π^4

Global Models WS-CDL and Choreographies

School on Formal Methods 2009
Steve Ross-Talbot

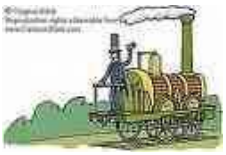
Agenda

- **Global Models 30 mins**
  - **Drivers 7**
  - **Requirements 3**
  - **Solution 2 - need to blend in some of the method slides ….**
    - **Descriptions**
    - **Methodologies**
      - **Inductive and Deductive**
- **WS-CDL 60 mins**
  - **As it is**
  - **As is should have been**
  - **Gaps in how to use it**
- **Uses 60 mins**
  - **Some examples**
- **Future 15 mins**
  - **Overlord**
  - **BPMN2**
  - **Scribble**

Cognizant
Passion for building stronger businesses

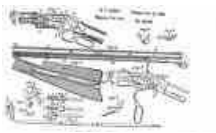
Global Models

The Industrial Revolution and Ambiguity



Stevenson's Rocket

Micrometer



Enfield Rifle

Steam Engine

The micrometer removed ambiguity between specification and implementation leading to both Stevenson's rocket and the off shoring of production of the Enfield rifle during the US Civil War.

IT and Ambiguity

- **Ambiguity[1]**
  - in requirements
  - between architecture and requirements[2]
  - between implementation and architecture

- **Ambiguity exists[3] because requirements are divorced from architecture and architecture from implementation, as a result we end up with:**
  - Poor alignment of IT to business
  - High cost in managing complexity
  - High cost of testing
  - Lack of transparency and control in delivery and change management
  - Poor reuse of IT assets
  - Lack of business agility hindered by IT

- **Removing ambiguity, joining things up, moves us from art to engineering[4]**
  - What we need is a specification language and a means of measuring implementation against this language to show correctness or deviation (our micrometer)

Leads to industrialisation of IT

Problem Domain: Requirements

- Functional
  - Based on some business goal in English
    - On-line sales channel connecting buyers to sellers
  - Typically a flow described as a use case in UML or a sequence diagram with annotations
    - It must be possible to examine any message and see where it has been
- Non-functional
  - Based on business and/or technical constraints
    - Seller response time should be less than 5 seconds on offering goods
    - Use existing standards with the organisation
    - Budget is $500,000 for initial PoC
  - Expressed in English
    - Seller goods.response.receive_time <= 5 seconds
    - Must use J2EE

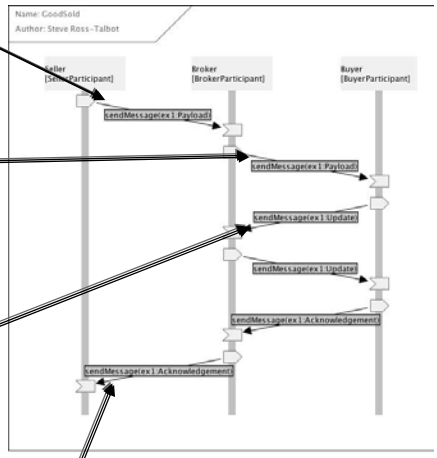
Problem Domain: Functional Requirements

```
<ex1:Payload xmlns:ex1="http://www.cognizant.com/broker">
    <ex1:id> 100 </ex1:id>
    <ex1:message> Widgets </ex1:message>
    <ex1:value> 105 </ex1:value>
    <ex1:reserve> 99 </ex1:reserve>
    <ex1:provenence> Receiver </ex1:provenence>
</ex1:Payload>
```

```
<ex1:Payload xmlns:ex1="http://www.cognizant.com/broker">
    <ex1:id> 100 </ex1:id>
    <ex1:message> Widgets </ex1:message>
    <ex1:value> 105 </ex1:value>
    <ex1:reserve> 99 </ex1:reserve>
    <ex1:provenence> Receiver, Broker </ex1:provenence>
</ex1:Payload>
```

```
<ex1:Update xmlns:ex1="http://www.cognizant.com/broker">
    <ex1:Payload>
        <ex1:id> 100 </ex1:id>
        <ex1:message> Widgets </ex1:message>
        <ex1:value> 102 </ex1:value>
        <ex1:reserve> 99 </ex1:reserve>
        <ex1:provenence> Receiver, Broker, Buyer </ex1:provenence>
    </ex1:Payload>
</ex1:Update>
```

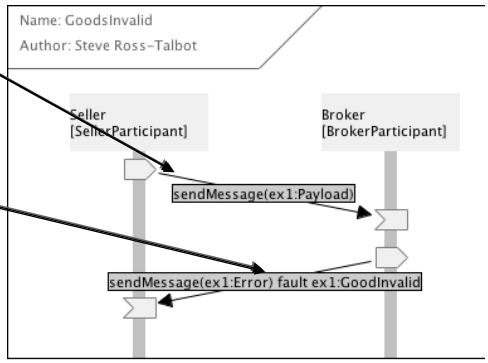
```
<ex1:Acknowledgement xmlns:ex1="http://www.cognizant.com/broker">
    <ex1:Payload>
        <ex1:id> 100 </ex1:id>
        <ex1:message> Widgets </ex1:message>
        <ex1:value> 99 </ex1:value>
        <ex1:reserve> 99 </ex1:reserve>
        <ex1:provenence> Receiver, Broker, Buyer, Broker </ex1:provenence>
    </ex1:Payload>
</ex1:Acknowledgement>
```



Problem Domain: Non-Functional Requirements

```
<ex1:Payload xmlns:ex1="http://www.cognizant.com/broker">
    <ex1:id> 100 </ex1:id>
    <ex1:message> Widgets </ex1:message>
    <ex1:value> 105 </ex1:value>
    <ex1:reserve> 99 </ex1:reserve>
    <ex1:provenence> Receiver </ex1:provenence>
</ex1:Payload>
```

```
<ex1:Error xmlns:ex1="http://www.cognizant.com/broker">
    <ex1:message> Unacceptable goods offered </ex1:message>
    <ex1:Payload xmlns:ex1="http://www.cognizant.com/broker">
        <ex1:id> 100 </ex1:id>
        <ex1:message> Widgets </ex1:message>
        <ex1:value> 105 </ex1:value>
        <ex1:reserve> 99 </ex1:reserve>
        <ex1:provenence> Receiver, Broker </ex1:provenence>
    </ex1:Payload>
</ex1:Error>
```

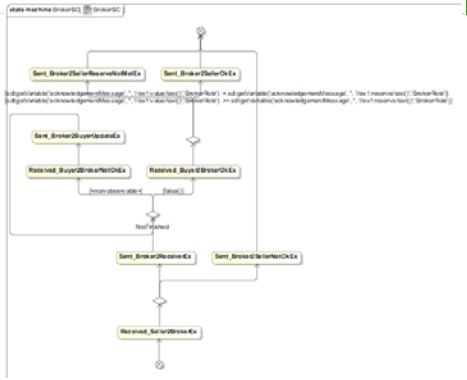


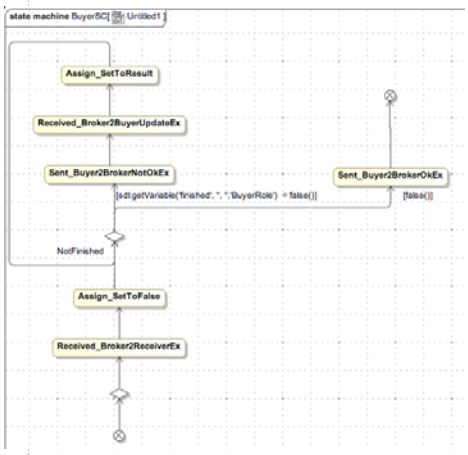
NFR 1: sendMessage(ex1:Error).receiveTime - sendMessage(ex1:payload).sentTime is less than 5 seconds

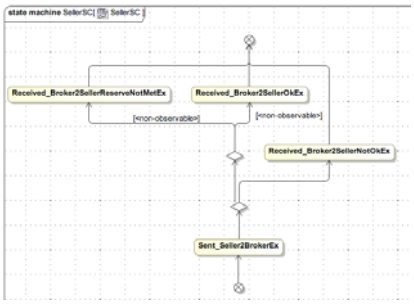
Problem Domain: Architecture

- TOGAF:

  An architecture[1] description is a formal description of an information system, organized in a way that supports reasoning about the structural properties of the system[2].







Problem Domain: Ambiguity

The reality is that

- we draw many pictures and write lots of text for the requirements and
- we draw many pictures and write lots of text for the architecture

- which are disjoint from each other and then disjoint from any implementation.

A lot of disjoint artefacts which breeds ambiguity

Observation

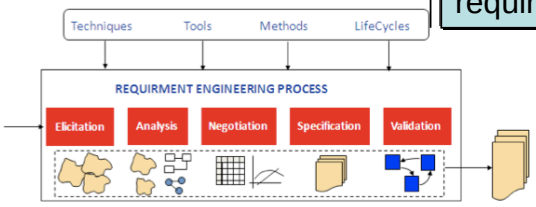
- How we come up with solutions today

1. Gather requirements (often poorly specified)
2. Model a solution (often at the wrong levels of abstraction and often not joined up)
3. Derive technical contracts to drive delivery
4. Implement
5. Unit testing remediation
6. Integration testing, remediation
7. User acceptance testing, remediation
8. Production

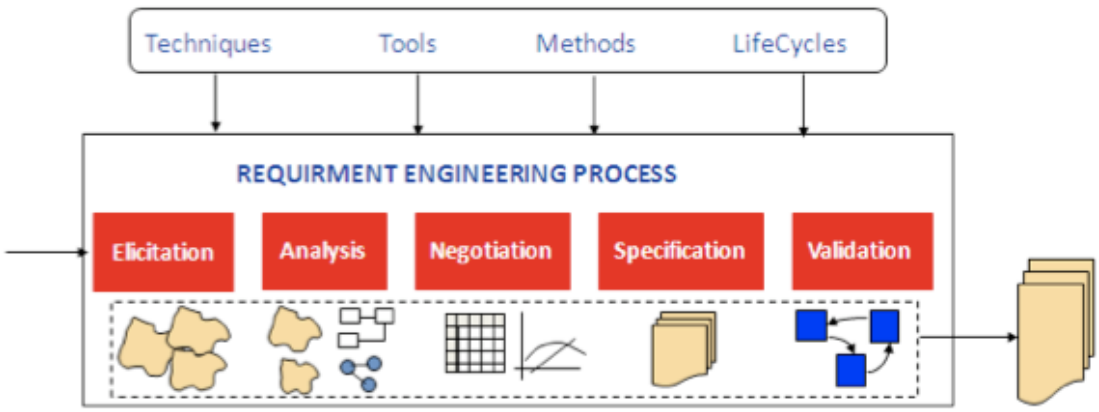
The model is deduced from the requirements

Technical contracts are induced from the model

Integration tests are induced from the requirements



Observation



The SOA promise

- Agility
  - I can change business rules to reflect business imperatives
  - I can build new services faster
- **Alignment**
  - I can see how a service implements a business function
  - I can track business transactions
- Reuse
  - I can leverage my existing assets
  - I can build new services faster
- De-coupling
  - I can change service implementation without impacting the system (but the contract must remain invariant)
  - I can evolve my IT landscape according to business need

The SOA fallacy: it is scale-invariant

- As we scale, complexity really hurts
  - As the number of service components increases so does the complexity of managing de-coupling, reuse, agility and alignment
  - As the number of service components increases change becomes more difficult and time consuming to manage
- **Imagine changing just one service out of 100**
  - Changing a service contract to have an additional error return type becomes a problem
  - How many services use this service?
  - How many services use the error return types?
  - How many composite services use this service in some other composition?
- This is why SOA Governance and Architecture Governance is so important

Requirements and Models

- Traditionally we gather requirements and refine them. At the top level (the business goals) we have requirements we might call R0. Successive refinement occurs through many levels (R1, R2, R3, R4 and R5). Where R5 tend to be requirements for executable code.

- Models are used to show how requirements can be satified. Models are expressed at the same level as the requirements that they satisfy. Thus we have a high level model that we might call L0 that satifies R0 and so on.

A solution must provide:

- A **global model**
  - an **unambiguous formal desciption of a set of components and their ordered interactions coupled with any constraints on their implementation and behavior**. Such a description may be reasoned over to ensure consistency and correctness against requirements.
- A **methodological approach**
  - to refinement for both requirements that a solution must meet and models of that solution
- A **means of testing**
  - models against requirements to ensure that no ambiguity exists

Observation

- How we could come up with solutions today
  1. Gather requirements (strongly specified)
  2. Model a solution[2] (at the correct levels of abstraction and joined up)
  3. Derive technical contracts to drive delivery
  4. Implement
  5. Unit testing remediation
  6. Integration testing, remediation
  7. User acceptance testing, remediation
  8. Production

Requirements are valdiated against the model

The model is induced[1] from the requirements

Technical contracts are induced from the model

Unit tests are induced from the requirements

Integration tests are induced from the requirements

Consider the sequence 1,2,3, X, …
What is X?

