

mule as an esb-style integration front-end to java ee applications

gerald loeffler, phd mba
senior architect/developer
objectlab financial ltd

ObjectLab Financial Ltd

ARTIUM[®]

references and resources

- [1] mule resources
 - web site/wiki/guides <http://www.mulesource.org>
 - mailing list
 - examples (!)
 - source code (!)
- [2] "enterprise integration patterns", gregor hohpe et al.
 - <http://www.eaipatterns.com/>
- [3] "Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus", jean-louis marechaux, ibm

esb - pattern or product?

- "An Enterprise Service Bus is an *architectural pattern* and can be implemented by many *different products* within the organization, and assembled together to act as a *federated bus*. More and more vendors are now offering a *complete product* to fulfill *enterprise integration* needs." [3]
- "combines *event-driven* and *service oriented* approaches to simplify integration of business units, bridging heterogeneous platforms and environments. The ESB acts as an intermediary layer to *enable communication* between different application processes." [3]

esb - pattern or product?

- "facilitates and simplifies business integration through *transport, event and mediation services*. It connects and mediates all communications and interactions between heterogeneous nodes, both in a Service-Oriented Architecture (synchronous one-to-one approach) and an Event-Driven Architecture (asynchronous many-to-many approach)." [3]
- esb characteristics/features:
 - supports synchronous and asynchronous interaction
 - message routing and transformation
 - transport mediation
 - distributed/federated?

mule 1/2

- event-based open-source esb
 - event \approx message
- mature, proprietary (neither jbi nor sca)
- active community
- integrates nicely with spring
- supports many transports/protocols
 - ejb invocation (jrmf, iiop), email (smtp, pop3, imap), file, ftp, http/https, jdbc, jms, ip multicast, tcp/ip and ssl, udp, soap/wsdl (via axis1, xfire, glue), xmpp, internal in-memory ("vm") [1]

mule 2/2

- deployment stand-alone or in app server (also as rar)
- framework for and implementations of
 - transformers
 - routers
 - splitters/aggregators
- transaction demarcation (local and xa, pluggable)
- thread, queue and pool management
- currently 1.4.1
 - 2.0 (july 2007) will be more tightly integrated with Spring 2.x

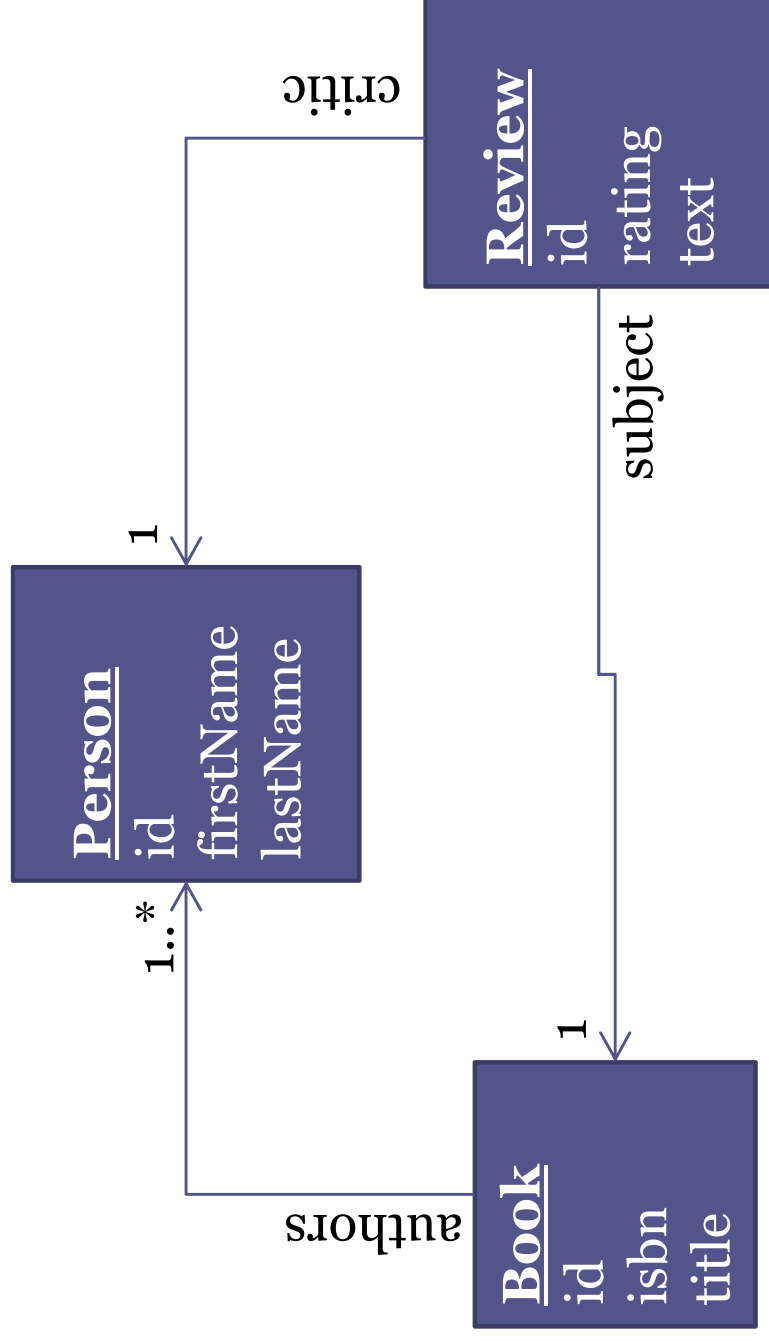
the integration scenario

book reviews, java ee with remote ejbs, file upload, webservice, feed

the integration scenario

- "our" application ("A") deals in book reviews
 - domain objects Book, Person, Review
- deployed in java ee application server, accessed by java clients through remote slsbs
- need to integrate with publishing house Zsolnay ("Z") to exchange reviews
 - step 1: they will initially send us reviews in flat files
 - step 2: they intend to move to web services real soon now
 - step 3: will ultimately also accept reviews from our system
- we expect many more integrations along these lines
- make our system/application integration ready/friendly

application domain model



application services

ReviewService

```
List<Review> getReviewsByBook(String isbn)  
void addReview(Review newReview)
```

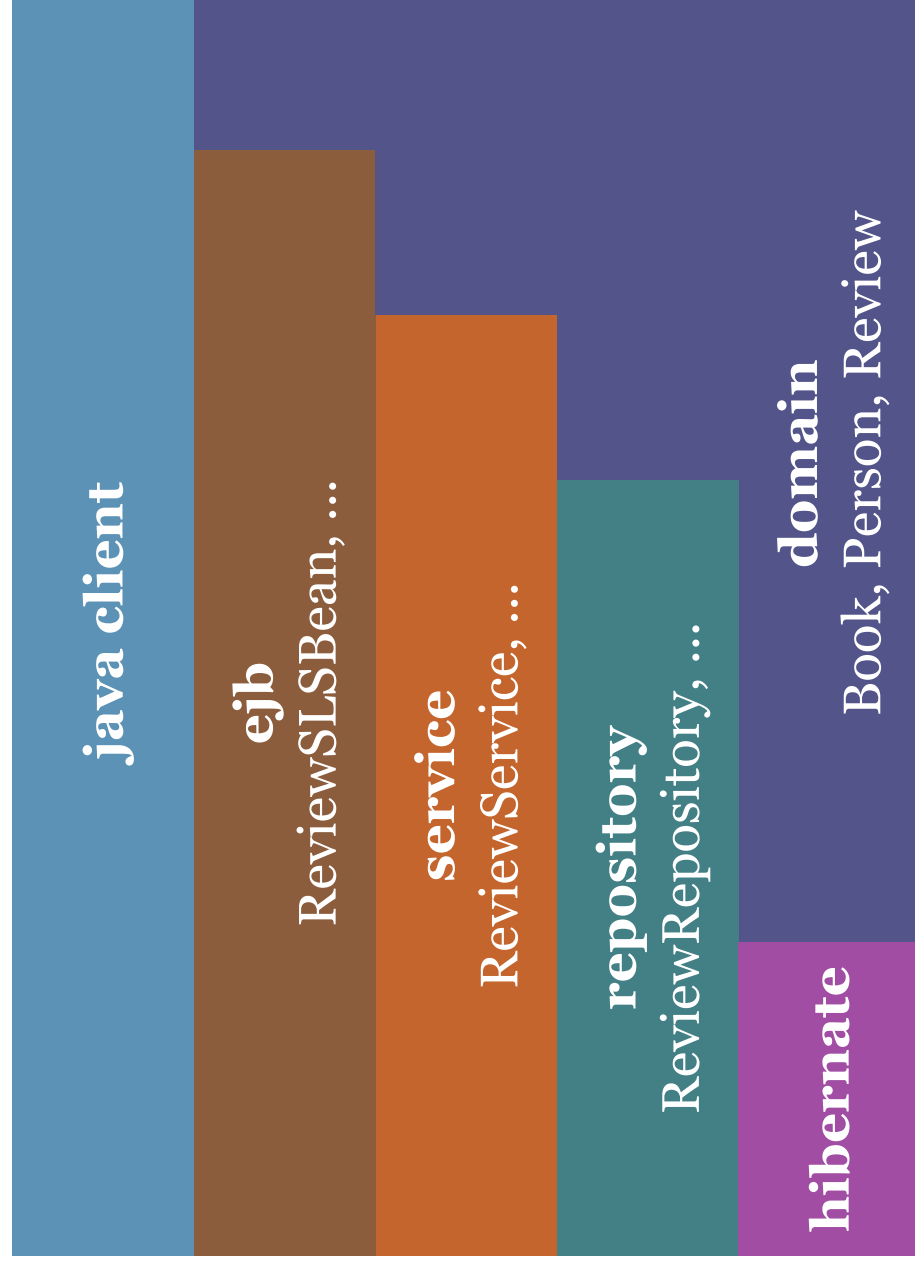
PersonService

```
List<Person> getPersonsByName(String firstName, String lastName)
```

