Mule – story of a service bus

a play in 3 acts
or
11-12 slides and
(quite) a bit of
code

Before jumping into the deep end with code –

Time to bore you with a few buzzwords and service bus concepts

- Loose coupling
- messages, payloads, properties
- Message flows
- Components,Transformers,Filters
- MEL the mule expression language
- Writing custom endpoints to emit messages
- Testing

Loose coupling

“every time you ignore loose coupling a little donkey dies”

quote lifted and stolen, uh, borrowed from -

http://blogs.mulesoft.org/every-time-you-ignore-loose-coupling-a-little-donkey-dies/

Mulesoft's CTO Ross Mason -

- What would happen if you added or deleted a property from a class?
- What would happen if you changed the data type of a column in a table?
- What would happen if you added a column to a table?
- What would happen if you changed your asynchronous email component?
- What would happen if your database server changed from Oracle to SQL Server or mySQL?

The mule message

- Lets play the old SAT game
  - the mule message is to the payload like
    an envelope is to a letter.
  - A property is to the message like
    an address/stamp/signature or heck a wax seal on an envelope
  - Message Queues or JMS broker is to a mule endpoint like
    the postman is to the mailbox.

Message Flows

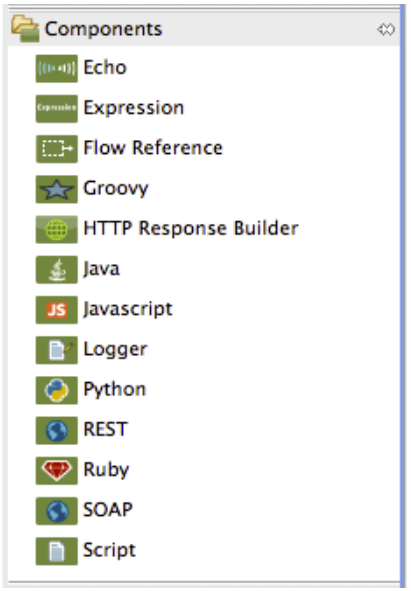
- Messages are processed in “flows”.
- Emitted by 'Sources'
- transformed or processed by components.
- routed and filtered using filters and controls.
- a more precise word for flows is scope.
- flows are represented in mulestudio as scopes.

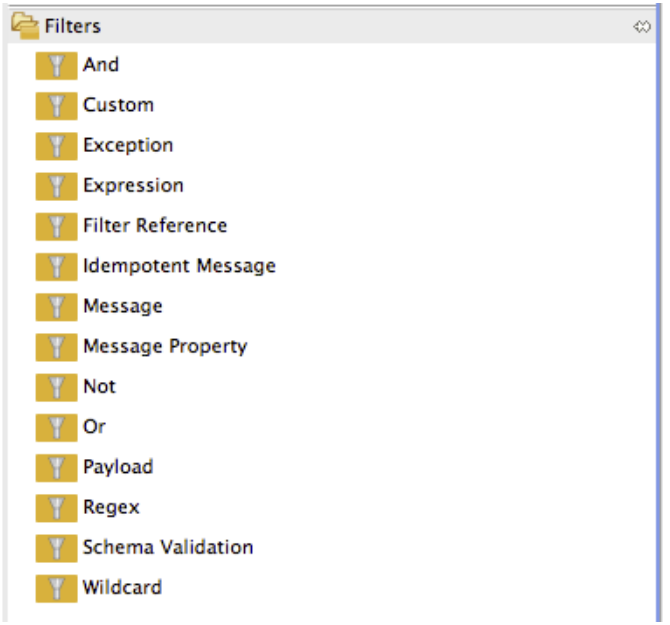
EH? Yeah, I know. Caches, async blocks, foreach constructs, sub-flow, polling scopes, transactions are all managed scopes executed within a flow which in and of itself is a scope.