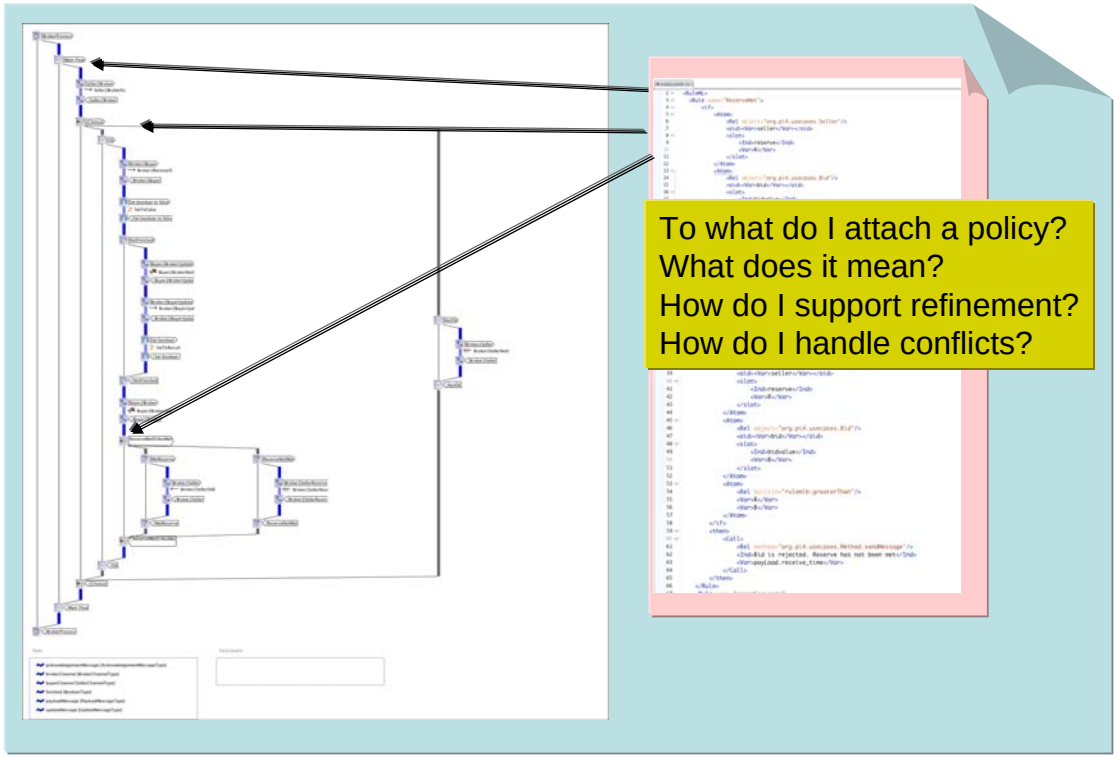
Requirements and Models

- Requirements are for humans.
- Requirements express some need and/or constraint on an outcome that we might think of as a solution.
- The level of a requirement and the semantics of that level are entirely to do with the level of abtraction that we wish to use in order to make the points that need to be made.
- Refinement moves us from one level to a deeper level,

Requirements and Models

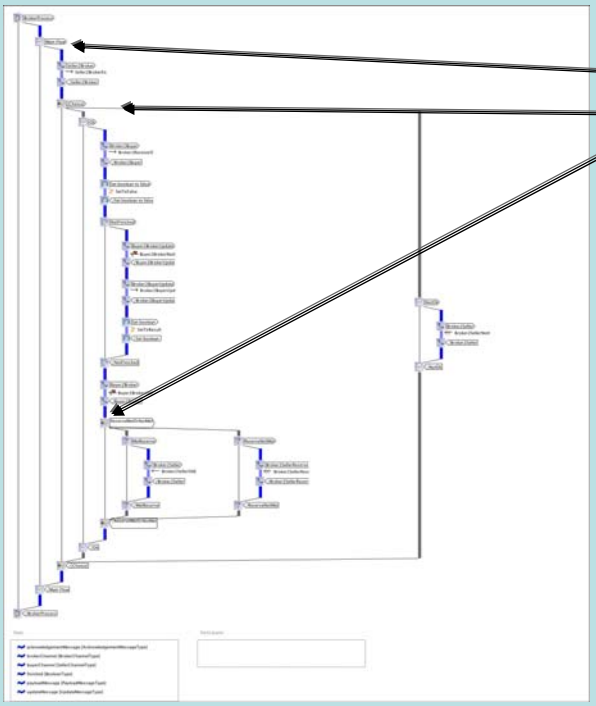
- Models are for humans.
- Models are used to create some representation of some domain.
- The level of a model and the semantics of that level are entirely to do with the level of abtraction that we wish to use in order to make the points that need to be made.
- Abstraction can be seen as a scoping operator over a domain in which some things are hidden that do nothing to make the points that need to be made.
- Models and their levels should be complete and unambiguous with respect to their level.
- A model at any level should be able to be type checked and checked for consistency so that it may be said to be correct against that level.
- Levels should support operators that enable a full or partial mapping from one level to another.

Some considerations



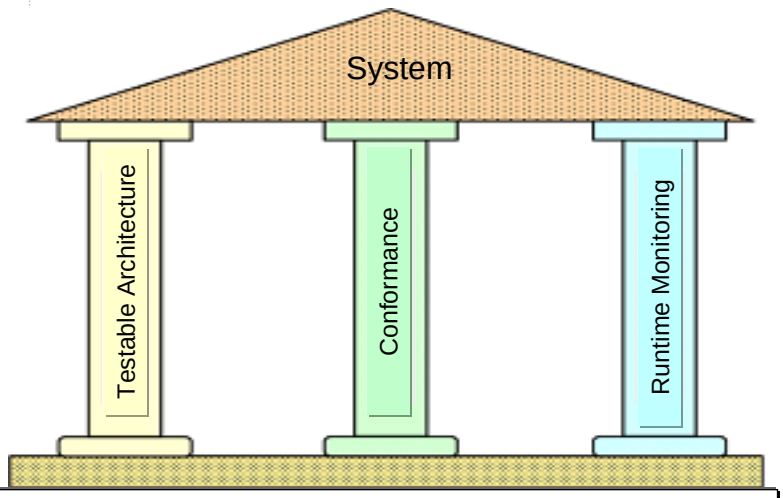
To what do I attach a policy?
What does it mean?
How do I support refinement?
How do I handle conflicts?

Some considerations



Where is the Information Model?
What is the relationship between in the Information Model and the Communication Model?

SOA: The three pillars to managing complexity



- •Ensuring architecture meets requirements before you cut code
- •Impact analysis to determine the effect of change against a precise architectural description.
- •Analysis to ensure that SLA's are achievable

- •Ensuring services meet their collaborative obligations (interoperability)

- •Ensuring services interoperate correctly
- •Ensuring business transaction meets their SLA's
- •Managing the business on an exceptions basis

The fundamental problem is that there is no A in SOA and so SOA solutions can easily be misaligned and hard to manage as they evolve.

What we need is **living architecture.**

A language for describing all functional reqs

- **Standard language to describe system architecture (WS-CDL)**
  - Unambiguous language for describing systems within an enterprise (a system description for an SOA) and/or the business process connecting enterprises (Invoices-Payments, Confirmations, etc)
  - Describes the ordered interactions between peered services, an interaction is the sending AND receiving of a message (i.e. a RequestToConfirm send from Party A to the DTCC) or the invoking of a method on a service and it's subsequent return (I.e. Party A invokes "getprice" on the Pricing service).
- **Amenable to the testing of a model against requirements**
- **Amenable to the generation of implementation artefacts**
- **Amenable to the monitoring of implementation against a model**

Solution

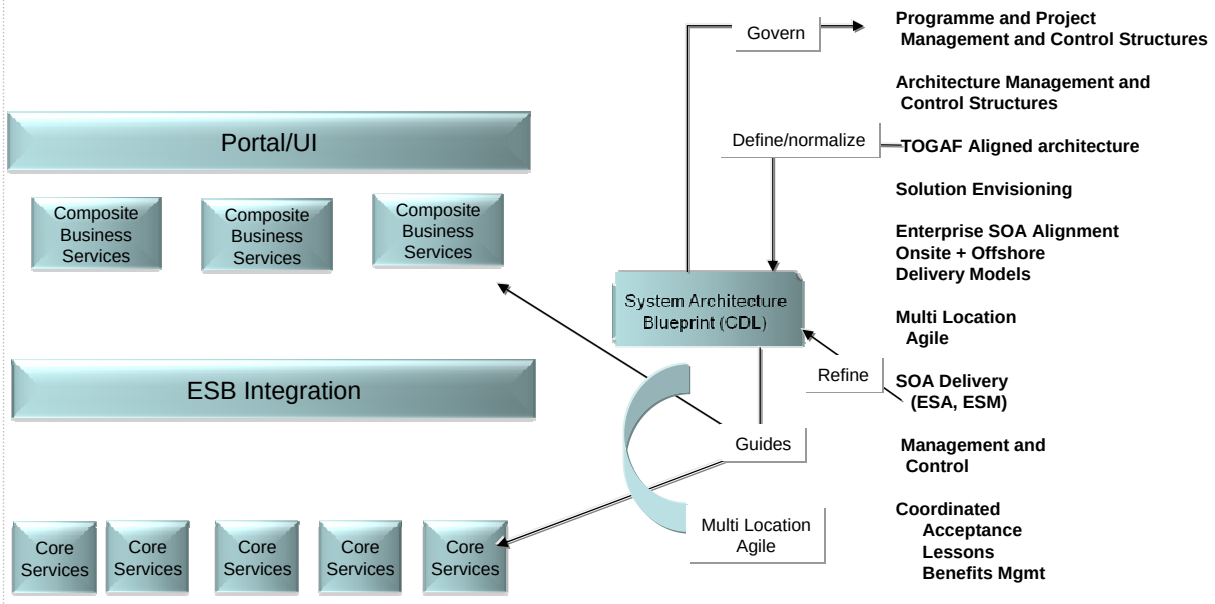
π^4

A language for describing all non-functional reqs

- **Fully Webized Open Standard for Rule Modeling, Classification, Serialization, Interoperation**
  - RuleML identifies expressive sublanguages for Web Rules (Derivation, Reaction, etc)
  - RuleML enables markup, translation, interchange, execution, publication, archiving in XML, RDF, OWL, UML, ASCII
- **Amenable to the testing of a model against requirements**
- **Amenable to the generation of implementation artefacts**
- **Amenable to the monitoring of implementation against a model**

π^4

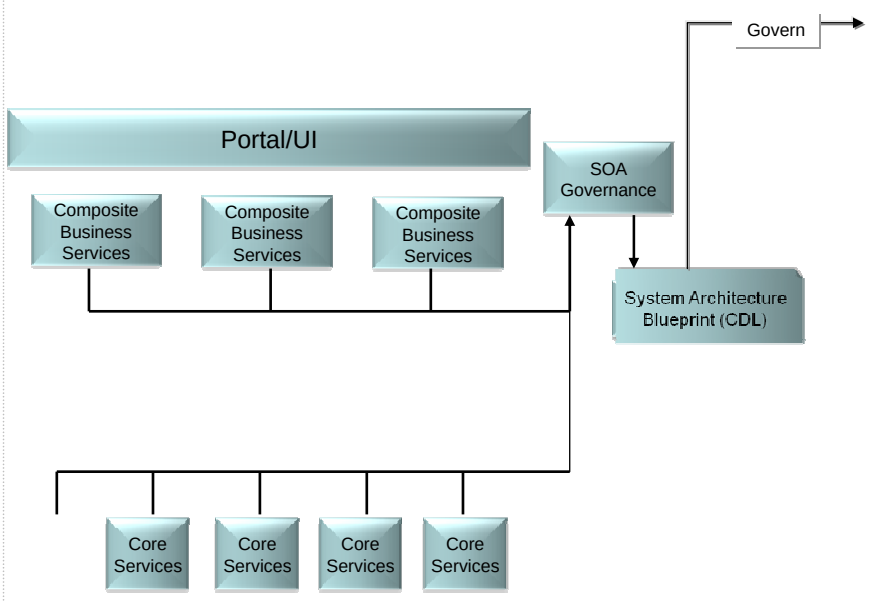
Model, Test, Implement



Solution

π^4

Govern



Programme and Project Management and Control Structures

Architecture Management and Control Structures

TOGAF Aligned architecture

Solution Envisioning

Enterprise SOA Alignment
Onsite + Offshore Delivery Models

Multi Location Agile

SOA Delivery (ESA, ESM)

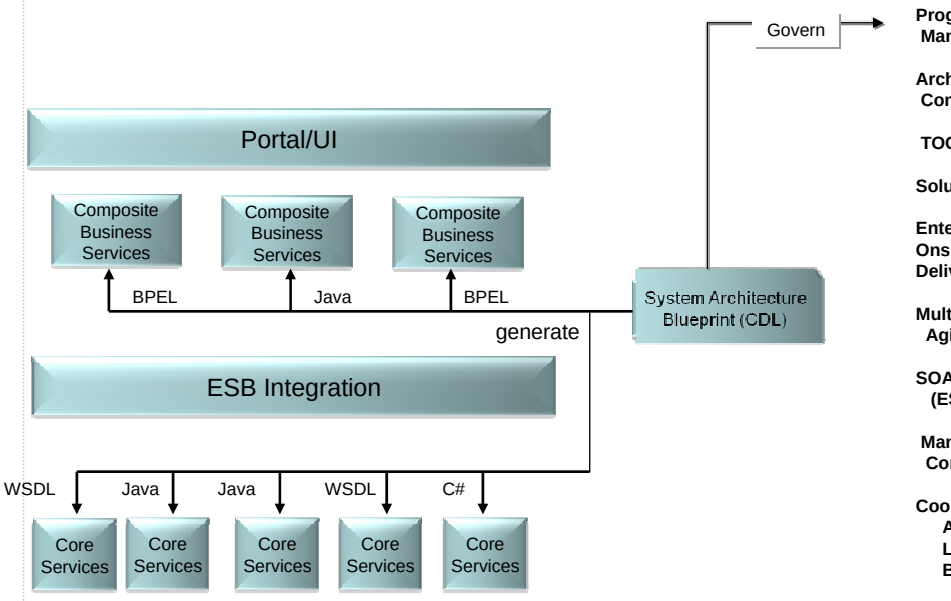
Management and Control

Coordinated
- **Acceptance**
- **Lessons**
- **Benefits Mgmt**

Solution

π^4

Generate



Programme and Project Management and Control Structures

Architecture Management and Control Structures

TOGAF Aligned architecture

Solution Envisioning

Enterprise SOA Alignment
Onsite + Offshore Delivery Models

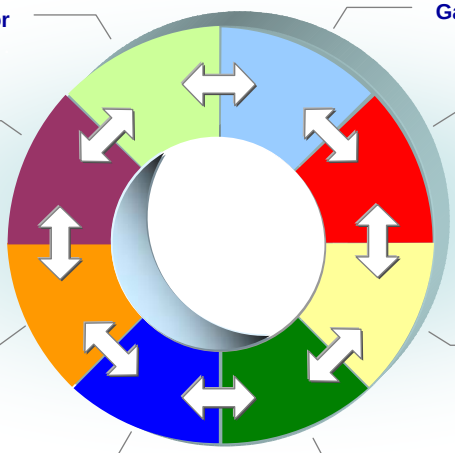
Multi Location Agile

SOA Delivery (ESA, ESM)

