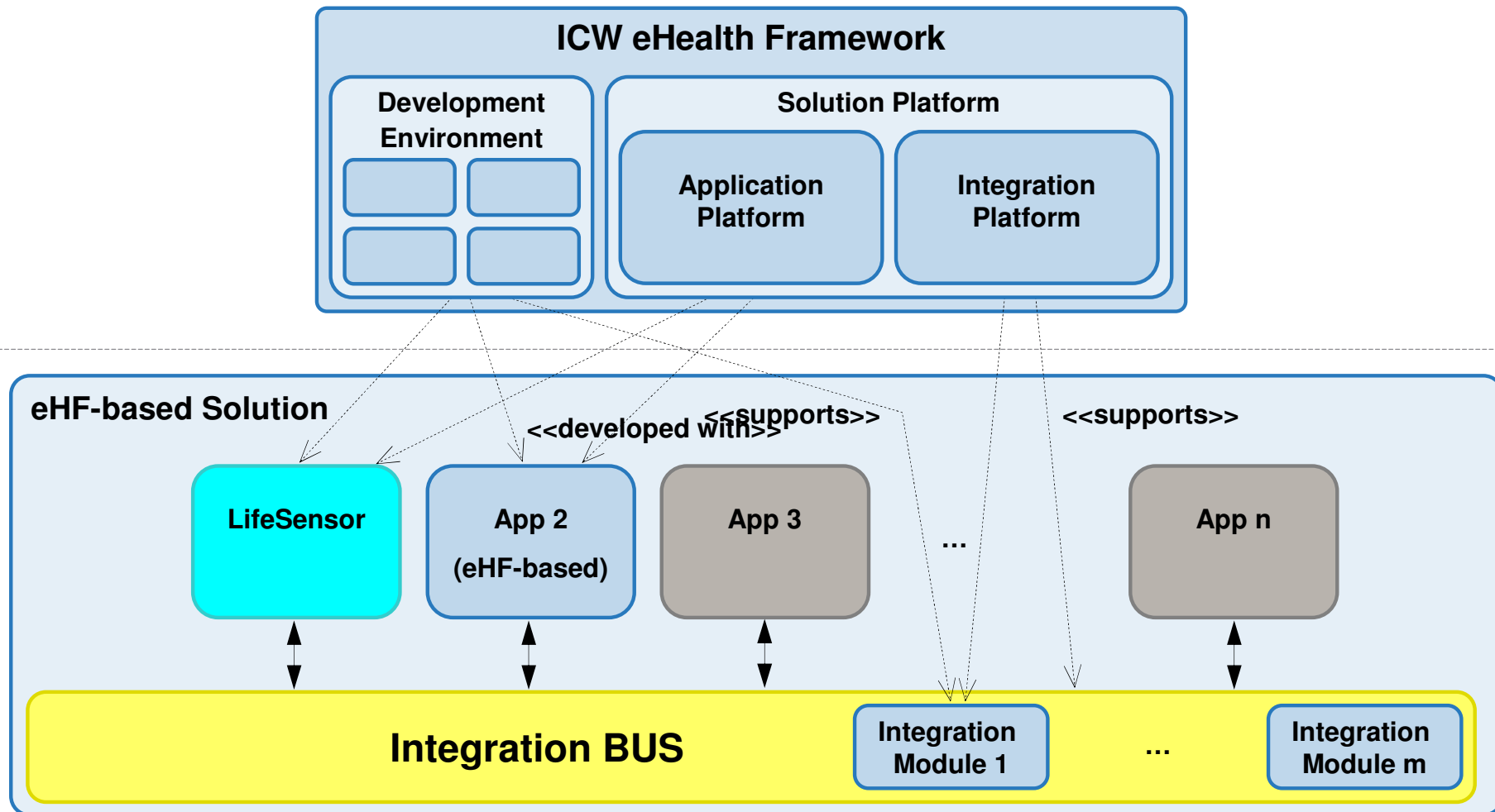


Thank you for your attention!