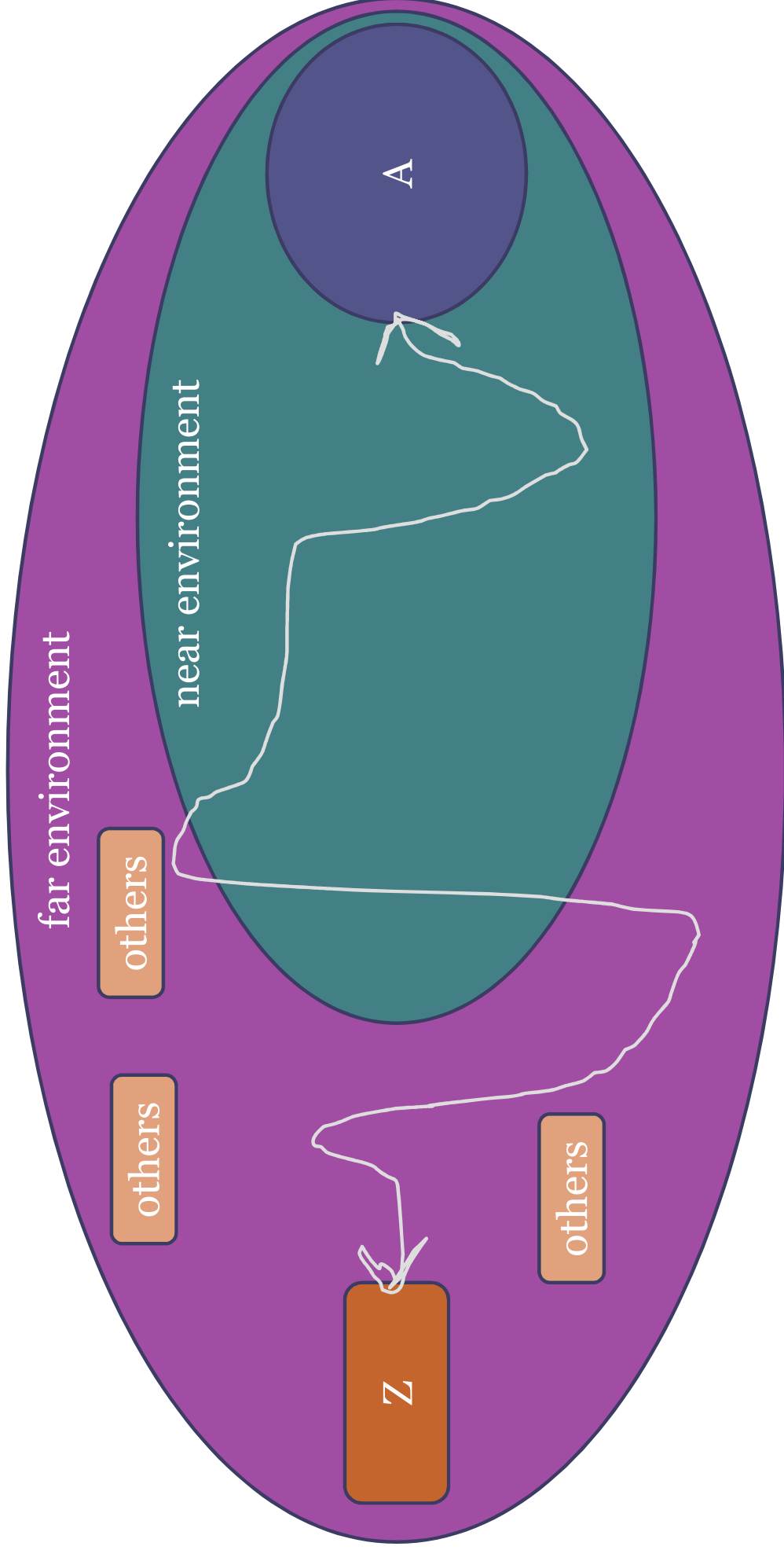
BookService

```
Book getBookByIsbn(String isbn)
```

application layers



a systems map of this integration scenario



criteria for assessing integration solutions

- correctness
- elegance
- development/maintenance/operation effort (i.e., €£\$)
- resilience to infrastructure outages
- runtime performance
- extent of coupling ...

different forms of coupling

- (degree of) coupling := the extent of (implicit or explicit) assumptions between communication partners
 - no communication without assumptions!
 - looser coupling results in
 - increased *resilience to change* of a communication partner
 - broken communication going undetected
- in time (absolute and difference)
 - partner has to be available *now* and every 5 seconds
- in control flow
 - A passes control to Z, and blocks until Z passes control back
- in data/wire formats
 - endianness, iiop, varchar field length

different forms of coupling

- in programming language
 - java/jvm, is json really javascript?
- in exchange of structural meta-data
 - java classes/files, client-side stubs, xml-schemata, dtlds
- in reliance on shared state
 - session/conversation, shared database and keys (!)
- in semantics of data or communication
 - querying for the spouse of a man returns a woman
 - the last 6 digits of the ssn consist of the date of birth
 - returning null from a query signals illegal input, returning an empty list means no results
 - first invoke service S1 before invoking any other service

does use of xml/ws lead to loose coupling?

- it may - but consider that these honourable techniques increase coupling:
 - validation against a common schema, possibly versioned
 - compile-time binding to java artefacts (jax-ws, jaxb, ...)
 - use of "deep" xpath expressions
 - request-response interactions
 - reliance on ws-rm, ws-security (incl. ciphers)
 - inference of dateTime time zones
 - sharing data models: ubl, ...
 - making assumptions about semantically under-defined aspects of the service data model

a "static fanatic"'s use of xml payloads

- cherish xml as human-readable/editable and language-agnostic – the xml payload is a primary artifact
- paranoid of misinterpretations of xml payloads that are detected late (at runtime rather than compile-time) or not at all
- prefer strict xml-schema over lax xml-schema over dtd over no schema
 - always validate each message at least once against its schema
- always try to use static (compile-time) xml-java binding (jaxb) and operate on xml documents either through their bound java objects or through xslt/xpath
 - but always fine-tune xml-schema making full use of its features

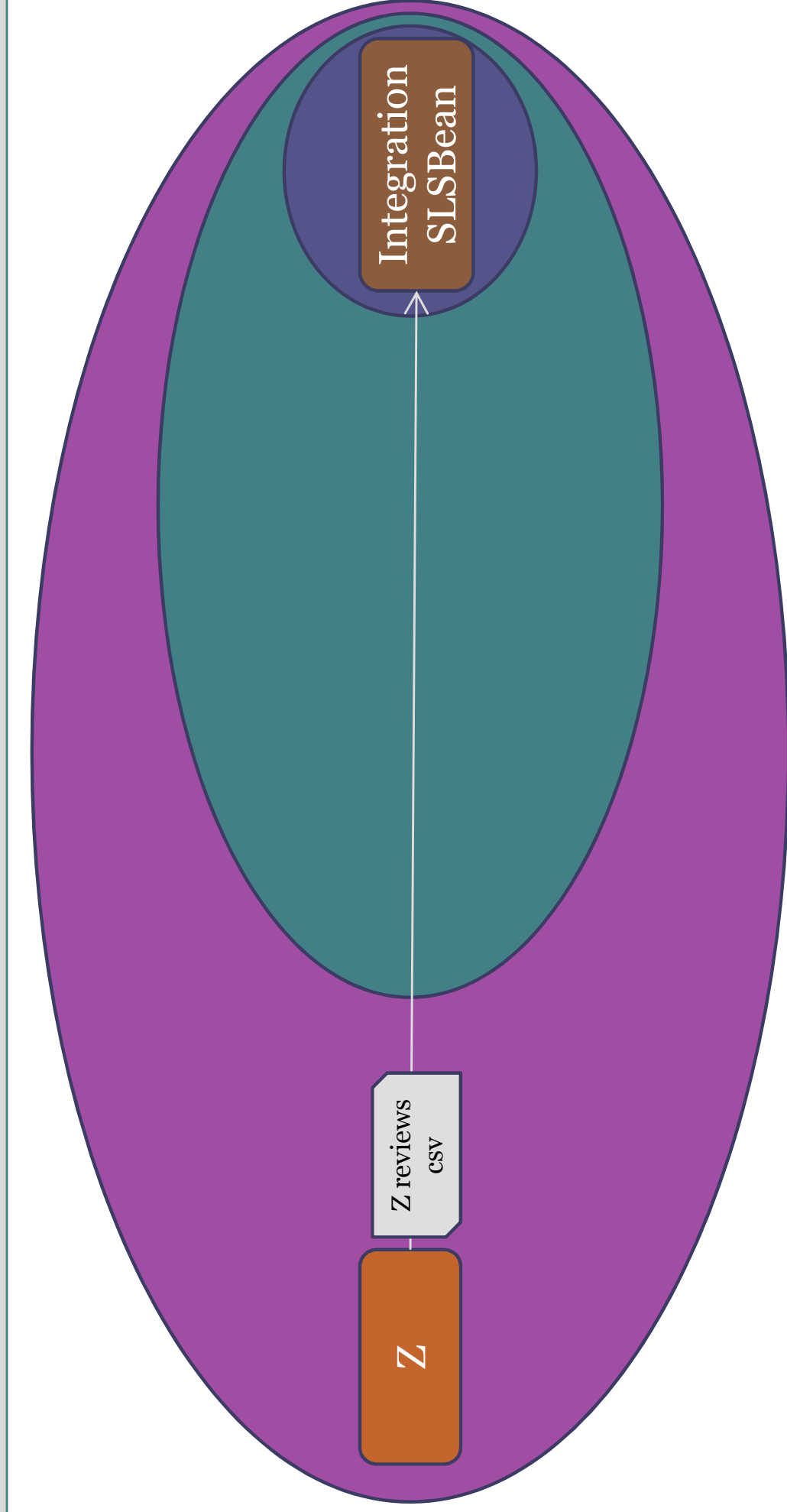
step 1: file upload

Z reviews csv, direct ejb, mdb, monolithic esb, distributed esb

the Z reviews csv file format

reviewer	book id	mark	review
First1 Last1	Z1	1	Best ever!
First3 Last3	Z2	5	Simply Terrible
Un Known	Z1	2	Good by nobody
First2 Last2	Z3	2	Quite OK
	Zx	1	Good!