Management and Control

Coordinated
- **Acceptance**
- **Lessons**
- **Benefits Mgmt**

How to use it



Monitor
Runtime enforcement

Gather requirements
Sequence diagrams & messages

Test
J2EE, .NET against model

Model
System architecture of services

Implement
J2EE, .NET

Test
- the model against requirements

Guide Implementation
UML, WSDL, BPEL, HTML

Verify model
Sign off on description - BPMN, HTML

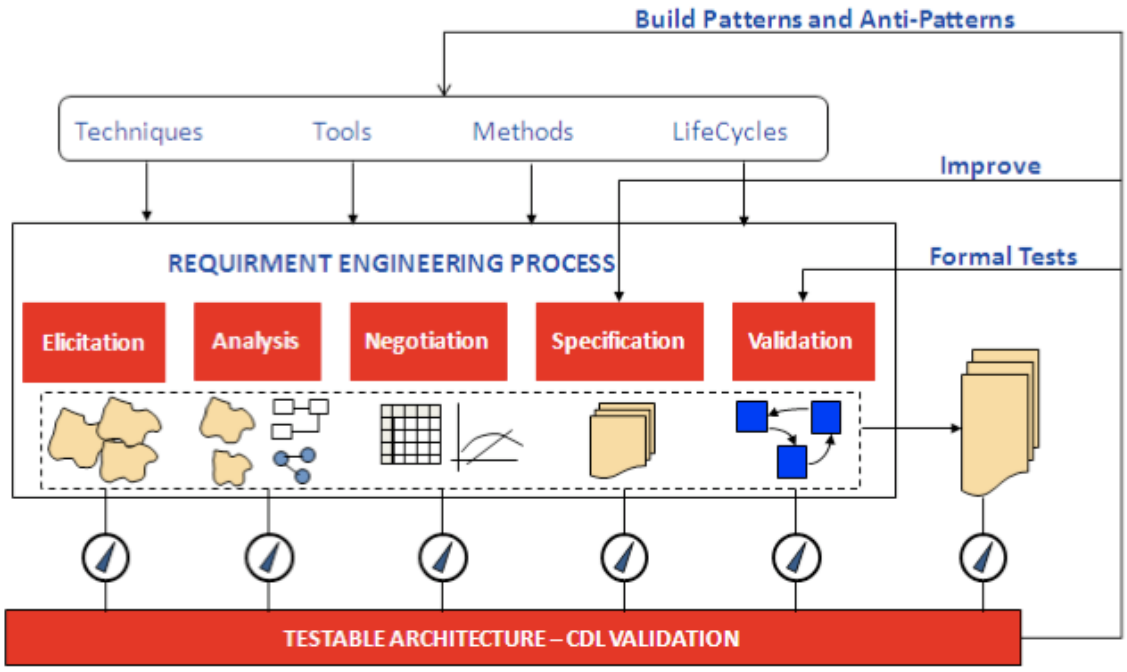
Removing <u>Ambiguity</u> means:

- ***Driving up quality***
- ***Driving down costs***
- ***Increasing agility in a controlled way***

Methodology

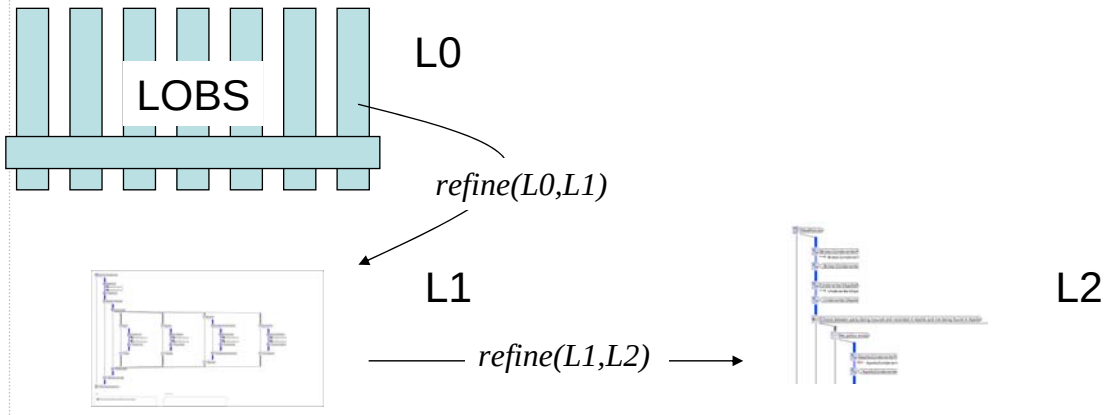
π^4

Improved Requirement Process to remove Ambiguity



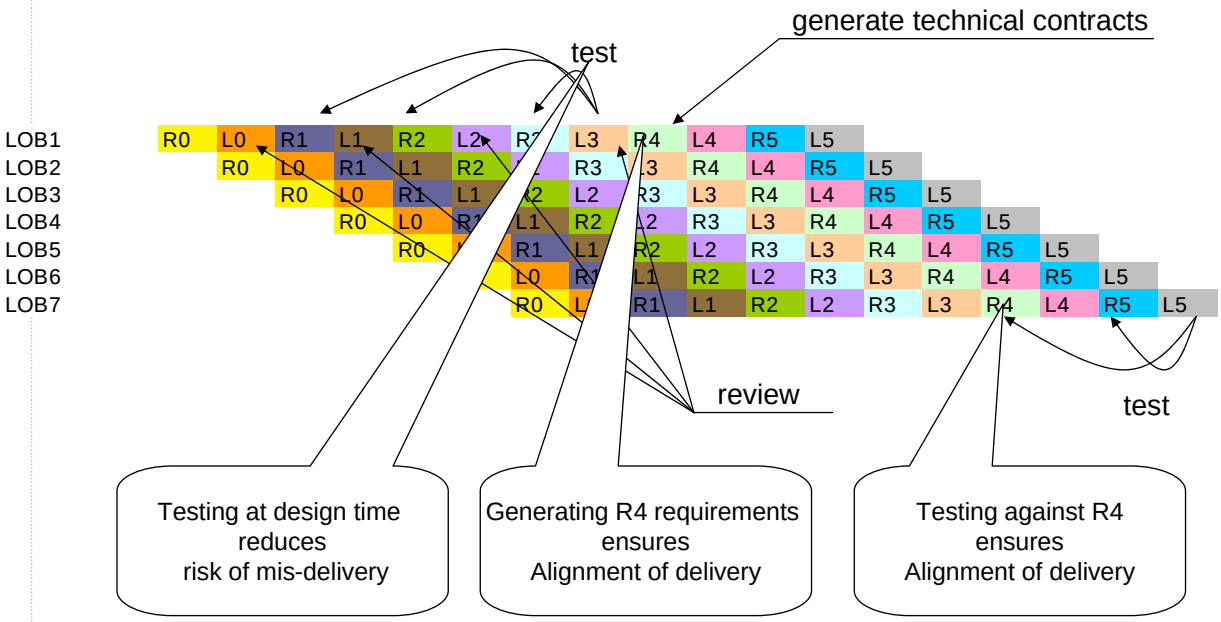
Requirements and Models

- If a set of requirements at R0 is said to be met by a model L0 and that model L0 is comprised of several parts (usually aligned to lines of business) then a phased approach can be adopted as follows:



π^4

Engagments



Models and Levels

- Having a precise definition of the semantics of a model and the operators that can be used to map from one layer to another
  - ensure that models are aligned
  - provide computable assurance that lower levels implement higher levels
- The net effect is lowering the risk of mis-delivery, increasing the quality of delivery and reducing the time to deliver

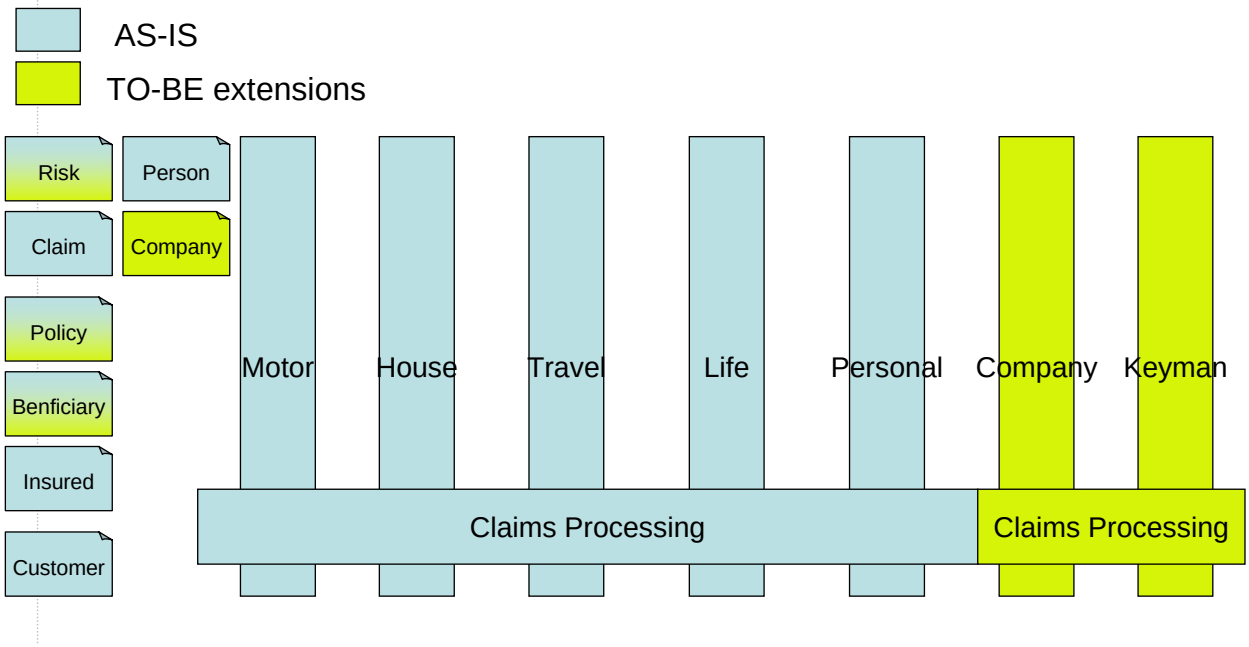
Levels

Model	Description
L0	The functional decomposition of an enterprise in terms of lines of business and business enities needed to enact those lines of business
L1	The lifecycle in terms of process names that are needed to drive a line of business in L0
L2	The observable communication model based on L1 that needs to be enacted over a set of AS-IS and TO-BE components or application or services in order to deliver lifecycle processes plus a logical data model for the business entities
L3	The observable bound communication model (binding of physical data model to L2) that needs to be enacted over a set of AS-IS and TO-BE components or application or services in order to deliver lifecycle processes plus a physical data model for the business entities such that they are bound to the communication model.
L4	The end point projects of the participants in L3 (applications, services and components) as state machines either as generated code or as technical contracts that will drive the implementation of the state machines.
L5	The executable code for each participant in L4 including all non-observable behavior (business logics) needed to implement a runnable state machine.

Methodology

π^4

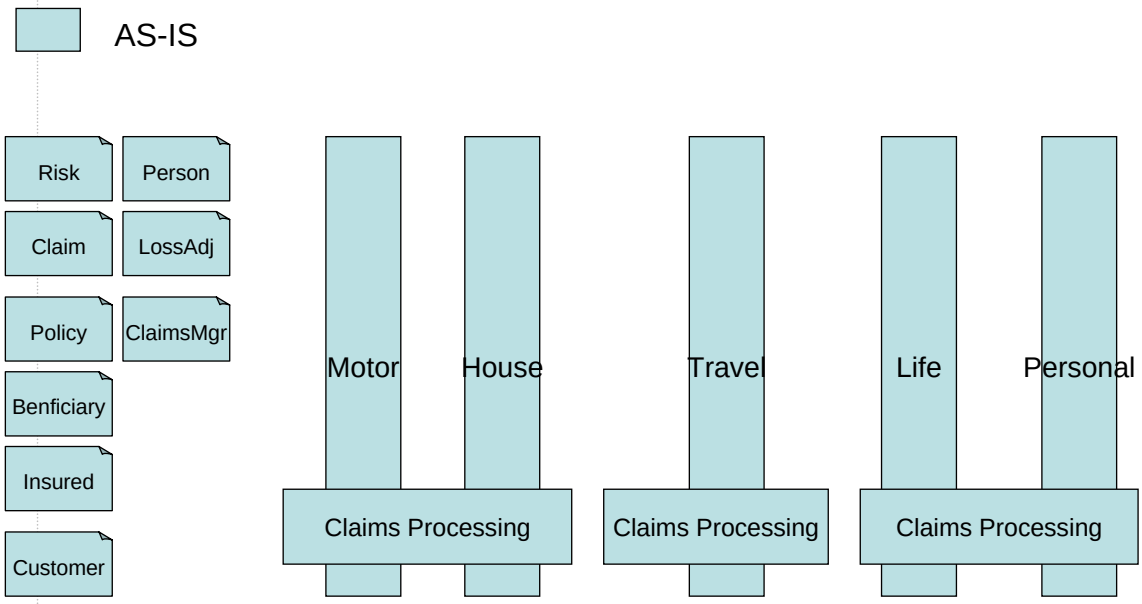
Level 0



Level 0 describes only the functional business decomposition of an enterprise in terms of high level areas of business and business/information entities

π^4

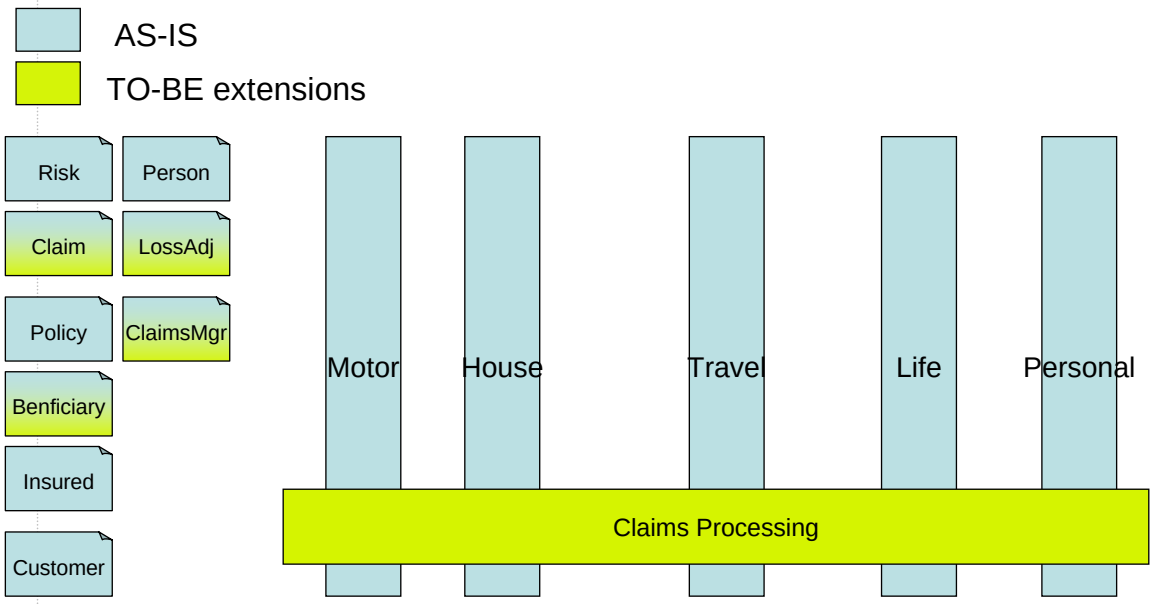
Level 0 AS-IS



Level 0 describes only the functional business decomposition of an enterprise in terms of high level areas of business and business/information entities