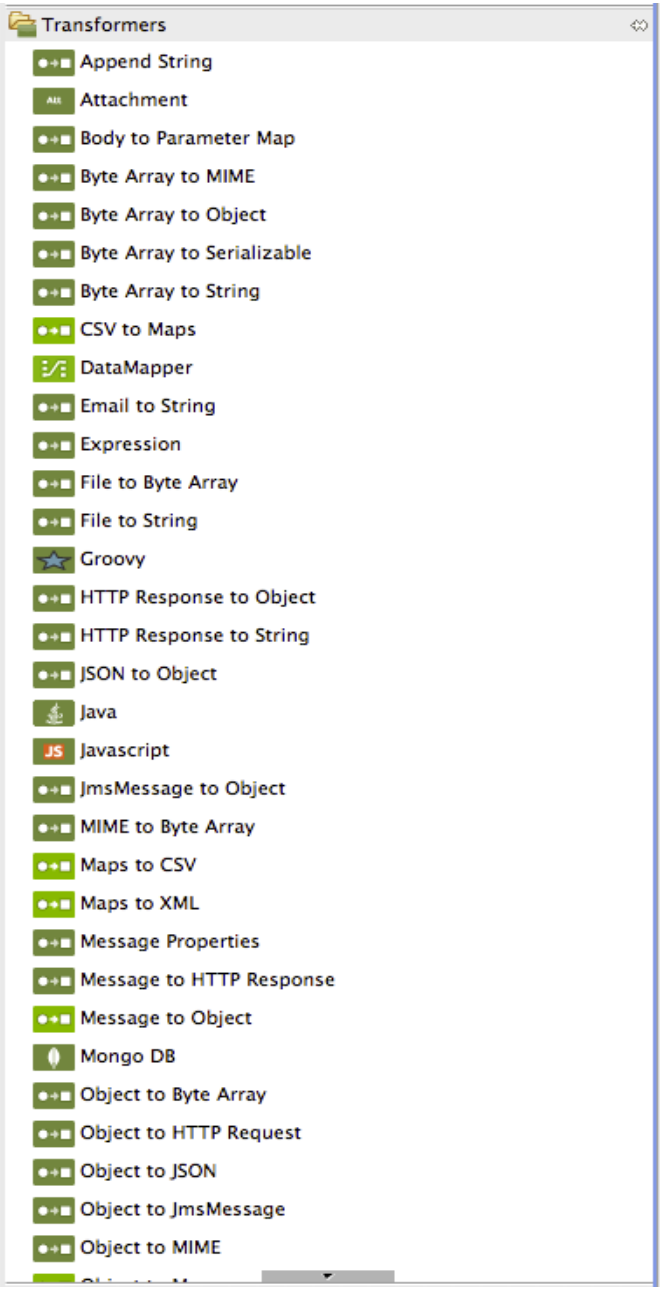
Components, transformers, filters

Somebody, someplace

probably wrote what you want







MEL – the mule expression language

- MEL provides the glue to access the message, payload or payload attributes, and decouples mule from the payloads passed.
- You can always write the glue bits using a jsr223 scripting language like ruby, python, groovy, clojure etc. But scripting languages on the JVM always come with the price of entering and leaving the scripting scope, MEL seems to be a bit faster. Does this actually matter? TEST (probably not btw)

Writing custom endpoints in java

- Yup, sure can. You can write transformers too!

But … why?

Why write custom endpoints, transformers, filters and so on?

- If a flow is sufficiently complicated or contains a number of similar member methods – **BUT(!)** when is a component in a flow not 'sufficiently' complicated?
- If a component in a flow requires security considerations - **BUT(!)** don't they all?
- If a component is or can be reused - **BUT(!)** can't all/most components be reused?
- If a component requires extensive unit testing or documentation external to the mule flow - **BUT(!)** but shouldn't every component have extensive automated testing?
- So … Always? Never? Depends?

Testing

Please? No really, Please!!

Loosely coupled systems lends itself exceedingly well to unit and integration testing.

Mule messages are easily constructed for unit tests.

Mule flows and sub-flows are easily instantiated and run independently of a mule server in maven projects.

#noexcuses just write the tests

<CODE>

//time to git our code on

git clone git@github.com:patarleth/mule_samples.git

git clone git@github.com:patarleth/hackathon.git

</CODE>

Links

- everything I've learned about mule is here -
  - https://github.com/patarleth/mule_samples/wiki
- Writing custom connectors
  - https://github.com/patarleth/mule_samples/wiki/Every-mule-link-i've-found-all-in-one-handy-spot
  - https://github.com/patarleth/mule_samples/wiki/Making-a-mule-connector-act-as-an-endpoint-and-emit-messages
  - http://www.mulesoft.org/documentation/display/current/Mule+DevKit
- If wrapping or porting an existing java class to mule give my code generation tool a shot
  - https://github.com/patarleth/devkit-assist/wiki