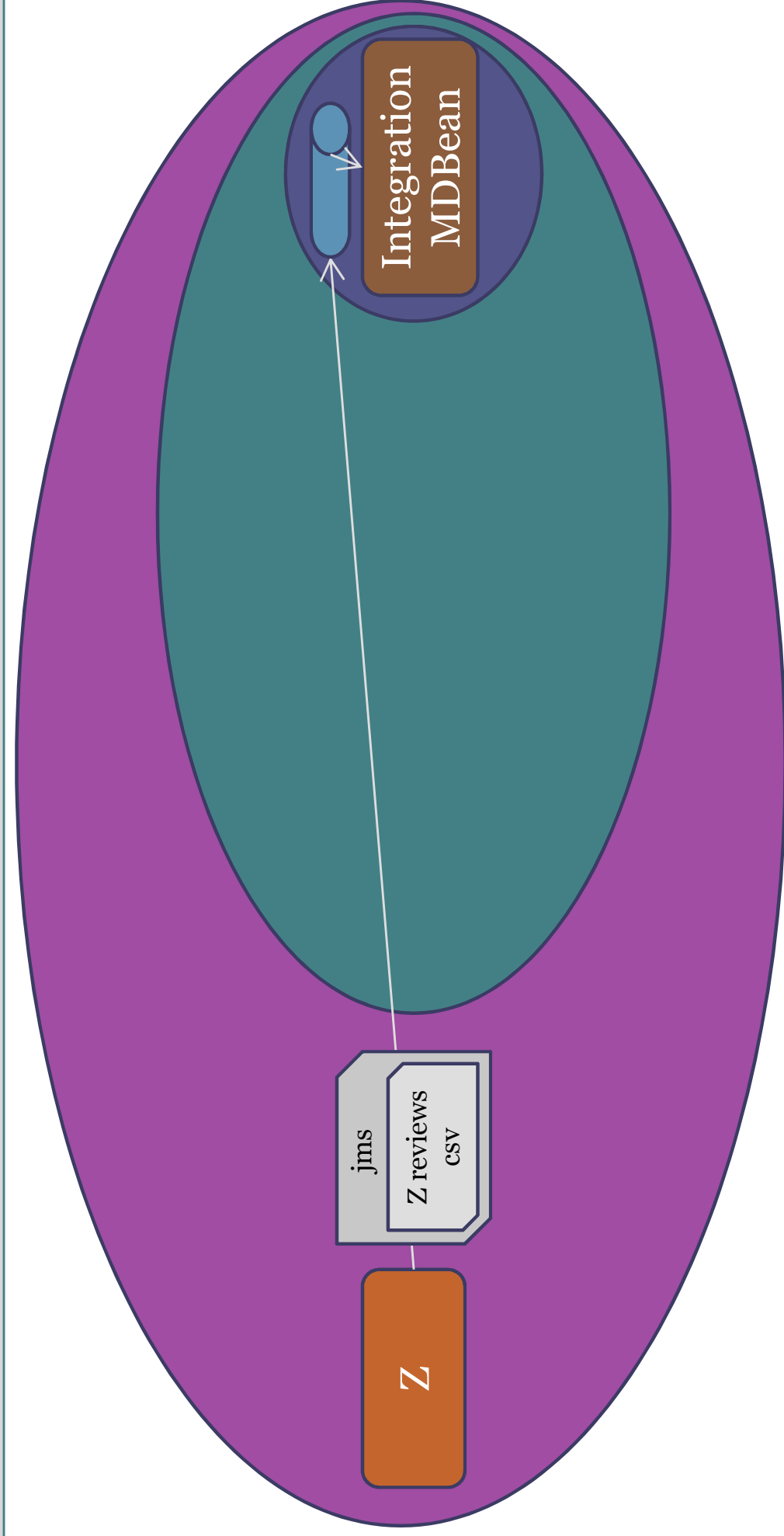
solution 1: direct ejb call



solution 1: characteristics

- need to change application to add IntegrationSLSBean
- iiop or jrmp not firewall-friendly
 - need http tunnelling or similar
- Zsolnay and our system are tightly coupled in almost every respect, in particular:
 - change to file format => change/redeployment of our app
 - need to ship client jar and app server client jars to Z
 - synchronous call
 - risk tx and/or protocol timeout with large files
- strongly affected by infrastructure outages
- easy to develop and test
- deterministic performance characteristics

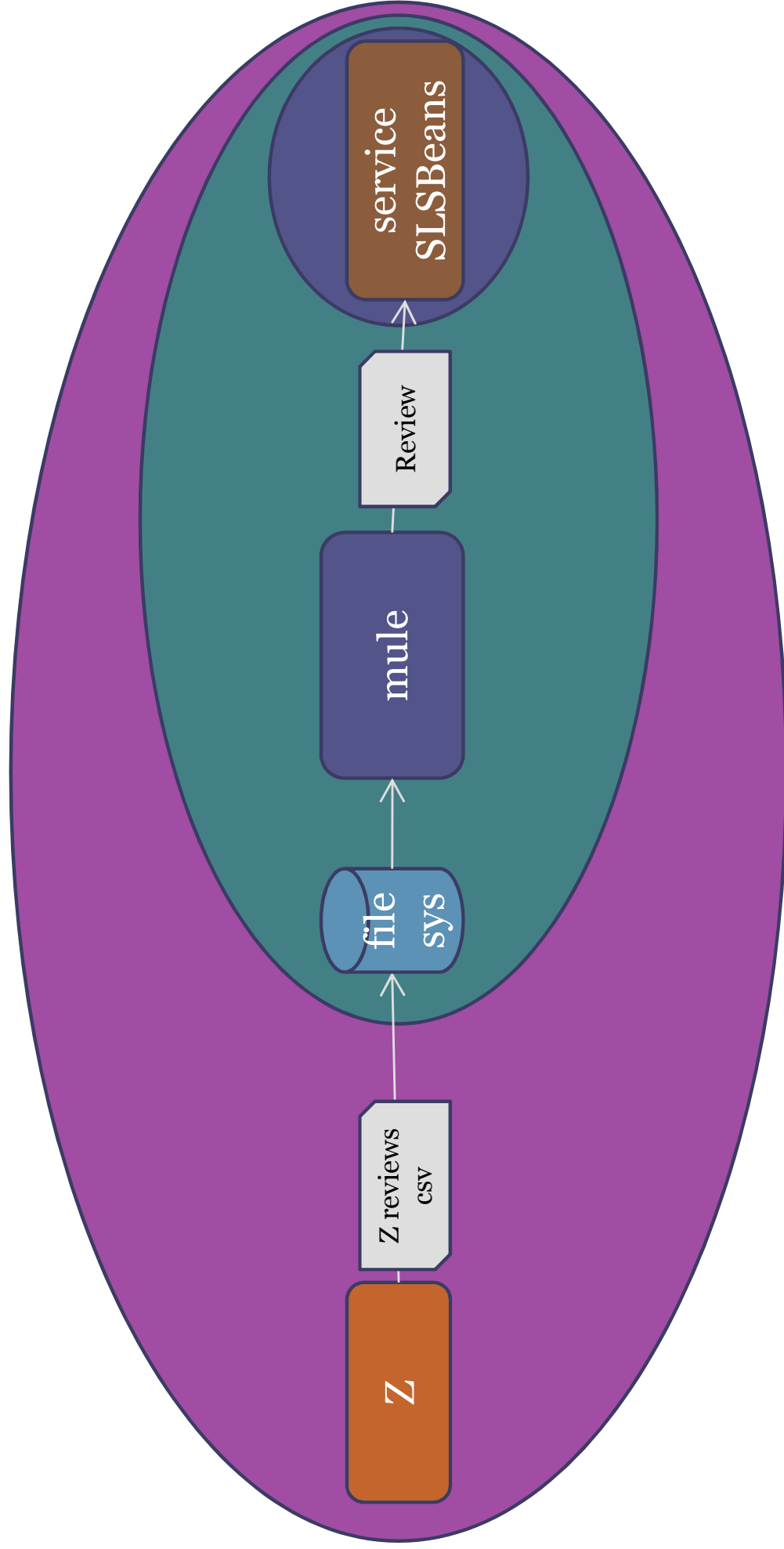
solution 2: mdb-based implementation



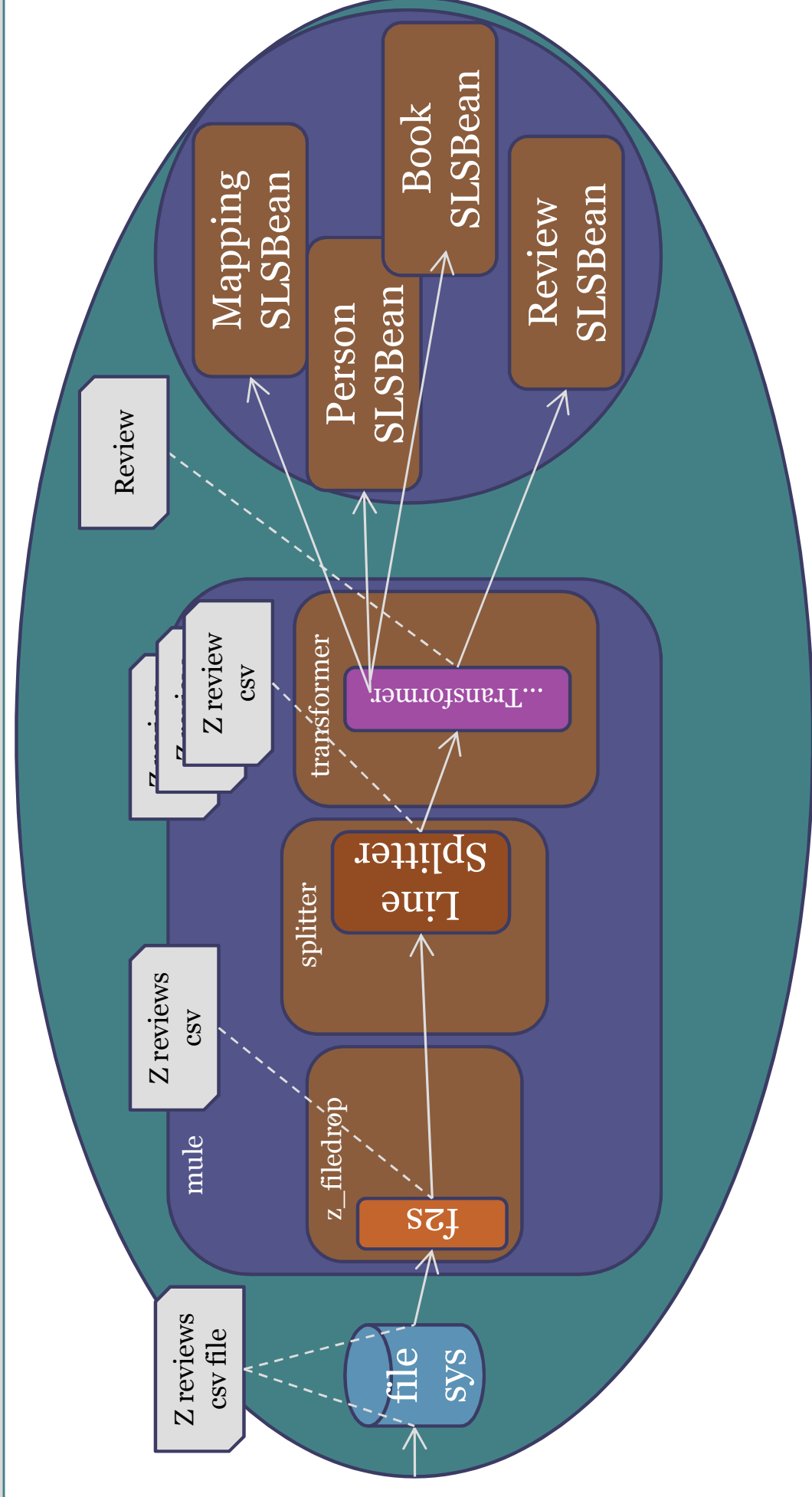
solution 2 compared to solution 1

- no need to ship client jar to Z (but app server client jars)
- asynchronous delivery and processing
 - no risk of protocol timeout on large files
 - still risk of tx timeout if mdb processes in one tx (cmt)
- still strongly affected by infrastructure outages!
 - jndi lookups and message send are still synchronous remote calls; jndi-bound objects may not survive app server restart
- harder to test
- risk of incorrectness introduced?
 - messages may be re-ordered by jms if not sent within one jms session; may be processed concurrently by mdb instances
- less deterministic performance

solution 3: monolithic esb



solution 3: monolithic esb in more detail



mule xml configuration file

```
<mule-configuration>  
  <environment-properties />  
  <mule-environment-properties />  
  <container-context />  
  <connector />  
  <transformers />  
  <global-endpoints />  
  <interceptor-stack />  
  
  <model>  
    <exception-strategy />  
    <mule-descriptor />  
  </model>  
</mule-configuration>
```

environment properties

```
<environment-properties>
```

```
  <property name="to_submit_dir" value="to_submit" />
```

```
  <property name="submitted_dir" value="submitted" />
```

```
  <property name="jndi_initial_factory"
```

```
    value="org.jnp.interfaces.NamingContextFactory" />
```

```
  <property name="jndi_provider_url" value="//localhost:1099" />
```

```
</environment-properties>
```