π^4

Level 0 TO-BE

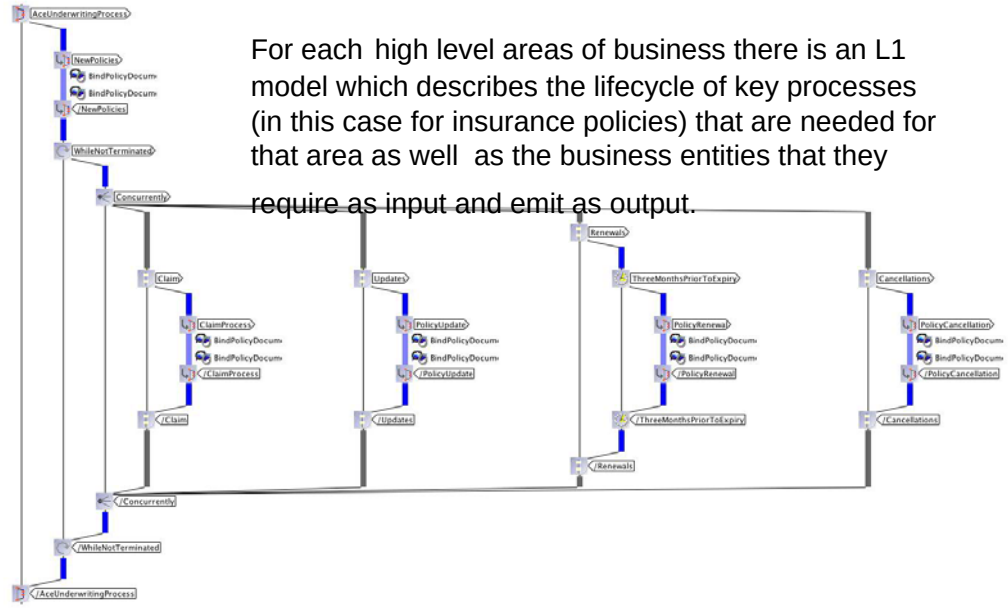


Level 0 describes only the functional business decomposition of an enterprise in terms of high level areas of business and business/information entities

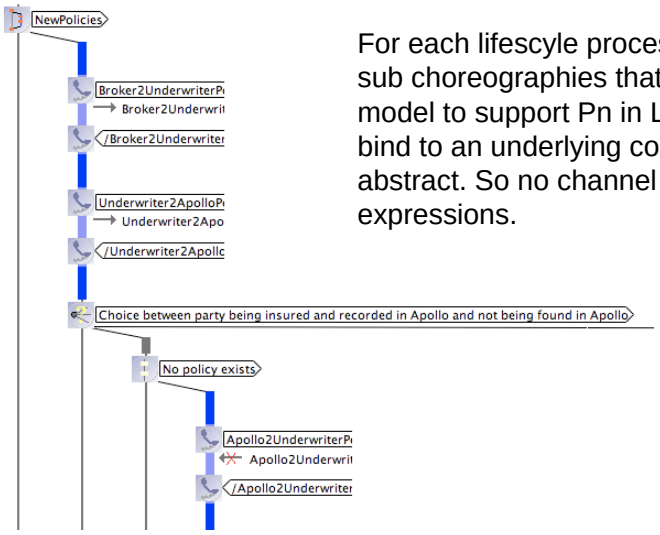
π^4

Level 1

For each high level areas of business there is an L1 model which describes the lifecycle of key processes (in this case for insurance policies) that are needed for that area as well as the business entities that they require as input and emit as output.

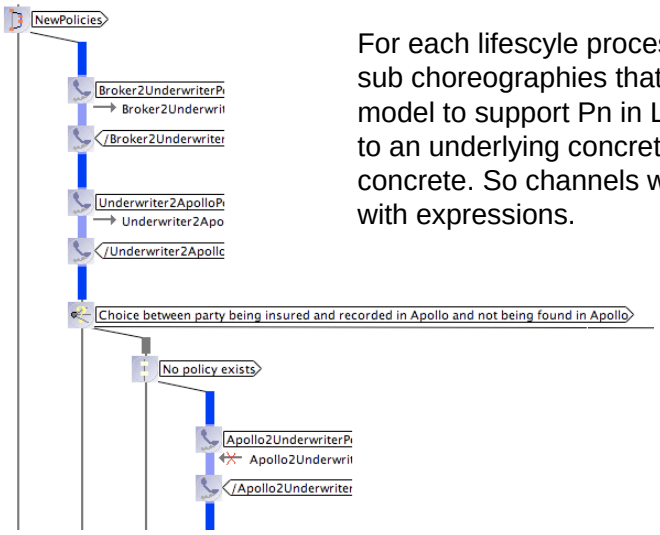


Level 2



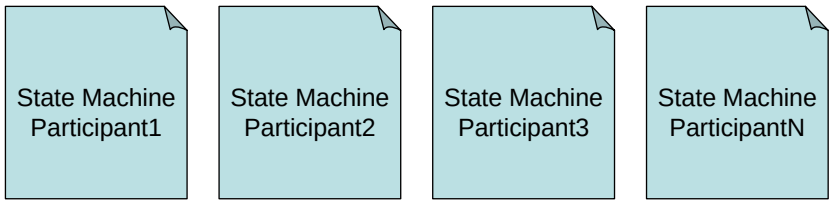
For each lifescyle process Pn in L1 there is one or more sub choreographies that describe the communication model to support Pn in L1. This model does not need to bind to an underlying concrete information model and is abstract. So no channel identities and no xpath expressions.

Level 3



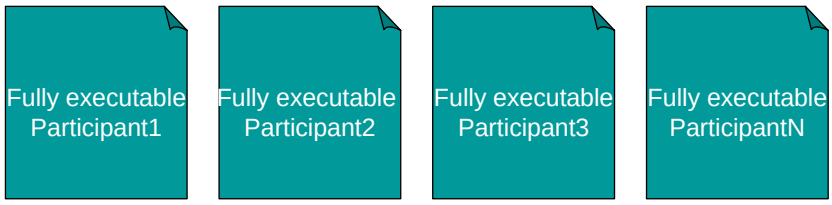
For each lifescyle process Pn in L1 there is one or more sub choreographies that describe the communication model to support Pn in L1. This model does need to bind to an underlying concrete information model and is concrete. So channels with identities and conditionals with expressions.

Level 4



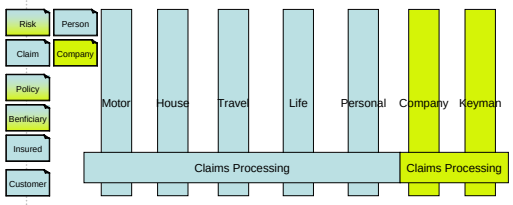
For each participant Pi in L3, using end-point-projection, there is a state machine that represents the observable behavior of that participant.
The state machine may be generated code or may be hand coded based on generated technical specificiations. In each case the behavior of the state machine is testable against an L3.

Level 5



For each State Machinein L4 there is a fully executable application or process that retains the state behavior that is observable in L3 but adds non-observable business logic to the state machine.

Levels



f(L0-Model,L1)
partial

f(L1-Model,L0)
full

f(L1-Model,L2)
partial

f(L2-Model,L1)
full

Levels

f(L2-Model,L3)

partial

f(L3-Model,L2)

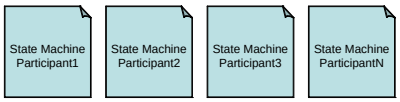
full

f(L3-Model,L4)

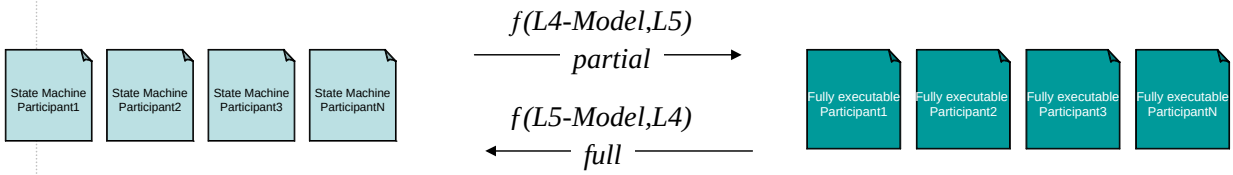
partial

f(L4-Model,L3)

full



Levels


State Machine Participant1
State Machine Participant2
State Machine Participant3
State Machine ParticipantN
f(L4-Model,L5)
partial
f(L5-Model,L4)
full
Fully executable Participant1
Fully executable Participant2
Fully executable Participant3
Fully executable ParticipantN

Models and requirements

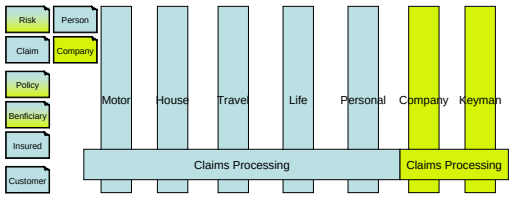
R0 Driven by business goals

We want to open up a new line of business for company insurance

We want to have a cross functional claims process

L0

Business architect



This is all about capturing the business goals in some high level picture that shows what exists and what needs to be added to meet the goals.

Models and requirements

Business/Enterprise architect

R1 Driven by high level requirements

There must be a policy in existiance prior to updating, renewing,canceling or querying a policy.

Updating, renewing, cancelling and querying can happen at any time

Updating, renewing, cancelling and querying can happen zero or more times until a policy is terminated

Renewals are 3 months prior to expiry

L0 is an input to L1

L1

This is all about taking the L0 and adding in high level requirements that start to structure and give order to high level processes that are needed to realize a business goal. This might also include cross cutting functional concerns such as accounting, status checks (queries), claims and so on.

Models and requirements

R2 Driven by low level requirements

All policies will have a policy reference that is a string

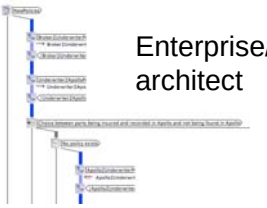
The portal process and business rules for initial qualification will be done in Pega.

A legacy system for policy management will be reused

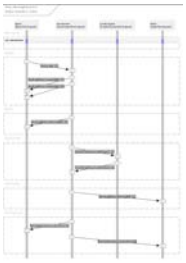
A set of concrete use cases and supporting sequence diagrams.

L1 is input to L2

L2



Enterprise/Solution architect



This is all about taking the L1 and adding in low level requirements that could be thought of as design and technology decisions. For example we might already know how we will reuse and augment the AS-IS landscape with the addition of mediators and mappers and other such services. The aim of L2 is to provide an abstract communication model with no firm binding to a concerete info model or technology.

Models and requirements

R3 Driven by low level requirements

Policy expires will be xsd:Date

Policy renewals will occur at expiry - 3 months

Policy termination will be when no referent policy is included in the latest status of the policy

Pega will use an XMLSchema called Pega.xsd

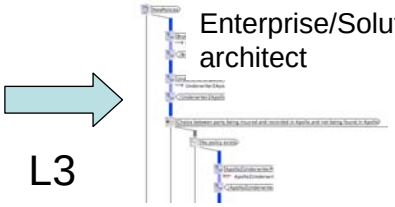
Legacy system will use a schema called Legacy.xsd

Mapping and mediation will be separate services able to run on a single machine or different machines

Use BPEL

Example messages and sequence diagrams

L2 is input to L3



Enterprise/Solution architect

L3

testable

This is all about taking the L2 model and binding it to a concrete information model which might be inferred or created based on the example messages. The channel identities for interactions need to be bound and any observable condition needs to be bound. The model properties may also be aligned to the delivery technology.

Models and requirements

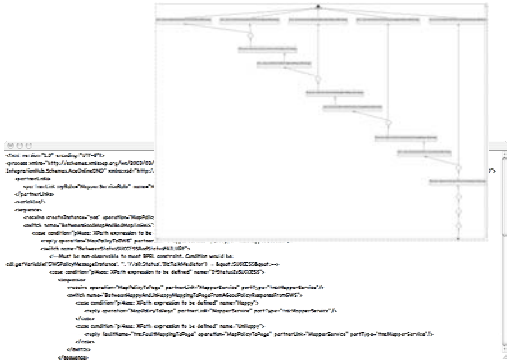
WSDL, BPEL, State Charts, etc

Enterprise/Solution architect

R4 Driven by and L3 contracts

Generated from an L3 model

L4



This is all about taking the L3 model and using it to generate the necessary technical contracts that specify the state behavior of the services, application and processes or indeed the code to implement that state behaviour. These contracts have everything except the internal business logic of the services.

Models and requirements

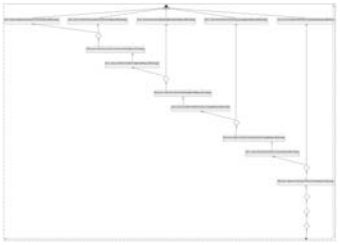
Executable services, applications or processes

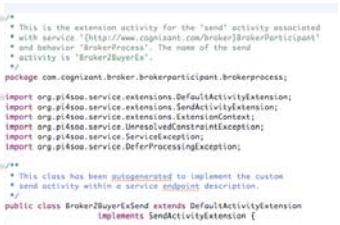
R5 Driven by and L3 contracts

Generated from an L3 model and augented by business logic from low level designs

L5

developer





This is all about taking the L4 technical contracts or indeed L4 generated state machines and adding the necessary business logic to each service, application or process.
In some cases the code needed is a thin veneer wrapping some legacy application in others it is the full business logic required. It all cases the state behavior is held invarient and by so doing ensures the delivered code meets its integration obligations.

Testing

- Testing uses a hypothesis and determines if that hypothesis holds against some experiment
- An experiment may be the running or simulation of running some computable artefact based on some known inputs and with a hypothesis as to what the outputs should be.

Testing, Models and Requirements

- Experiments are based on stimulating (running or simulating) some model at some level against some known inputs and expected outputs
- An input at a system level (across services) is the initial sending of a message in a sequence diagram.
- Inputs at the service level are the messages that each service receives over some known order.
- An output at a system level (across services) is the final receive of a message in a sequence diagram.
- Outputs at the service level are the messages that each service sends in response to some known input over some known order.