<presenter>@icw.de

Introduction

Developing Solutions with eHF



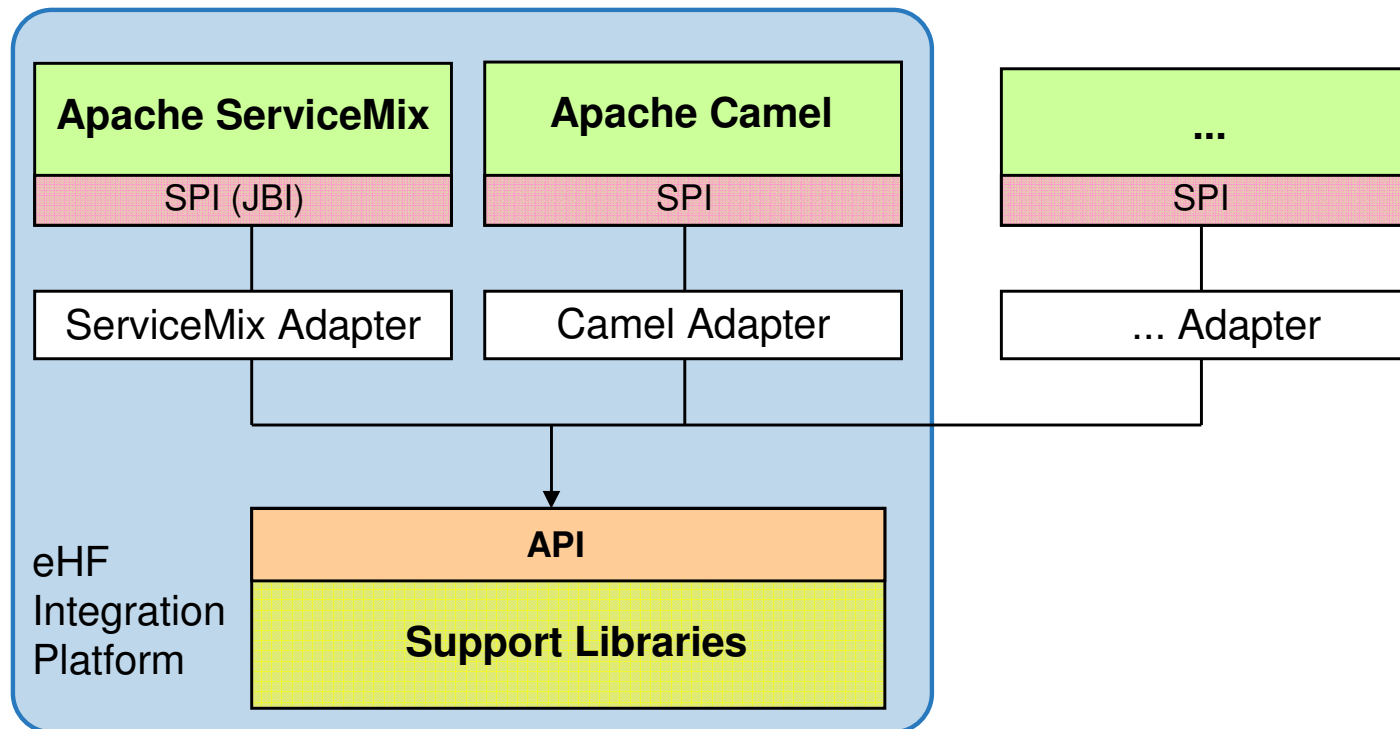
Structuring Integration Logic

- Goals
 - re-use custom integration logic across different integration projects and platforms
 - avoid lock-in to a certain ESB product technology
- Approach
 - implement integration logic independent from the interfaces and the infrastructure of a certain ESB product
 - provide a thin adapter layer between integration logic and the ESB product's Service Provider Interface (SPI)
- Implementation
 - Support libraries implementing re-usable and custom integration logic
 - Apache ServiceMix (incl. support library adapters)
 - Apache Camel (incl. support library adapters)
 - ESB-specific extensions

eHF Integration Support Libraries

- Independent of Apache Camel and ServiceMix
- Message processing POJOs (70% of Integration Platform code)
- Domain-neutral components (jars)
 - `transform, csv, pdf ...`
- Infrastructure-specific components (jars)
 - `flow, ticket ...`
- Domain-specific components (jars)
 - `hl7v2, ghapi, observation, rtx, ls ...`
- Project-specific components (jars)
 - `orbis, thipa, mhri ...`

Structuring Integration Logic



eHF Integration Support Library Adapters

- Most support libraries implement a common API
 - Defined by `transform-api` component
 - **Interfaces:** `Validator`, `Transmogriifier`, `Parser` ...
 - Common `transform-api` is **Camel-independent**
- Adapters are Camel-dependent
 - Enable use of `transform-api` implementors in Camel routes
 - Adapters are generic and re-usable across support libraries
 - New support libraries can be integrated with existing adapters

eHF Integration Extensions

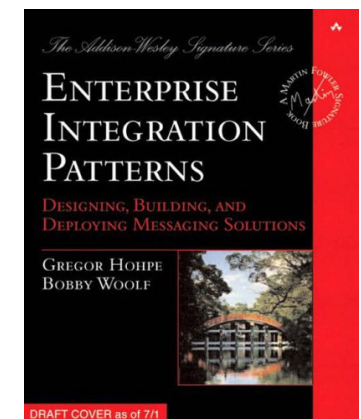
- Custom Camel Components
 - RTX device connector
 - HTTP connector (ext.)
 - MLLP connector
 - ...
- Custom Camel Processors
 - Validation process
 - Content enricher
 - Exchange bridges
 - ...
- Infrastructure Services
 - Flow management
 - ...

eHF Integration Scripting Layer

- Based on dynamic programming language
 - Route definition and extensions are written in Groovy
 - Support for functional programming paradigm
- Customization of existing integration solutions without recompilation
 - Dynamic route changes
 - Dynamic extension of Camel DSL
 - Definition of processing patterns (coarse-grained route building blocks)
- Supports 100% reuse of any existing Java-based Integration Platform features
 - Processors (integration patterns, protocol adapters ...)
 - Services (flow manager ...)
 - Route definitions (integration solutions ...)

Integration Patterns

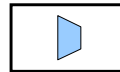
- Pattern: a theme of recurring events or objects, which repeat in a predictable manner.
- Integration logic implemented by ESB applications can intuitively be described in terms of [Enterprise Integration Patterns](#)
- Used for modeling and documentation of integration processes



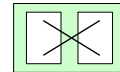
Integration Patterns (continued)

- Mediation Patterns

- Message Endpoint



- Message Transformer



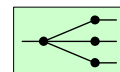
...

- Routing Patterns

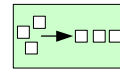
- Content based router



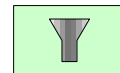
- Recipient List



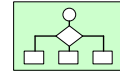
- Resequencer



- Filter



- Process Manager



...