- referenced with `${name}`
 - in all value and address attributes

mule client-mode & connection strategy

```
<mule-environment-properties clientMode="true">
  <connection-strategy
    className="org.mule.providers.SimpleRetryConnectionStrategy">
    <properties>
      <property name="retryCount" value="3600" />
      <property name="frequency" value="1000" />
    </properties>
  </connection-strategy>
</mule-environment-properties>
```

- enable client-mode unless server-mode is needed
 - the `MuleClient` remote dispatcher needs server-mode
- connection strategy deals with exceptions in connectors
 - highly recommended when remote calls are involved

container context and spring integration

```
<container-context
  className='org.mule.extras.spring.SpringContainerContext' >
  <properties>
    <property name="configFile"
      value="springContext-mono1ith.xml" />
  </properties>
</container-context>
```

- mule containers are external component & object factories
 - `object/beans` referenced with `implementation` attribute
- other container context implementations
 - `jndi tree`, `session ejbs`, `pico`, `plexus`, `hivemind [1]`
- mule can also be configured and launched entirely from a spring context (without a mule config xml)!

file connector

```
<connector name="file_connector"
  className="org.mule.providers.file.FileConnector">
  <properties>
    <property name="moveToDirectory" value="{{submitted_dir}}" />
    <property name="moveToPattern"
      value="{{ORIGINALNAME}}.{{DATE:yyyy-MM-dd_HH-mm-ss-SSS}}" />
    <property name="binary" value="false" />
    <property name="autoDelete" value="false" />
  </properties>
</connector>
```

- impossible (?) to configure it to not move/delete files when outbound communication fails
 - make outbound communication impossible to fail

ejb (slsb) connector

```
<connector name="ejb_connector"
  className="org.mule.providers.ejb.EjbConnector">
  <properties>
    <property name="jndiInitialFactory"
      value="{{jndi_initial_factory}}"/>
    <property name="jndiProviderUrl" value="{{jndi_provider_url}}"/>
    <property name="securityPolicy" value="security.policy"/>
  </properties>
</connector>
```

- for calling slsb method through mule endpoints
 - from outbound or nested routers (but prefer spring to latter)
- ejb container context is an "early binding" alternative
- undocumented lookup/caching behaviour fails on reconnect
 - prefer spring jee:remote-slsb whenever possible

jms connector

```
<connector name="jms_connector"
  className="org.mule.providers.jms.JmsConnector">
<properties>
  <property name="jndiInitialFactory"
    value="{jndi_initial_factory}" />
  <property name="jndiProviderUrl" value="{jndi_provider_url}" />
  <property name="connectionFactoryJndiName"
    value="ConnectionFactory" />
  <property name="jndiDestinations" value="true" />
  <property name="forceJndiDestinations" value="true" />
  <property name="persistentDelivery" value="true" />
  <property name="specification" value="1.1" />
  <property name="maxRedelivery" value="5" />
</properties>
</connector>
```

transformers

```
<transformers>
  <transformer name="f2s"
    className="org.mule.providers.file.transformers.FileToString" />
  <transformer name="o2jms" className=
    "org.mule.providers.jms.transformers.ObjectToJMSMessage" />
</transformers>
```

- infrastructural, technical transformers provided by mule
- more business-related transformers written in projects
 - extend `org.mule.transformers.AbstractTransformer`
 - not a spring bean if configured via mule xml
 - simple transformation => transformer, complex => component
 - see later for example

endpoints

```
<global-endpoints>
  <endpoint name="to_submit_dir" address="file://${to_submit_dir}"
    connector="file_connector" transformers="f2s" />
  <endpoint name="z_reviews_csv" address="//z_reviews_csv" />
  <endpoint name="review"
    address="ejb:/ReviewLSBean?method=addReview"
    connector="ejb_connector" remoteSync="true" />
  <endpoint name="invalid_msg"
    address="jms://queue/INT.INVALID_MESSAGE"
    connector="jms_connector" transformers="o2jms" />
</global-endpoints>
```

- central vehicle for mules protocol/transport abstraction
- best defined as global endpoints
- only define fixed transformers => can amend at use time

interceptors

```
<interceptor-stack name="default">  
  <interceptor className="util.esb.mule.LoggingInterceptor" />  
</interceptor-stack>
```

```
public class LoggingInterceptor extends EnvelopeInterceptor {  
  
    @Override public void before(Invocation i) {...}  
    @Override public void after(Invocation i) {...}  
}
```

- cf. aop around advice
- applies to component invocations
 - not invoked for `BridgeComponent`
 - use `PassThroughComponent` instead
- for debugging, but also for message store/history/audit [2]

component exception strategy

```
<exception-strategy
  className="util.esb.mule.ExceptionAwareExceptionStrategy">
  <global-endpoint name="invalid_msg" />
  <global-endpoint name="manual_intervention" />
</exception-strategy>
```

```
public class ExceptionAwareExceptionStrategy implements
  ExceptionListener {
  public void exceptionThrown(Exception e) {...}
}
```

- handles exceptions thrown by components
 - usually business exceptions, i.e. project-dependent
 - a special form of synchronous content-based routing for exceptions

read file, pass on as string payload

```
<mule-descriptor name="z_filedrop"
  implementation="org.mule.components.simple.PassThroughComponent">
  <inbound-router>
    <global-endpoint name="to_submit_dir" />
  </inbound-router>
  <outbound-router>
    <router className=
      "org.mule.routing.outbound.OutboundPassThroughRouter">
      <global-endpoint name="z_reviews_csv" />
    </router>
  </outbound-router>
  <interceptor name="default" />
</mule-descriptor>
```

split csv payload into 1-line csv messages

```
<muLe-descriptor name="splitter" implementation=
  "org.mule.components.simple.PassThroughComponent">
  <inbound-router>
    <global-endpoint name="z_reviews_csv" />
  </inbound-router>
  <outbound-router>
    <router className="util.esb.mule.Linesplitter">
      <global-endpoint name="z_review_csv" />
    <properties>
      <property name="preserveHeader" value="true" />
    </properties>
  </router>
</outbound-router>
</muLe-descriptor />
```

transform csv into Review, invoke ejb

```
<mule-descriptor name="transformer"
  implementation="zReviewCsvToReviewTransformer">
  <inbound-router>
    <global-endpoint name="z_review_csv" />
  </inbound-router>
  <outbound-router>
    <router className=
      "org.mule.routing.outbound.OutboundPassThroughRouter">
      <global-endpoint name="review" />
    </router>
  </outbound-router>
  <interceptor name="default" />
</mule-descriptor>
```

springContext-monolith.xml

```
<bean id="zReviewCsvToReviewTransformer" scope="prototype"
class="bookreview.integration.esb.ZReviewCsvToReviewTransformer">
  <property name="mappingService" ref="mappingSLSB" />
  <property name="personService" ref="personSLSB" />
  <property name="bookService" ref="booksLSB" />
</bean>
<jee:remote-slsb id="booksLSB" jndi-name="BookSLSBean"
business-interface="bookreview.service.BookService"
home-interface="bookreview.ejb.BookSLSBHome"
lookup-home-on-startup="false" cache-home="true"
refresh-home-on-connect-failure="true">
  <jee:environment>
    ...
  </jee:environment>
  ...
</jee:remote-slsb>
```

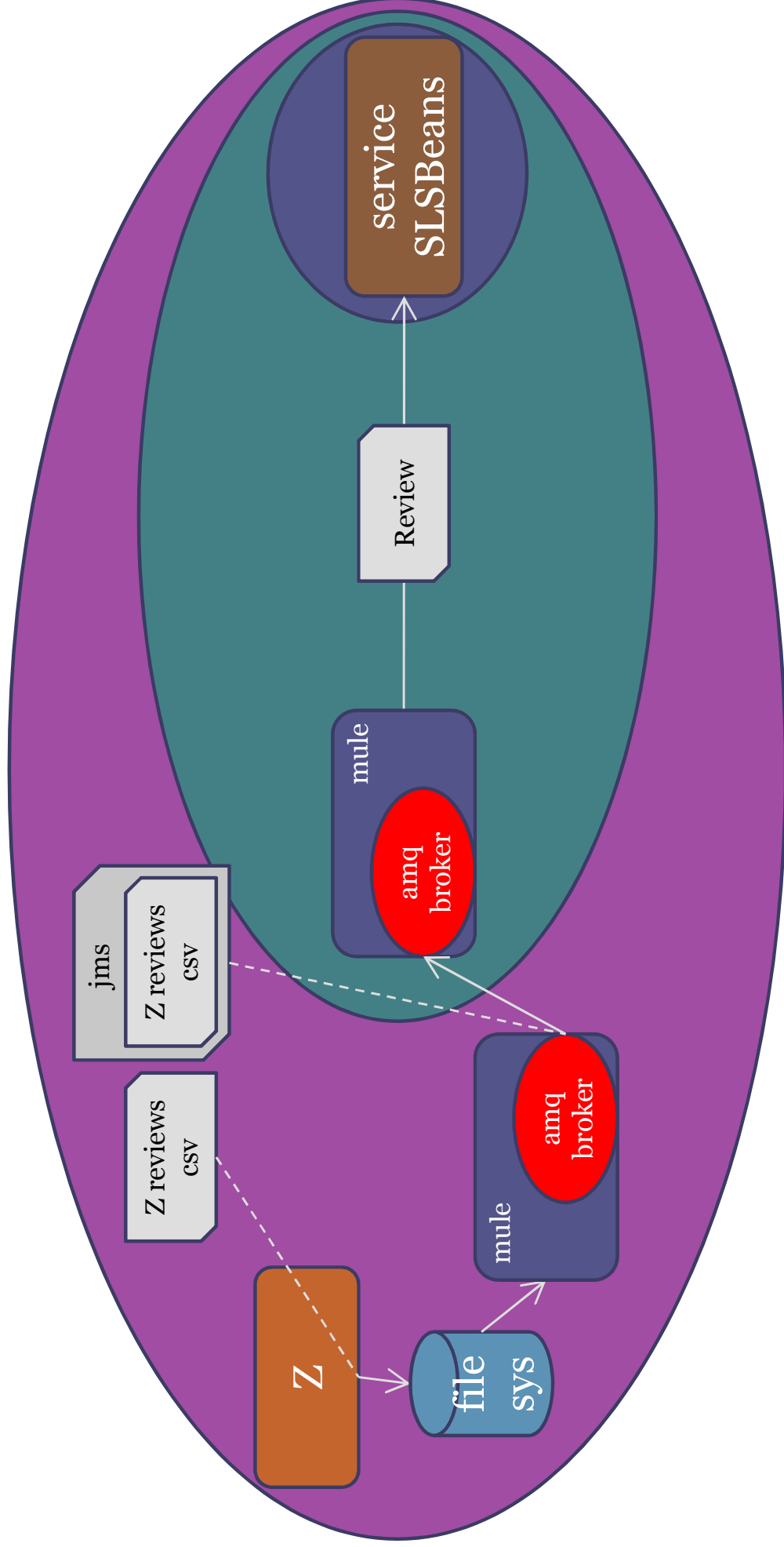
a transforming component using spring di

```
public class ZReviewCsvToReviewTransformer {  
  
    public void setMappingService(MappingService mappingService) {  
        this.mappingService = mappingService;  
    }  
  
    public void setPersonService(PersonService personService) {  
        this.personService = personService;  
    }  
  
    public void setBookService(BookService bookService) {  
        this.bookService = bookService;  
    }  
  
    public Review transform(String csv)  
        throws InvalidMessageException, MappingException {...}  
  
    ...  
}
```