Testing, Models and Requirements

- Testing can occur at different levels
  - For any Model M at Level I we may test based on Requirements R described at that level.
  - We say that Mi satifies Ri when the testing results in an expected outcome.
  - We say that Mi does not satify Ri when the testing results in an unexpected outcome.
- Testing may be automated or manual

Levels

Level	Implemented	Testing operator
0		*satisfies(L0-Model,R0) returns true if L0-Model satifies R0*
1	*X*	*satisfies(L1-Model,R1) returns true if L1-Model satifies R1*
2	*X*	*satisfies(L2-Model,R2) returns true if L2-Model satifies R2*
3	*X*	*satisfies(L3-Model,R3) returns true if L3-Model satifies R3*
4	*X*	*satisfies(L4-Model,R4) returns true if L4-Model satifies R4*
5		*satisfies(L5-Model,R5) returns true if L5-Model satifies R5*

Levels

Model	Refinement operator
L0	*refine(L0-Model,L1) yields an incomplete L1 based on L0 which may be refined into an L1 but which is structurally bi-similar to L0.*
L1	*refine(L1-Model,L2) yields an incomplete L2 based on L1 which may be refined into an L2 but which is structurally bi-similar to L1.*
L2	*refine(L2-Model,L3) yields an incomplete L3 based on L2 which may be refined into an L3 but which is structurally bi-similar to L2.*
L3	*refine(L3-Model,L4) yields an incomplete L4 based on L3 which may be refined into an L4 but which is structurally bi-similar to L3.*
L4	*refine(L0-Model,L1) yields an incomplete L5 based on L4 which may be refined into an L5 but which is structurally bi-similar to L4.*
L5	

Levels

Model	Abstraction operator
L0	
L1	*abstract(L1-Model,L0) yields a complete L0 from an L1*
L2	*abstract(L2-Model,L1) yields a complete L1 from an L2*
L3	*abstract(L3-Model,L2) yields a complete L2 from an L3*
L4	*abstract(L4-Model,L3) yields a complete L3 from an L1*
L5	*abstract(L5-Model,L4) yields a complete L4 from an L1*

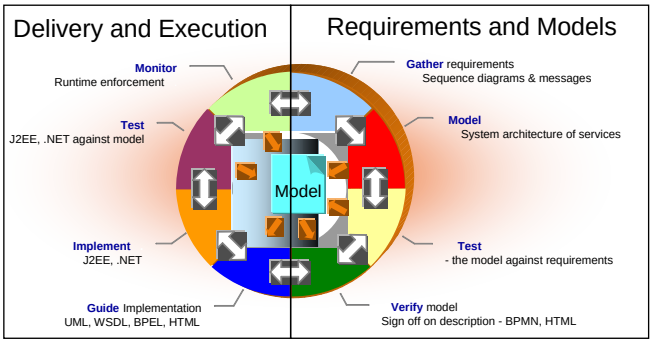
Levels

abstract(abstract(abstract(abstract(L5-Model,L4),L3),L2),L1),L0) gives an L0 from an L5 because it is generates or translates from lower to upper levels with no loss wrt to the semantics of the upper levels.

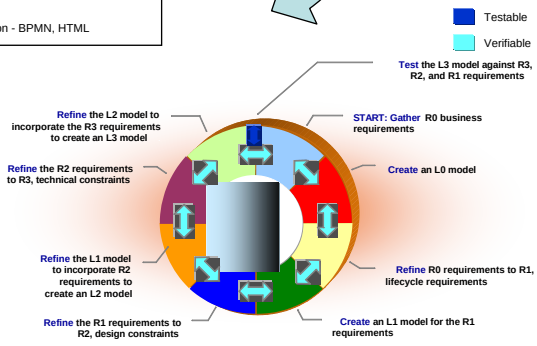
The same cannot be said of the refine operator because it only provides an incomplete translation to the lower levels because it does not have the necessary information.

For any models Mi and Mi+1, given that abstract(Mi+1,i) is the same as Mi, then if satisfy(Mi,Ri) means that satisfy(Mi+1,Ri). That is if we show that for some Mi that is valid against it's requirements Ri, and providing that Mi+1 is an implementation of Mi (abtract(Mi+1,i) = Mi), then Mi+1 also satifies Ri.

How to use it



Requirements gathering
Modeling
Model testing



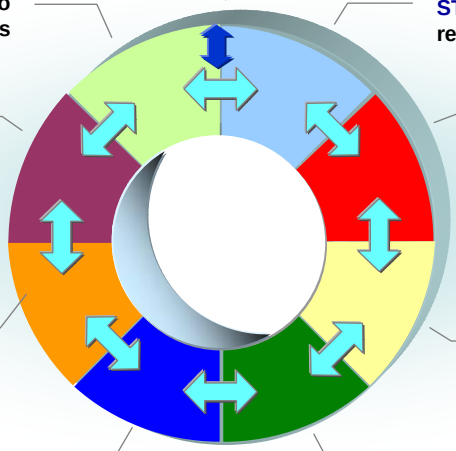
Successive refinement of requirements and models leading to streamlined delivery

π^4

Testable

Verifiable

How to use it



Test the L3 model against R3, R2, and R1 requirements

START: Gather R0 business requirements

Create an L0 model

Refine R0 requirements to R1, lifecycle requirements

Create an L1 model for the R1 requirements

Refine the R1 requirements to R2, design constraints

Refine the L1 model to incorporate R2 requirements to create an L2 model

Refine the R2 requirements to R3, technical constraints

Refine the L2 model to incorporate the R3 requirements to create an L3 model

Successive refinement of requirements and models leading to streamlined delivery

Cognizant
Passion for building stronger businesses

WS-CDL

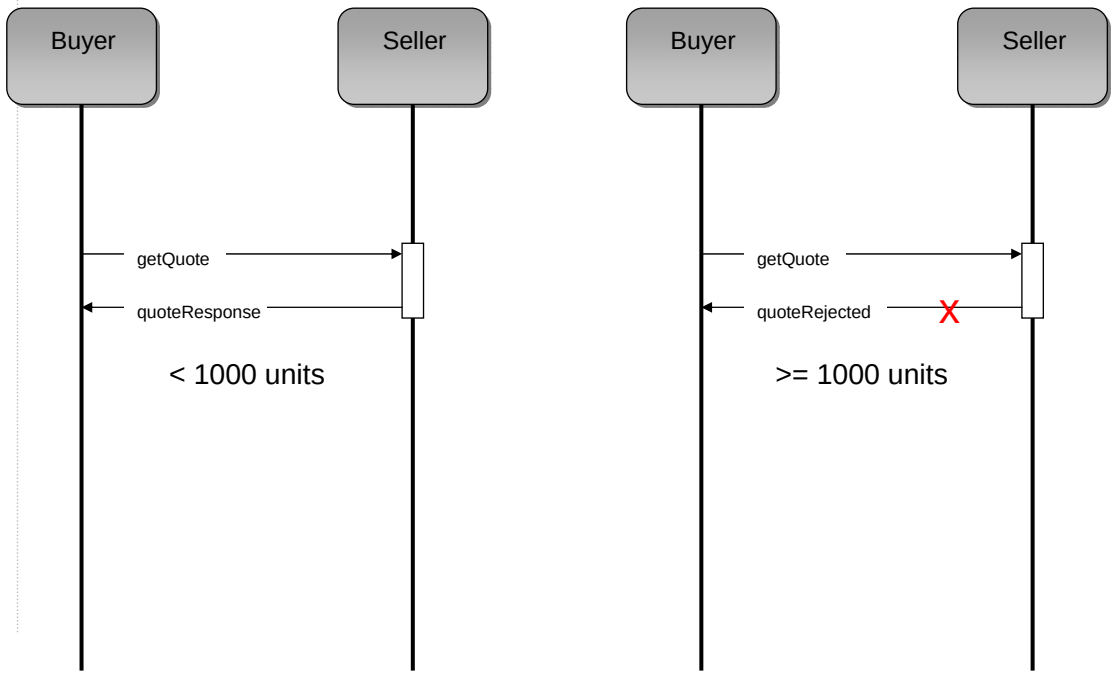
WS-CDL

A language to describe the **collaborative external observable behavior** of a set of **peered systems/services/applications** as an ordered set of **interactions over** one or more logical **channels** that connect them.

As it is

π^4

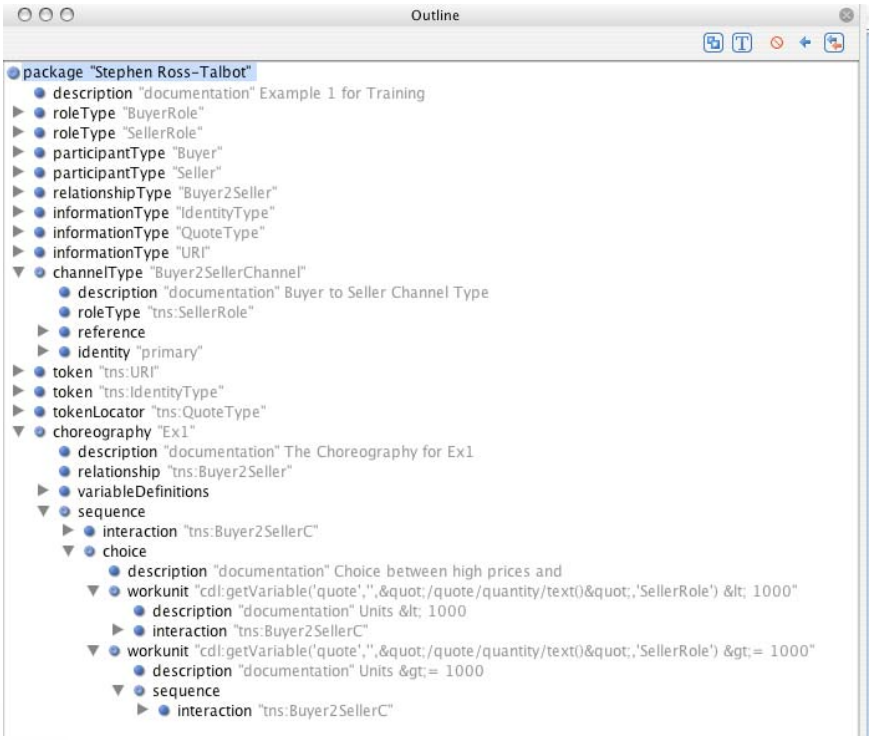
Simple example



As it is

π^4

Simple example

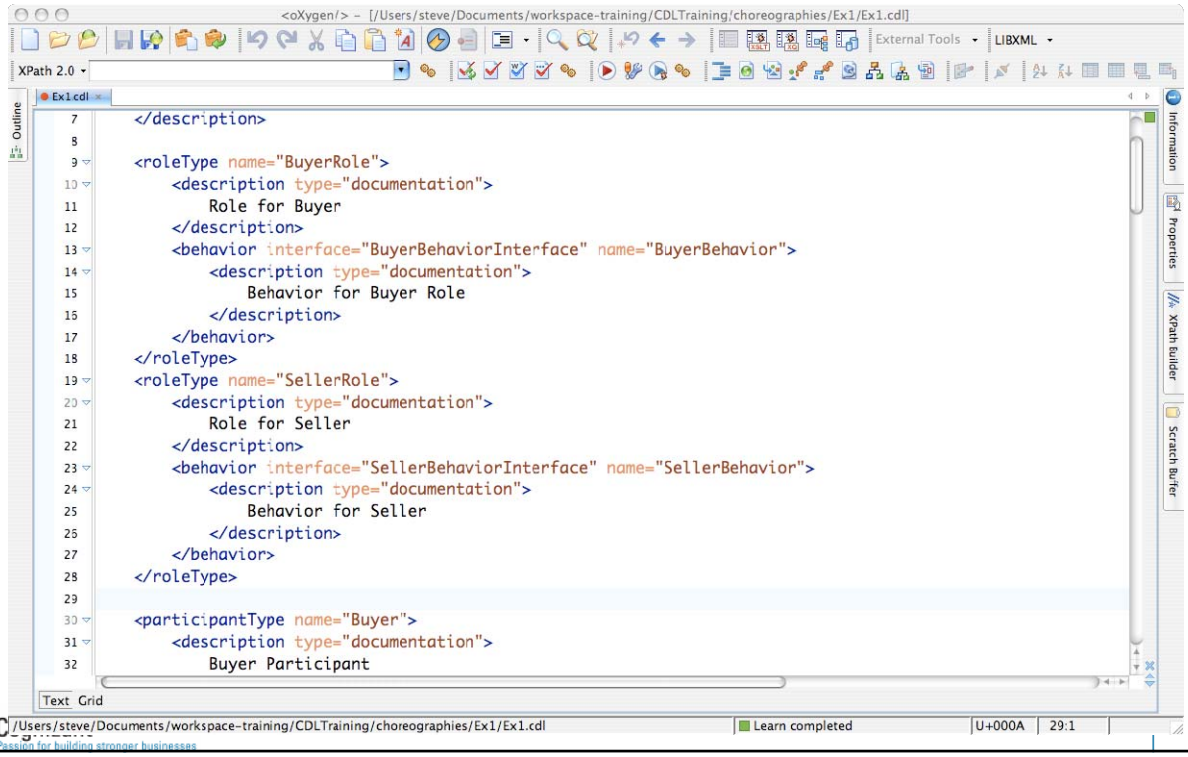


Cognizant
Passion for building stronger businesses

As it is

π^4

Roles

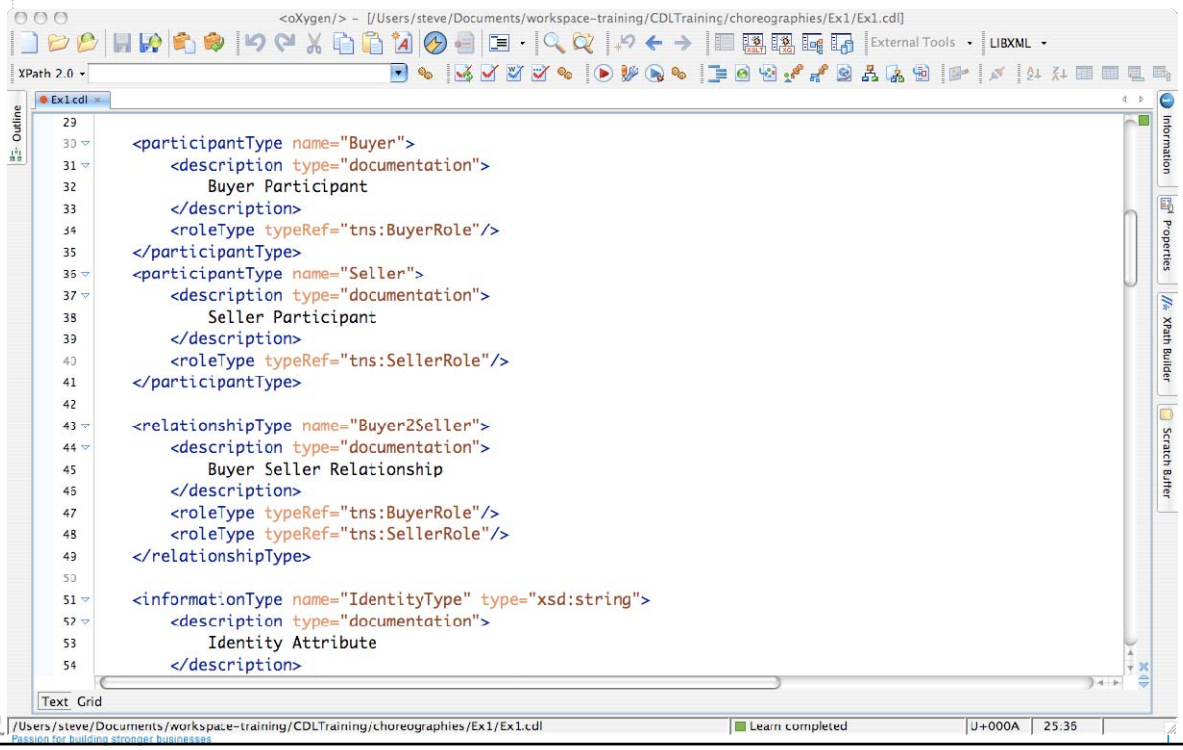


Passion for building stronger businesses

As it is

π^4

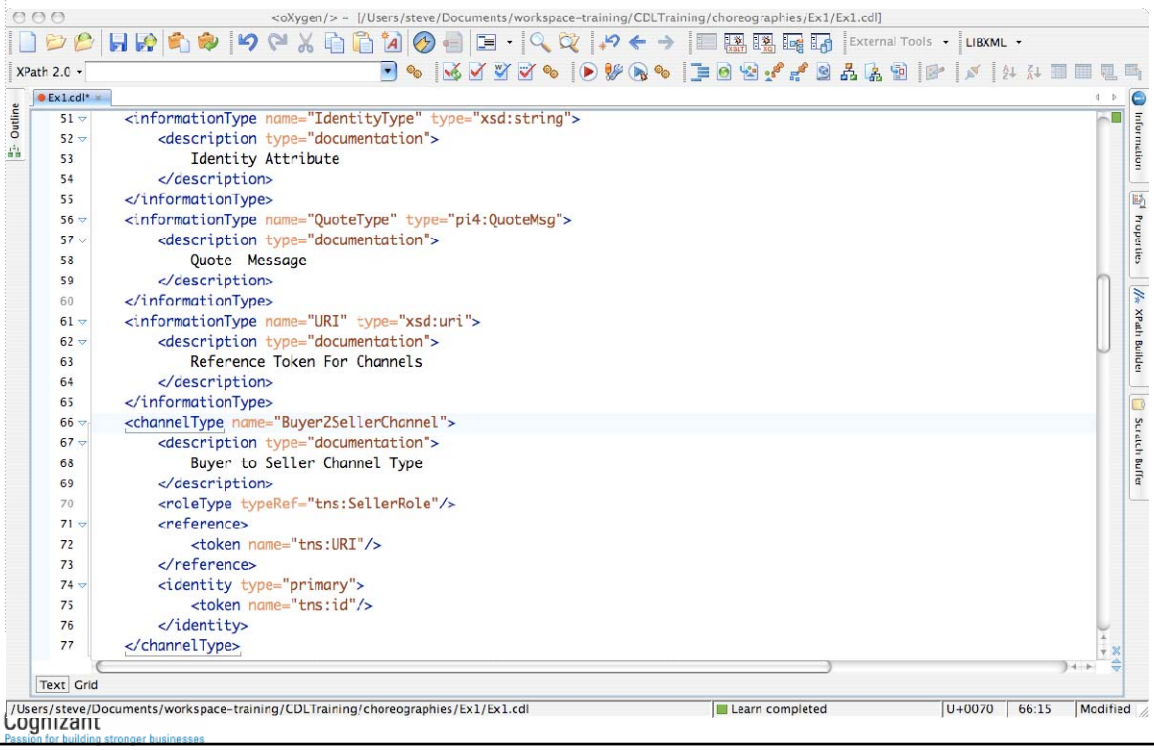
Relationships and Participants



As it is

π^4

Channels and Info types



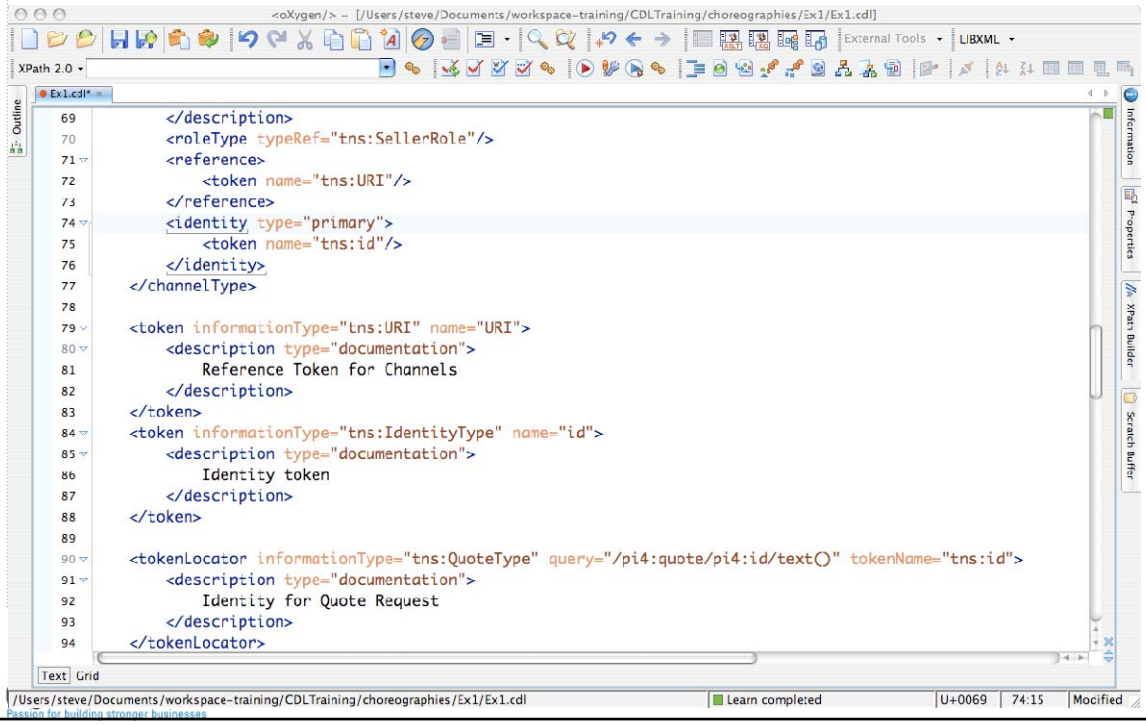
Cognizant

Passion for building stronger businesses

As it is

π^4

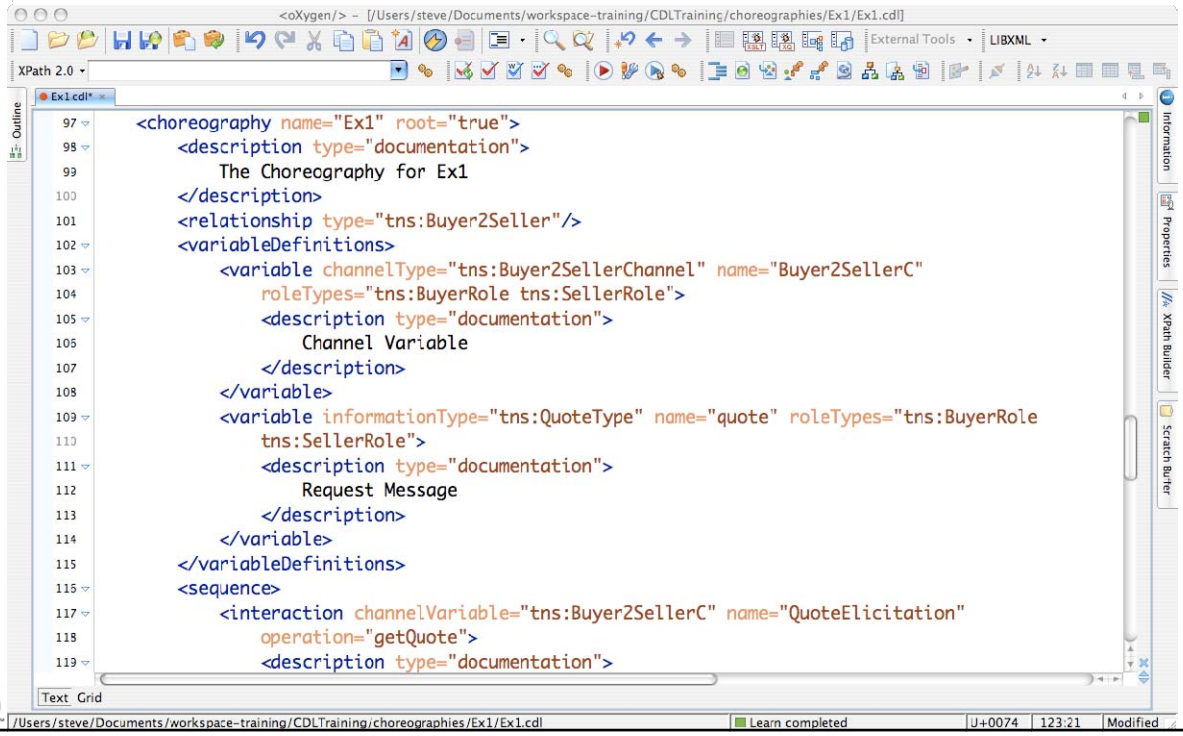
Bindings



As it is

π^4

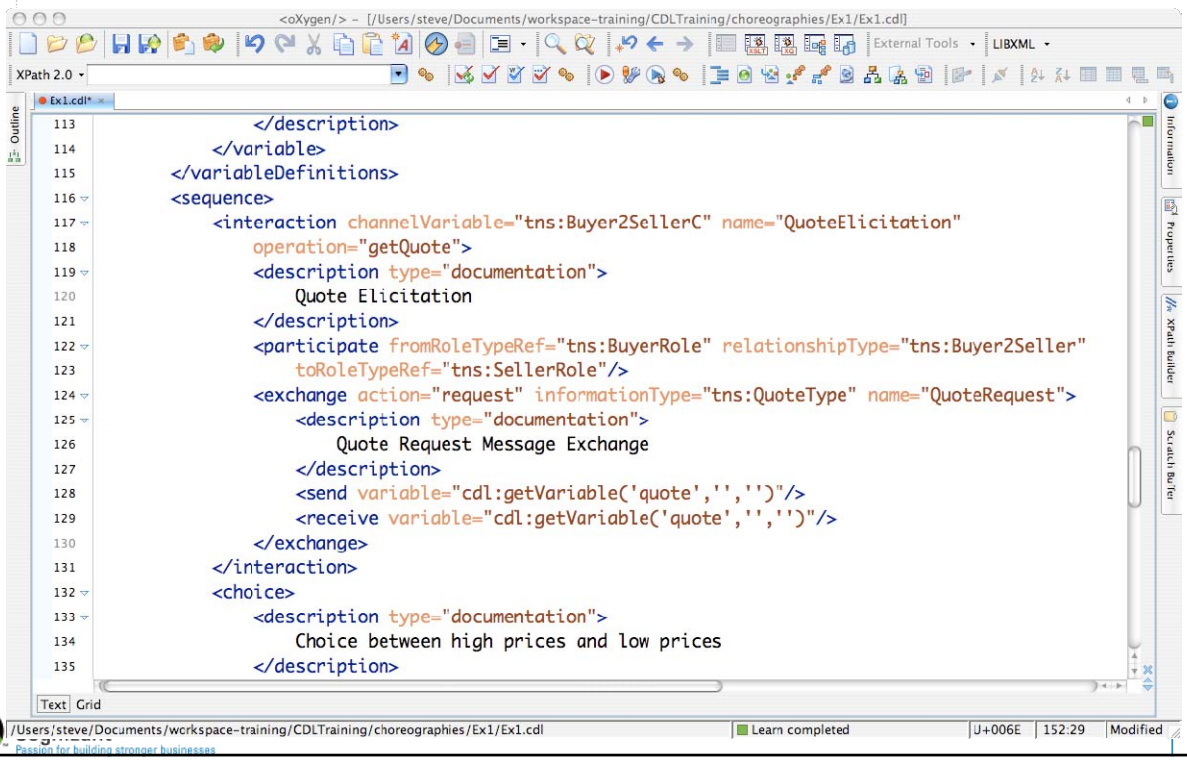
Variables



As it is

π^4

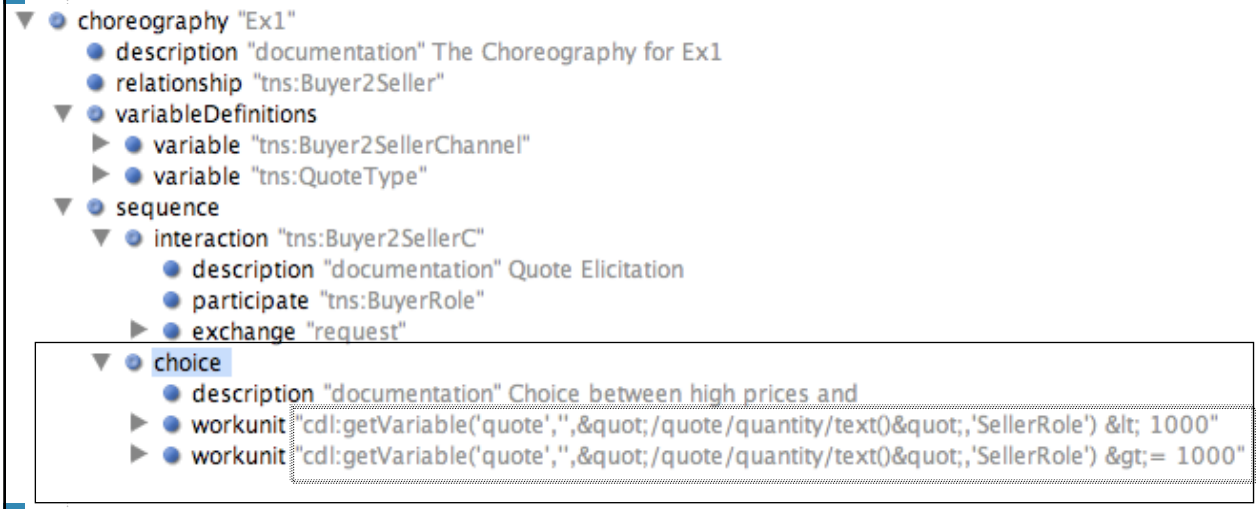
Interactions



As it is

π^4

Choice



As it is

π^4

Interactions

```
Interaction I1 over C operation O {
   exchange request R1 {
   }
   exchange response R1 {
   }
   exchange response { faultname F1
   }
   exchange response { faultname Fn
   }
}
```