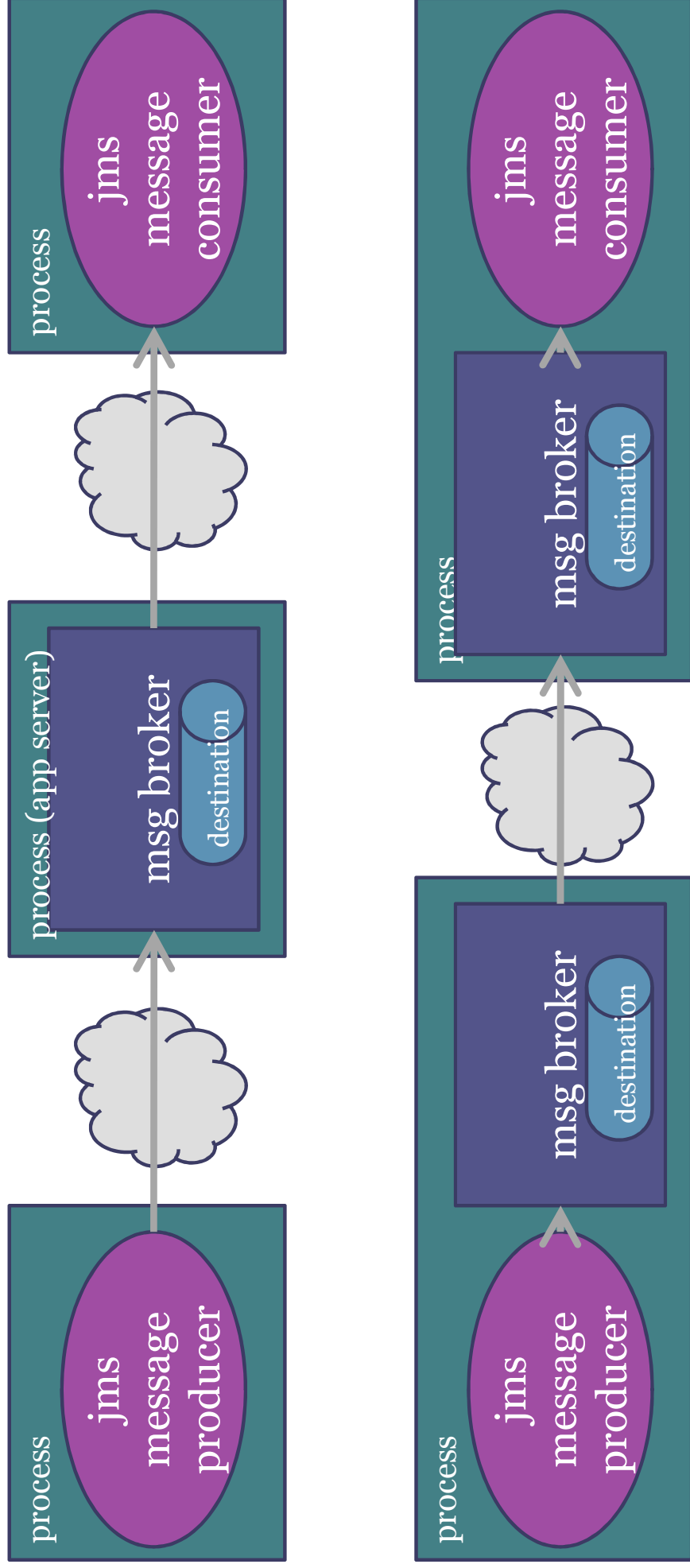

characteristics of the monolithic esb

- no need to change application once MappingService added
 - acceptance and transformation of Z review csv occurs in stand-alone mule outside of application
 - mule calls several slsbs exposed by the application
 - mule needs client jar and app server client jars
 - many fine-grained synchronous remote ejb calls!
 - each csv line transformed and imported separately
 - ordering may get lost
- Z now "just" needs to push csv file to us
- identified need for mapping service => see later
 - accessed as remote slsb
- a fragile, strongly coupled, very chatty solution

solution 4: distributed esb



store-and-forward vs. client-server messaging



principles behind a fail-safe distributed esb

- assume that remote connections will fail
- let infrastructure cope with failed connections
 - store-and-forward messaging architecture
- send locally, receive locally or remotely:
 - only send to local (in-process) message broker
 - never left stranded with a message that can not be sent
 - prefer to receive from local message broker but can consume from arbitrary remote endpoints if required
 - if connectivity fails then message processing never even starts
- synergistic deployment of mule with embedded active mq message broker (acting as jms provider)

Z-side mule xml config

```
<container-context
  className='org.mule.extras.spring.SpringContainerContext' >
  <properties>
    <property name="configFile" value="springContext-dist1.xml" />
  </properties>
</container-context>
<connector name="jms_connector"
  className="org.mule.providers.jms.activemq.ActiveMqJmsConnector">
  <properties>
    <property name="persistentDelivery" value="true" />
    <property name="specification" value="1.1" />
    <property name="maxRedelivery" value="5" />
    <container-property name="connectionFactory"
      reference="amqConnectionFactory" />
  </properties>
</connector>
```

Z-side spring context & active mq config

```
<bean id="amqConnectionFactory"
    class="org.apache.activemq.ActiveMQConnectionFactory">
    <constructor-arg
        value="vm://localhost?brokerConfig=xbean:activemq-distr1.xml" />
    </bean>

<broker brokerName="amq-distr1" persistent="true" useJmx="false"
    xmlns="http://activemq.org/config/1.0">
    <transportConnectors>
        <transportConnector uri="tcp://localhost:55501" />
    </transportConnectors>

    <networkConnectors>
        <networkConnector uri="static://(tcp://localhost:55502)" />
    </networkConnectors>
</broker>
```

A-side mule xml config

```
<container-context
  className='org.mule.extras.spring.SpringContainerContext' >
  <properties>
    <property name="configFile" value="springContext-distr2.xml" />
  </properties>
</container-context>
<connector name="jms_connector"
  className="org.mule.providers.jms.activemq.ActiveMqJmsConnector">
  <properties>
    <property name="persistentDelivery" value="true" />
    <property name="specification" value="1.1" />
    <property name="maxRedelivery" value="5" />
    <container-property name="connectionFactory"
      reference="amqConnectionFactory" />
  </properties>
</connector>
```

A-side spring context & active mq config

```
<bean id="amqConnectionFactory"
    class="org.apache.activemq.ActiveMQConnectionFactory">
    <constructor-arg
        value="vm://localhost?brokerConfig=xbean:activemq-distr2.xml" />
    </bean>

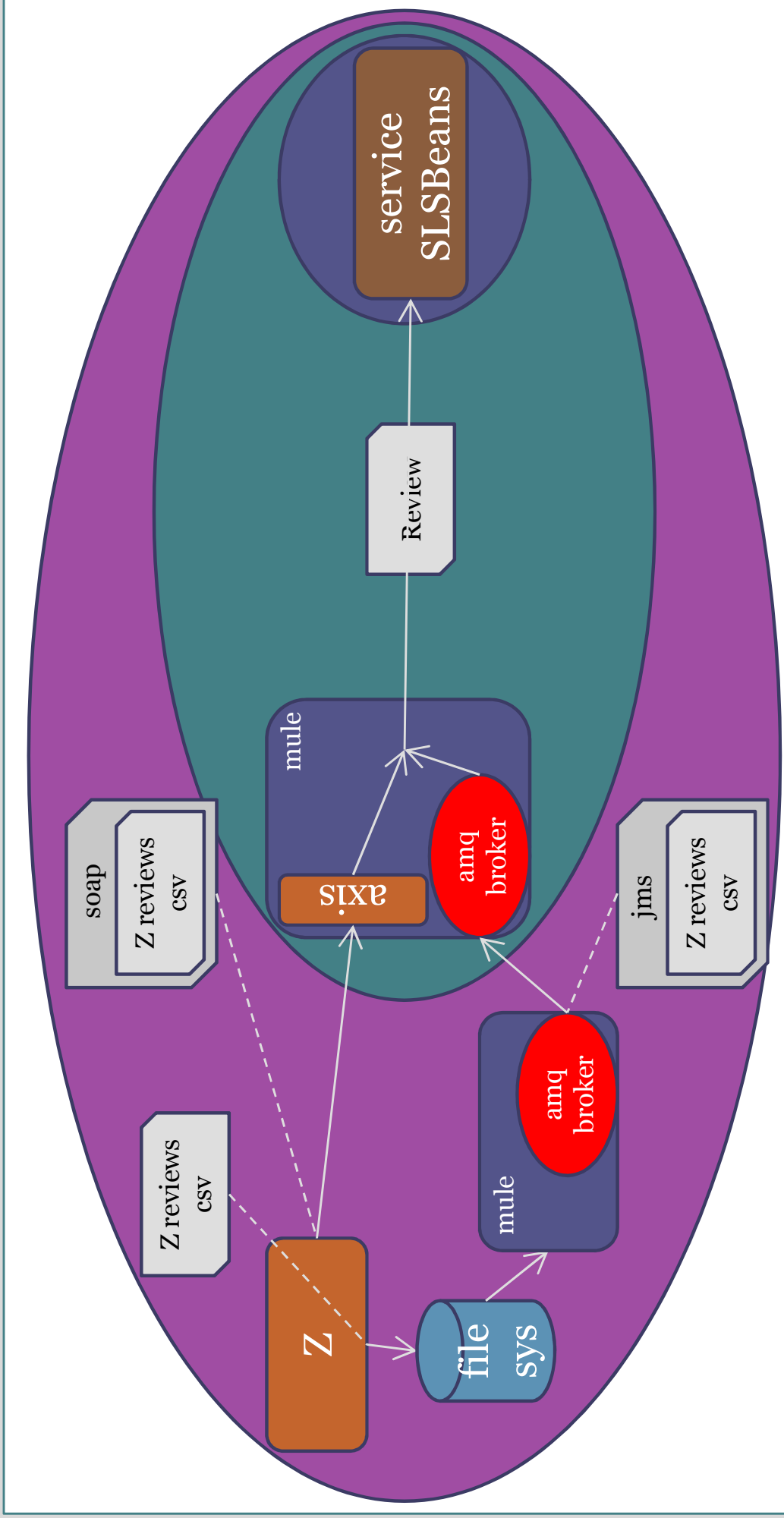
<broker brokerName="amq-distr2" persistent="true" useJmx="false"
    xmlns="http://activemq.org/config/1.0">
    <transportConnectors>
        <transportConnector uri="tcp://localhost:55502" />
    </transportConnectors>

    <networkConnectors>
        <networkConnector uri="static://(tcp://localhost:55501)" />
    </networkConnectors>
</broker>
```