Implied choice

Correlation is based on the channels and message content through token locators

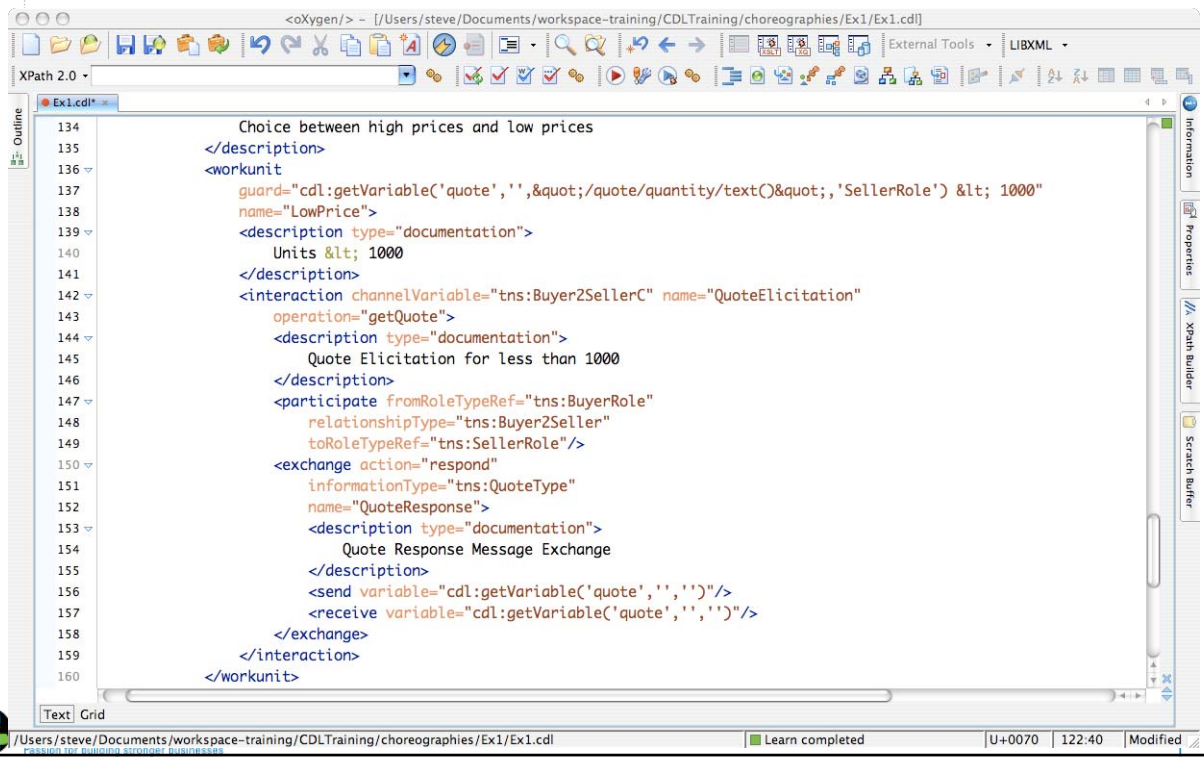
```
Interaction I1 over C operation O {
   exchange request R1 {
   }
Choice {
   Interaction I2 over C operation O {
      exchange response R1 {
      }
   }

   Interaction I3 over C operation O {
      exchange response { faultname F1
      }
   }

   Interaction I4 over C operation O {
      exchange response { faultname Fn
      }
   }
}
```

Explicit choice

As it is

π^4

Workunits



Workunits

Blocking

Workunit (G) (R) (B is True)
Body

Where G => guard condition, R => repeat condition, B => blocking attribute, Body => CDL activities within the work unit

A typical order of evaluation is as follows:
(G) Body (R G) Body (R G) Body

With respect to a G then the G is only evaluated when the variables are available and evaluate to True and otherwise we wait at the guard condition. Thus the Body after the first G only gets executed when G is True. Or put another way Body is primed ready for action and then is executed when G evaluates to True.

IF G is unavailable or evaluates to False THEN it equates to:

```
when (G) {
        Body
} until (!R)
```

IF G is always True THEN it equates to:

```
repeat {
        Body
} until (!R)
```

IF R is always False THEN it equates to:

```
when (G) {
        Body
}
```

Workunits

Non-blocking

Workunit (G) (R) (B is False)
Body

A typical order of evaluation is as follows:

(G) Body (R G) Body (R G) Body

Which equates to (in pseudo code):

```
while (G) {
        Body
} until (!R)
```

IF G is always True THEN it equates to:

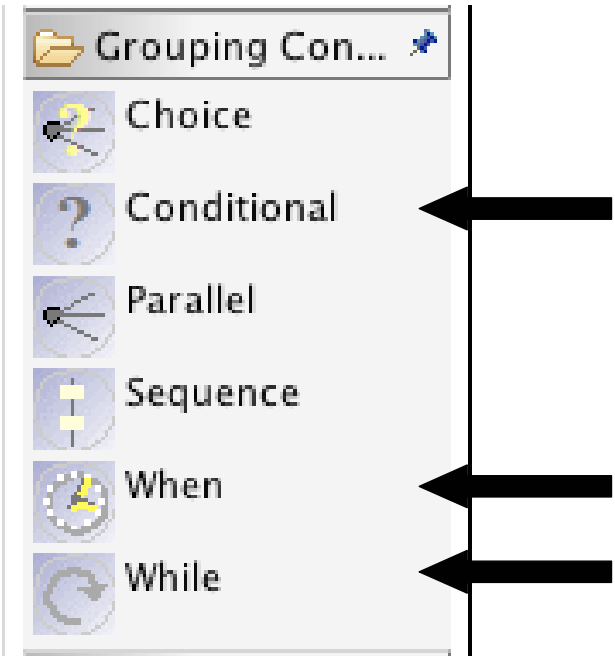
```
repeat {
        Body
} until (!R)
```

IF R is always False THEN it equates to:

```
if (G) {
        Body
}
```

This is what we used

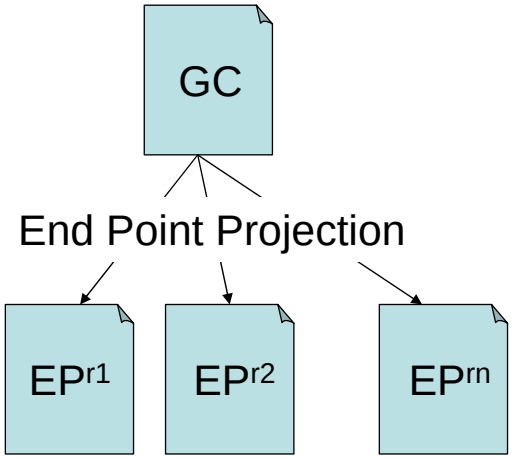
Workunits



This is what we provide in the Pi4Tech tools for workunits

Theory

- Global Calculus
  - Distillation of WS-CDL
  - Reduction rules, etc
- End point Calculus
  - Correspondance to Global Calculus
  - Reduction riles, etc



http://www.w3.org/2002/ws/chor/edcopies/theory/note.pdf

As it is

π^4

Theory

8.1. Formal Syntax. The formal syntax of the global calculus is given by the standard BNF. Below symbols $I, I', \ldots$ denote *terms* of the global calculus, also called *interactions*. Terms describe a course of information exchange among two ore more parties from a global viewpoint.

$$
\begin{array}{lll}
I ::= & A \to B : ch(\boldsymbol{\nu}\, \tilde{s})\,.\,I & \text{(init)} \\
| & A \to B : s\langle \mathsf{op},\ e,\ y \rangle\,.\,I & \text{(com)} \\
| & x@A := e\,.\,I & \text{(assign)} \\
| & \text{if } e@A \text{ then } I_1 \text{ else } I_2 & \text{(ifthenelse)} \\
| & I_1 + I_2 & \text{(sum)} \\
| & I_1 \mid I_2 & \text{(par)} \\
| & (\boldsymbol{\nu}\, s)\ I & \text{(new)} \\
| & X^A & \text{(recVar)} \\
| & \mathbf{rec}\ X^A\,.\,I & \text{(rec)} \\
| & \mathbf{0} & \text{(inaction)}
\end{array}
$$

http://www.w3.org/2002/ws/chor/edcopies/theory/note.pdf

As it is

π^4

Theory

11.1. Formal Syntax. The end-point calculus is an applied form of the π-calculus [**35**] augmented with the notion of participants and their local state (cf.[**3, 15, 21**]). Session initiation uses bound name passing, while in-session communication uses variables at a local store, in the spirit of [**15**]. The following grammar define *processes*, ranged over by $P, Q, R, \ldots$.

$$
\begin{array}{rcll}
P & ::= & !ch(\vec{s}).P & \text{(init-In)} \\
& | & ch(\nu\, \vec{s}).P & \text{(init-Out)} \\
& | & s \rhd \Sigma_i \mathsf{op}_i(x_i).P_i & \text{(input)} \\
& | & s \lhd \mathsf{op}\langle e \rangle P & \text{(output)} \\
& | & x := e.P & \text{(assignment)} \\
& | & \text{if } e \text{ then } P \text{ else } Q & \text{(conditional)} \\
& | & P \oplus Q & \text{(internal sum)} \\
& | & P | Q & \text{(parallel)} \\
& | & (\nu\, s)\, P & \text{(res)} \\
& | & X & \text{(variable)} \\
& | & \mathbf{rec}\, X.P & \text{(recursion)} \\
& | & \mathbf{0} & \text{(inaction)}
\end{array}
$$

http://www.w3.org/2002/ws/chor/edcopies/theory/note.pdf

WS-CDL Benefits

- A **complete and unambiguous description [*of a distributed system*]** of how peered roles or participants should interact and the order in which these interactions should take place.
- Enables **code generation** for a distrbuted system (end point projections) and guarantees interoperability of that system by design.
- Enables us to ask questions of a system so described:
  - Is it **free from deadlocks, livelocks and races**?

Two major gaps

1. Role are not shared between participants
   - Consider Wallmart. Wallmart is both a buyer and a seller of goods where “buyer” and “seller” may be roles. Consider a strawberry grower. They also play the role of “buyer” and “seller”.

     WallmartBuyer, WallmartSeller, StrawberryGrowerBuyer and StrawberryGrowerSeller.

     Participant Type: Market Participant (Buyer, Seller)
     Market Participant: Wallmart(Buyer)
     Market Participant: StrawberryGrower(Seller)
     Interaction BuyStrawberries (between Wallmart and StraberryGrower)

   - Bloating of descriptions because participants are not fully fledged types with instances.

   - Every interaction changes to include participant instances (actually it is the message exchanges) as an alternative to roles
   - Every CDL function changes to include participant instances as an alternative to roles
   - Every Assignment changes ….

Two major gaps

2. Message exchanges do not include any form of notification
   - Consider a scenario in which a buyer registers interest with an auction in some goods. How would be model notification of price changes?

When period is over
    Buyer request price from auctioneer

Buyer registers with auction
When price changes
    Auction notifies buyer of change

   - Models become poorly understood because of an inversion of the conceptual model needed to provide a solution.
   - Adding an additional message exchange type to request and response called notify

One major gap

1. Adding local behavior in-situ
   - CDL supports a slient action which can be situated at a role which represents non-observable behavior. It is a place holder and no more.
   - Today we cannot add any description to the global model for any local behavior that is non-observable.
   - We have to generate Java and then add to a generated state machine the business logic in code. So we mix description with code too early.

Cognizant
Passion for building stronger businesses

Examples

Example

π^4

Testable

Verifiable

Methodology of refinement

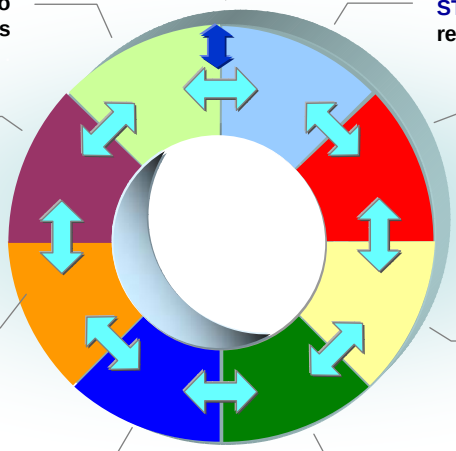
Test the L3 model against R3, R2, and R1 requirements

Refine the L2 model to incorporate the R3 requirements to create an L3 model

START: Gather R0 business requirements

Refine the R2 requirements to R3, technical constraints

Create an L0 model



Refine the L1 model to incorporate R2 requirements to create an L2 model

Refine R0 requirements to R1, lifecycle requirements

Refine the R1 requirements to R2, design constraints

Create an L1 model for the R1 requirements

Successive refinement of requirements and models leading to streamlined delivery

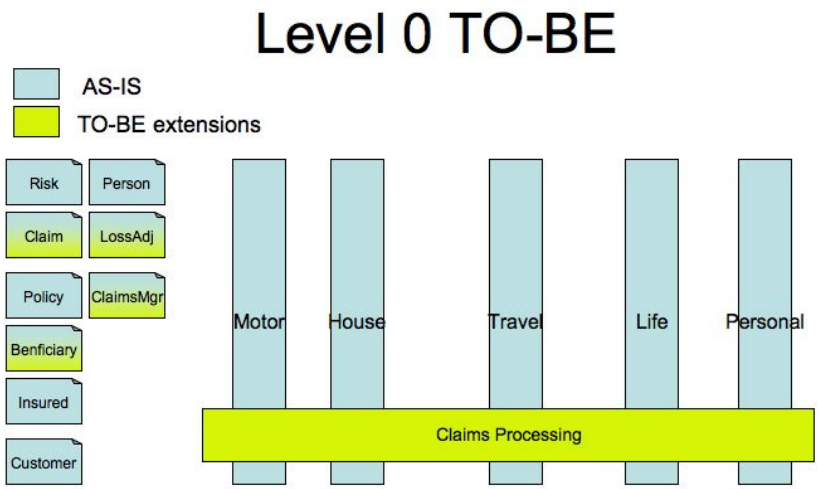
Example

π^4

L0 model and R0 requirements

R0

We want to have a cross functional claims process available online for both agents and insured parties.