step 2: data import via ws

mule axis provider

adding an axis endpoint to the distributed esb



axis webservice endpoint & transformer

```
<global-endpoints>  
  <endpoint name="ws_root"  
    address="axis:http://localhost:8008/service" />  
</global-endpoints>
```

```
<transformers>  
  <transformer name="args2s"  
    className="util.esb.mule.MethodArgsToFirstStringArgument" />  
</transformers>
```

exposing a component as a soap webservice

```
<mule-descriptor name="reviewsGateway"
  implementation="bookreview.integration.esb.ReviewsGatewayImpl">
  <inbound-router><global-endpoint name="ws_root" /></inbound-router>
  <outbound-router>
    <router className="...OutboundPassThroughRouter">
      <global-endpoint name=".." transformers="args2s o2jms" />
    </router>
  </outbound-router>
</properties>
<property name="style" value="wrapped" />
<property name="use" value="Literal" />
<property name="serviceName" value="urn:bookreview:service"/>
</properties>
</mule-descriptor>
```

the wsdl-defining component and interface

```
public interface ReviewsGateway {  
  
    void submitReviewsInTextFormat(String reviews);  
}  
  
public class ReviewsGatewayImpl implements ReviewsGateway {  
  
    public void submitReviewsInTextFormat(String reviews) {  
        // do not alter message  
    }  
}
```

a transformer that extracts the first string arg

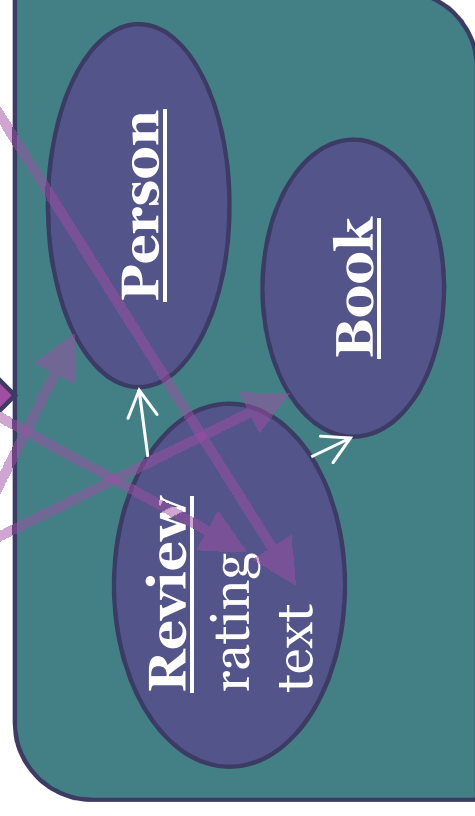
```
public class MethodArgsToFirstStringArgument extends
    AbstractTransformer {
    public MethodArgsToFirstStringArgument() {
        registerSourceType(Object[].class);
        setReturnClass(String.class);
    }
    @Override
    protected Object doTransform(Object src, String encoding)
        throws TransformerException {
        final Object[] args = (Object[]) src;
        for (final Object arg : args) {
            if (arg instanceof String) return arg;
        }
        return null;
    }
}
```

message transformation

parsing, mapping, formatting, the data mapping service

our message transformation problem

reviewer	book id	mark	review
First3 Last3	Z2	5	Simply Terrible



conceptualising message transformation

- in general m input messages are to be transformed to n output messages
 - using aggregators (a framework for which is part of mule) the m input messages can always be combined to one message => 1 input is transformed to n output messages
- pre-transformation step: detecting the input message format/type and version
 - java instanceof, xml namespace uri, xml dtd, xpath expression, type field, filename pattern
 - need access to "store" of supported message formats/types
 - usually followed by content-based routing to suitable transformer

conceptualising message transformation

- steps in message transformation:
 - step 1: parsing the input message, thereby validating it
 - jaxb, java deserialisation, flatpack file parser
 - step 2: mapping input fields to output fields
 - in general m-to-n
 - restrict to 1-to-1 mappings here for sake of simplicity
 - distinguish 3 types of mapping dependent on need for access to mapping data store/services and access to application services:
 - no need for mapping data store/services
 - split "firstName lastName" into "firstName" and "lastName"
 - copy "review" field into "text" field

conceptualising message transformation

- lookup in mostly static and application-independent "reference data" mapping data store
 - mapping "mark" field to "rating" field
- lookup in dynamic and application-dependent "live data" mapping data store or invocation of lookup (finder) application services
 - mapping "book id" field to isbn or internal book id or Book object
 - lookup Book object (or id) by isbn
 - lookup Person object (or id) by "firstName" and "lastName"
- **step 3: creating/formatting the output message(s)**
 - java new or factory, jaxb, xslt output
 - steps 1&2 are dependent on the input message format
 - steps 2&3 are dependent on the output message format(s)
 - either accept that each transformer is dependent on input and output message format(s)
 - or decouple through intermediary format (not discussed here)

a simple data mapping data store and service

mapped type	from value space	from value	to value
RATING	AUSTRIAN_MARKS	1	GOOD
RATING	AUSTRIAN_MARKS	5	UGLY
ISBN	ZSOLNAY	Z1	isbn1

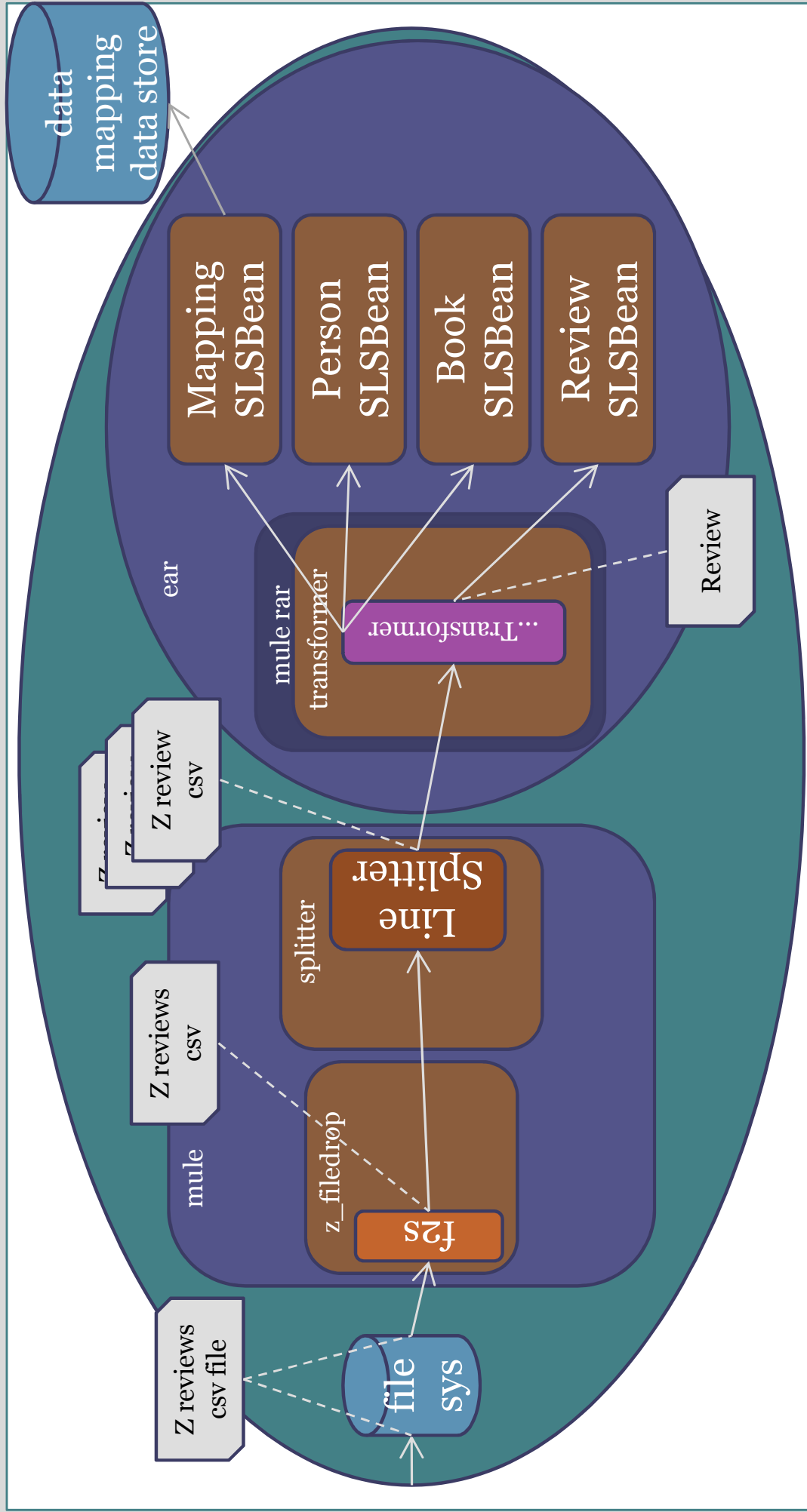
MappingService

```
String getMappedValue(MappedType type, String fromValueSpace,  
String fromValue)
```

positioning the data mapping service

- for performance reasons we do not accept many fine-grained remote calls during message transformation
 - => data mapping service and transformer must be co-located
 - access through local slsb or spring service
- 2 possible solutions:
 - implement data mapping service as an integral part of the application and deploy the transformer (and hence mule) with the application
 - mule can be deployed as a normal library (jar) within any war/ejb-jar/ear or as a jca resource adapter (rar) globally to the app server (not jboss) or within an ear
 - implementing the data mapping service as an integral part of the esb (mule) which hosts the transformer

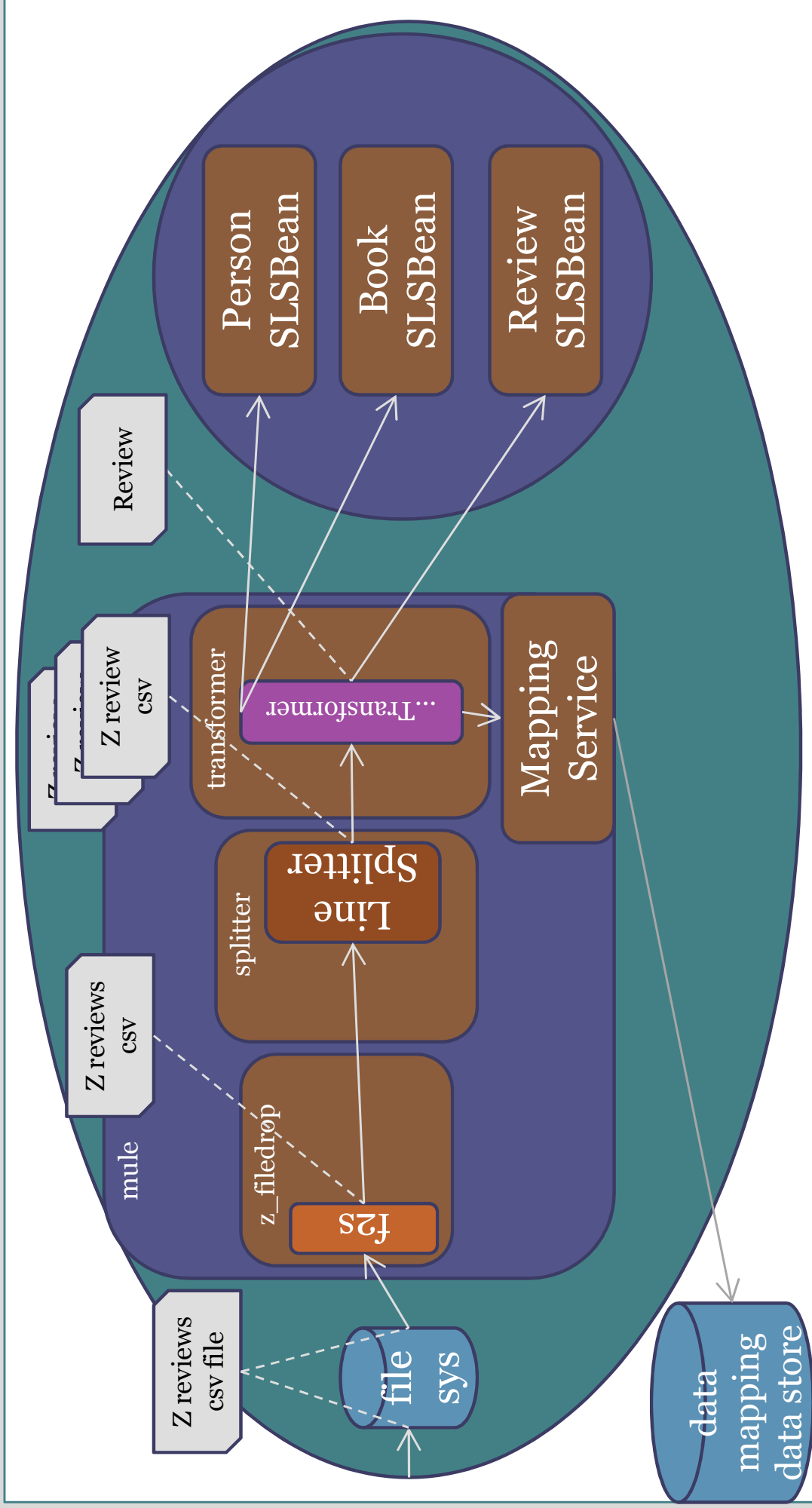
co-locating application, transformer and mule



co-locating application, transformer and mule

- not only data mapping service but also all other services are co-located
 - [PersonSLSBean](#), [BookSLSBean](#), [ReviewSLSBean](#)
- any change to a supported message format (and hence transformer) or the addition of a new message format or version thereof (and hence transformer) requires changing and re-deploying the application
- very hard to support several versions of the same message format if they are mapped to java classes of the same fully-qualified class name!
- complicates deployment considerably
 - jboss and mule have conflicting jar file dependencies

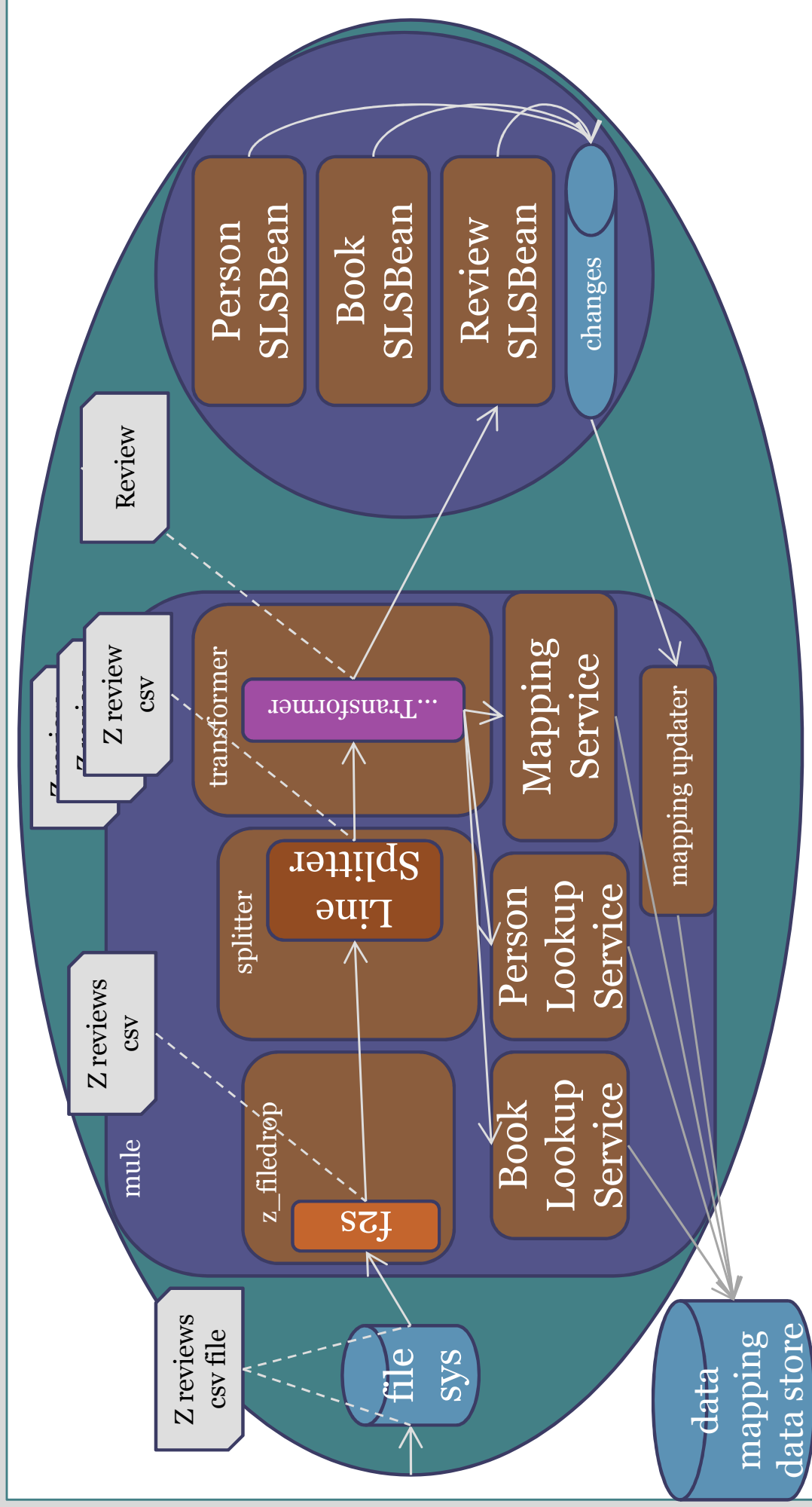
data mapping service within stand-alone mule



data mapping service within stand-alone mule

- dependencies on input message formats now confined to transformer(s) deployed in stand-alone mule
 - can modify/add message formats without changing application
- data mapping store and service now separated from application
 - can be tuned irrespective of application
 - in particular caching (and hence memory consumption) can be configured to fully benefit the mapping service!
 - but how is the dynamic "live data" part of the mapping data store kept in sync with the application?
- lookup/finder-style application services needed by transformer are still accessed remotely

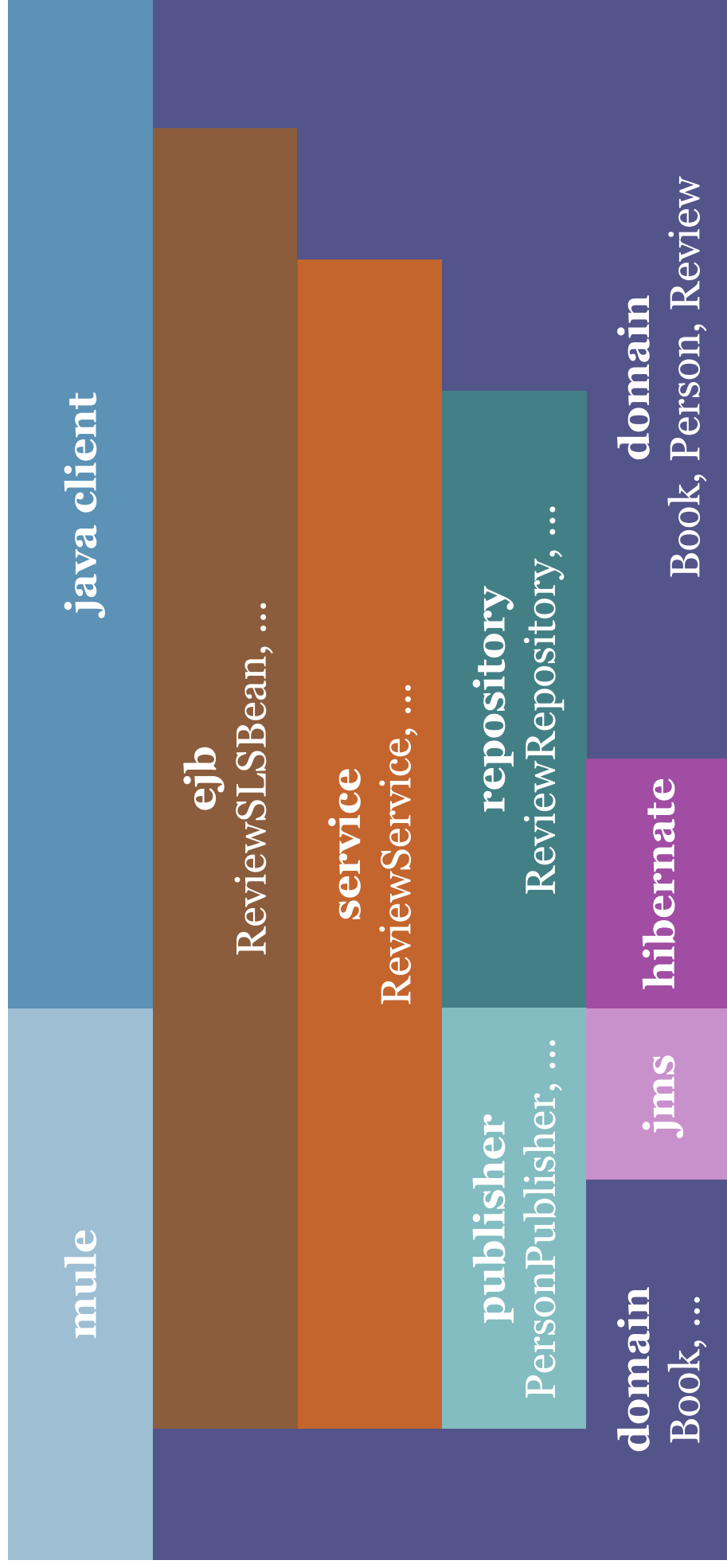
data mapping service and lookup services for application data within stand-alone mule



data mapping service and lookup services for application data within stand-alone mule

- all services needed by transformer(s) are deployed in stand-alone mule and accessible locally
- "mapping updater" component in mule subscribes to one or more change notification topics (publish-subscribe channels [2]; observer pattern) in application server where the application publishes changes to its domain model
 - "mapping updater" keeps data mapping data store in-sync with application "live data" at all times
- data mapping data store only keeps data it needs for the data mapping service and the lookup services built on it
 - table structure and caching strategy can be optimised for mapping

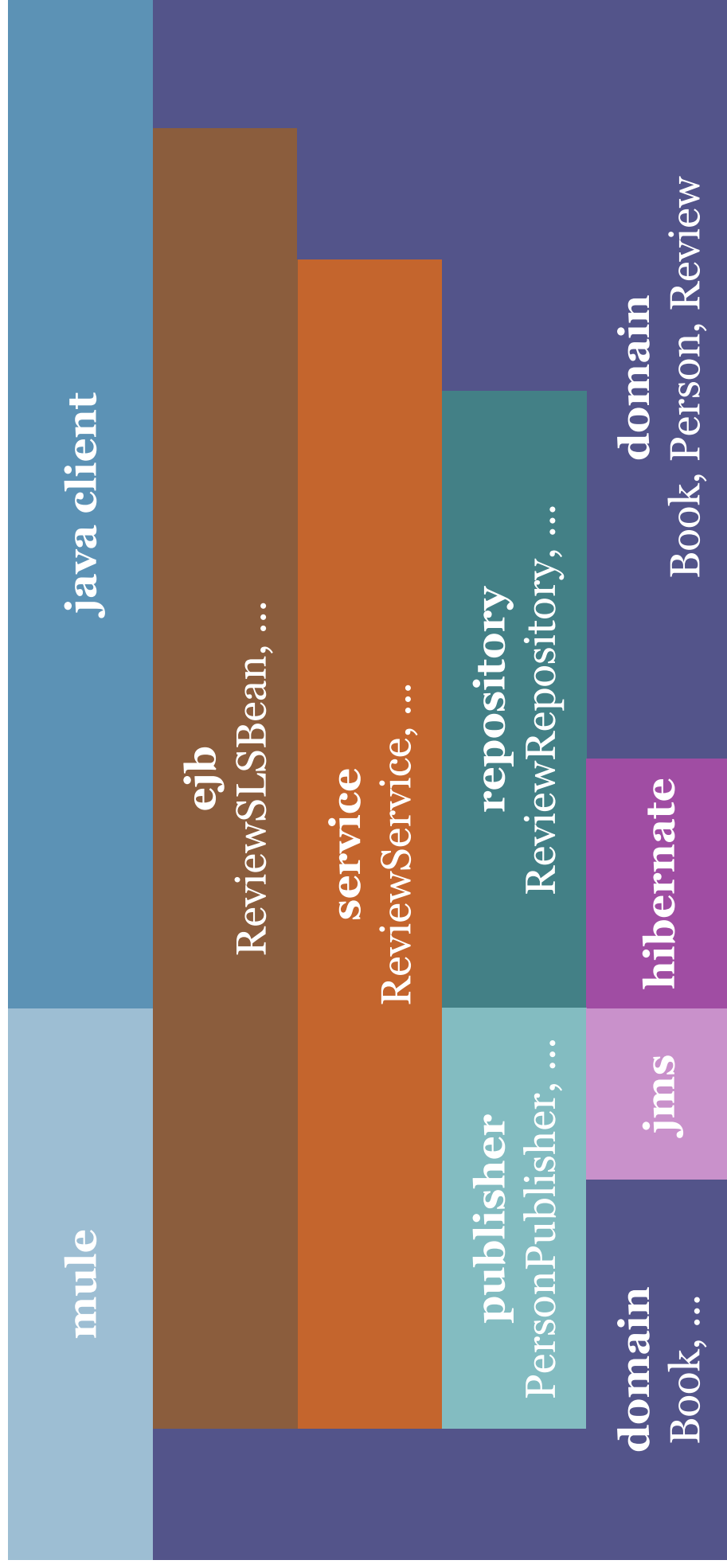
application layers w/ publisher layer



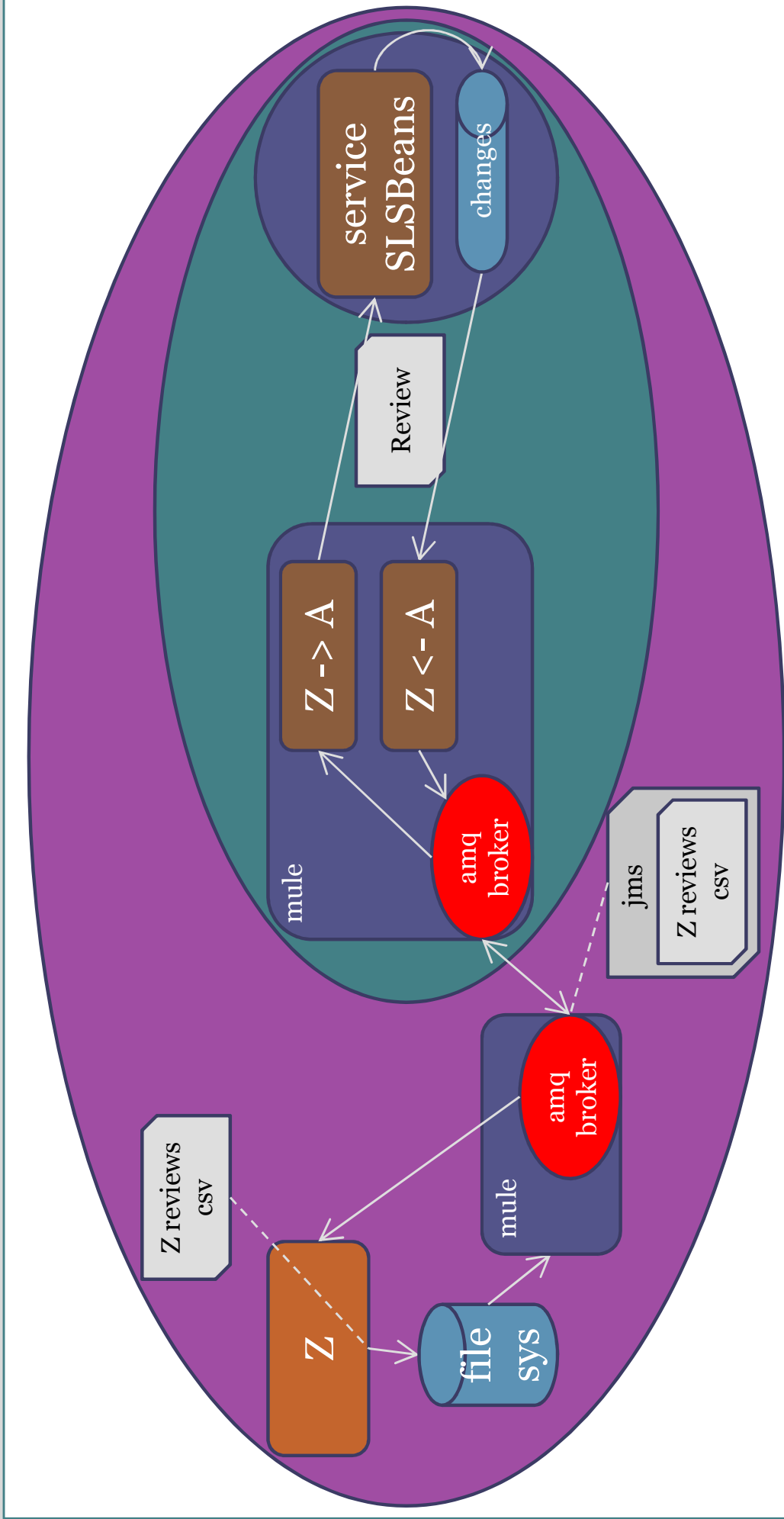
step 3: feeding data to external systems

capitalising on the publisher layer and the distributed esb

application layers supporting integration



feeding data to Z through the publisher layer



what we did not talk about

- transaction demarcation with mule
 - begin on inbound-router's endpoint
 - commit on outbound-router's endpoint
 - rollback through exception strategy
- security
 - most importantly securing access to endpoints
- operating a mule-based integration solution
 - jmx agent comes with mule
 - "console" available as a commercial product
- mule threading, queuing and component pooling
- most of the routers and components that come with mule