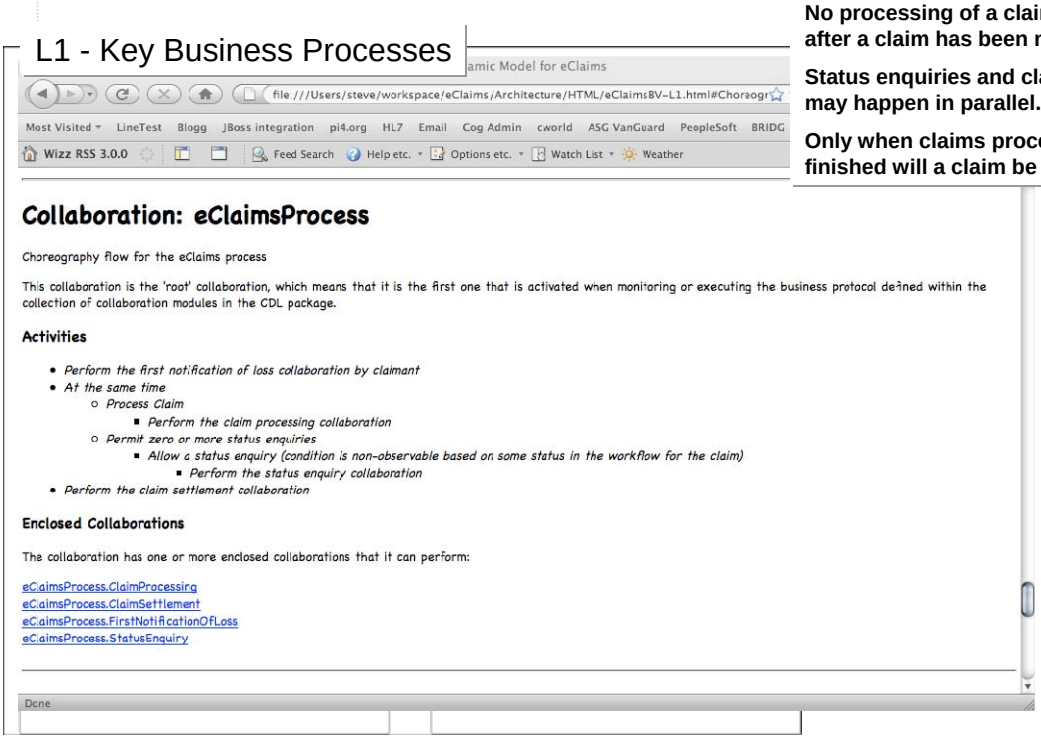


Example

π^4

L1 model and R1 requirements

L1 - Key Business Processes



R1

No processing of a claim can occur until after a claim has been notified.

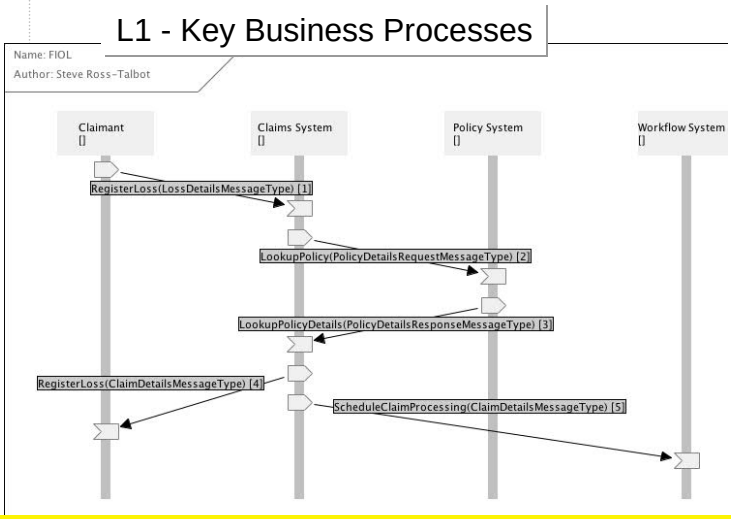
Status enquiries and claims processing may happen in parallel.

Only when claims processing has finished will a claim be settled.

Example

π^4

R2 requirements



R2

Policies will carry a Policy Reference

Claims will carry a Claim Reference

Loss Adjustors and claimants as well as agents will be provided with online access through a web channel.

Existing systems will be reused as services.

Existing services will be reused as is.

A set of concrete use cases and supporting sequence diagrams.

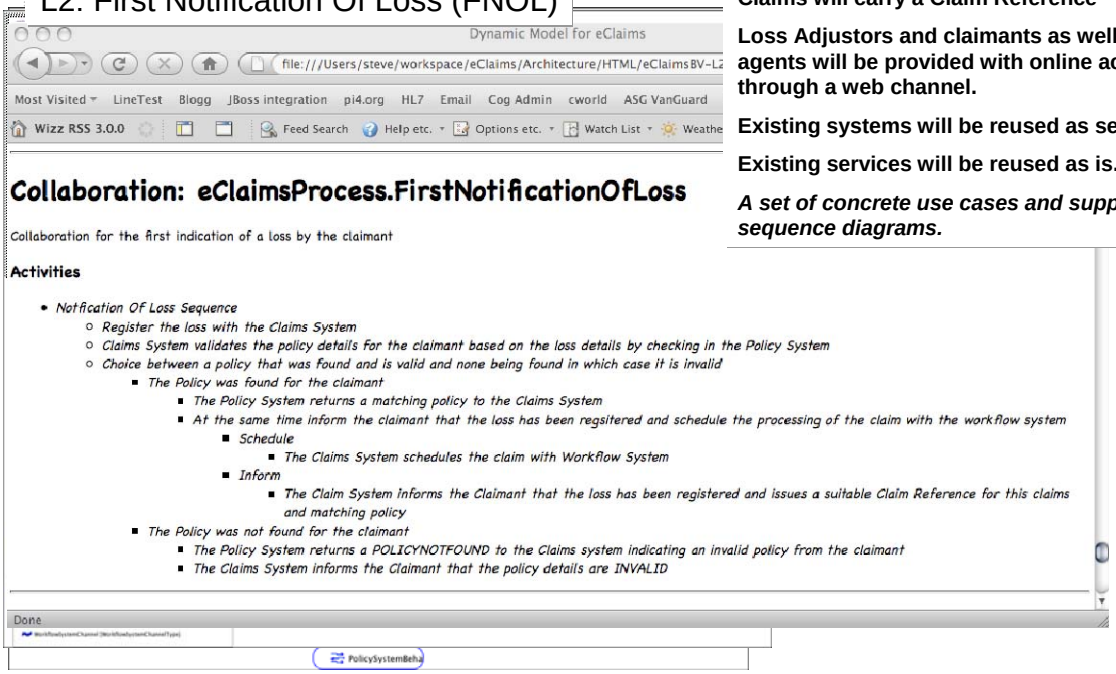
Ref	Identity Tokens	Identity Values	Query Expressions	Message File
[1]	null	null	null	LossDescriptionMessage.xml
[2]	null	null	null	PolicyDetailsRequestMessage.xml
[3]	null	null	null	PolicyDetailsMessage.xml
[4]	null	null	null	LossAcknowledgementMessage.xml
[5]	null	null	null	ScheduleClaimMessage.xml

Example

π^4

L2 Model and R2 requirements

L2: First Notification Of Loss (FNOL)



R2

Policies will carry a Policy Reference

Claims will carry a Claim Reference

Loss Adjustors and claimants as well as agents will be provided with online access through a web channel.

Existing systems will be reused as services.

Existing services will be reused as is.

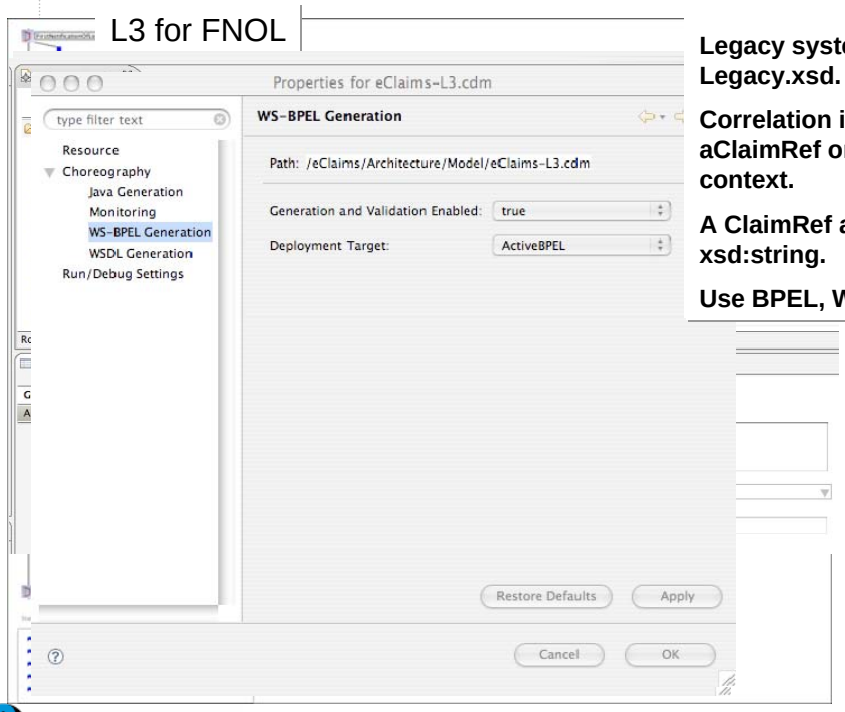
A set of concrete use cases and supporting sequence diagrams.

Example

π^4

R3 requirements

L3 for FNOL



R3

Legacy system will use a schema called Legacy.xsd.

Correlation identities are provided by aClaimRef or PolicyRef depending on context.

A ClaimRef and PolicyRef will be an xsd:string.

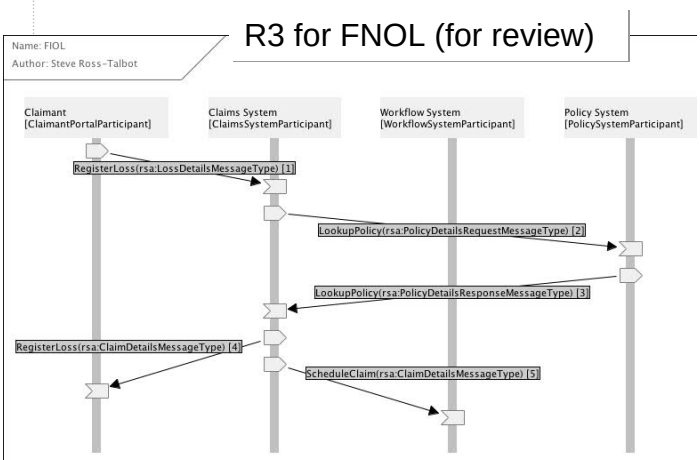
Use BPEL, WSDL1.1 and WebMethods

Example

π^4

R3 Requirements

R3 for FNOL (for review)

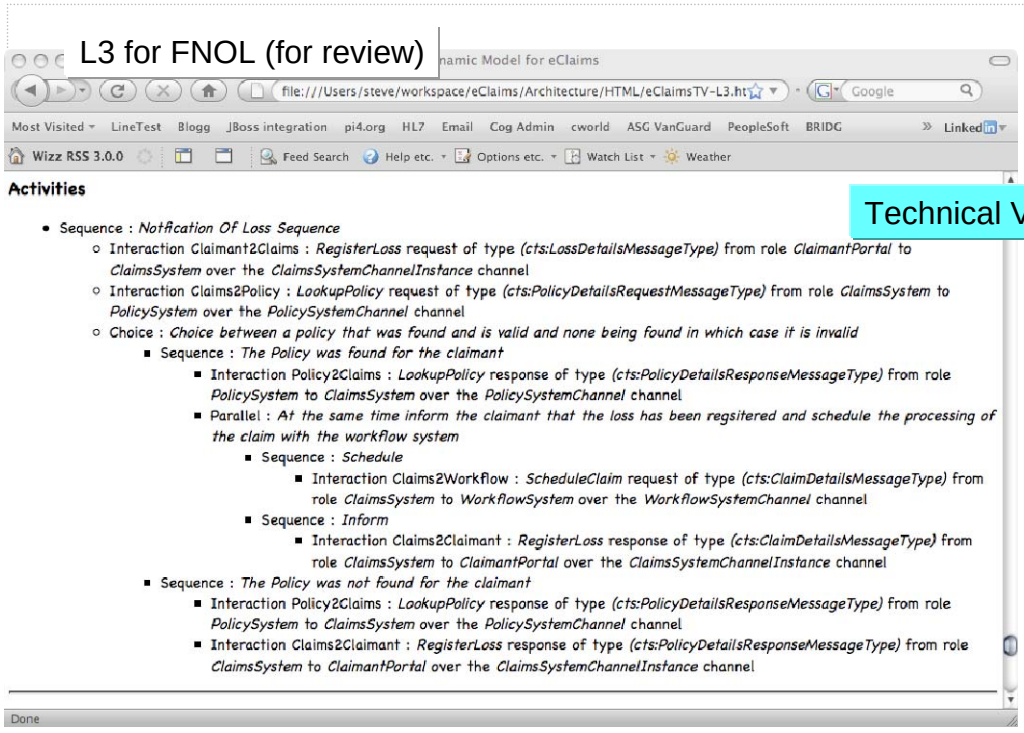


Ref	Identity Tokens	Identity Values	Query Expressions	Message File
[1]	PolicyRefToken	RSA-Motor-ABC123	//rsa:PolicyRef/text()	LossDescriptionMessage.xml
[2]	PolicyRefToken	RSA-Motor-ABC123	//rsa:PolicyRef/text()	PolicyDetailsRequestMessage.xml
[3]	PolicyRefToken	RSA-Motor-ABC123	//rsa:PolicyRef/text()	PolicyDetailsMessage.xml
		RSA-Motor-ABC123	//rsa:PolicyRef/text()	
[4]	PolicyRefToken ClaimRefToken	RSA-Motor-Claim-3	//rsa:ClaimRef/text()	LossAcknowledgementMessage.xml
		RSA-Motor-ABC123	//rsa:PolicyRef/text()	
[5]	PolicyRefToken ClaimRefToken	RSA-Motor-Claim-3	//rsa:ClaimRef/text()	ScheduleClaimMessage.xml

Example

π^4

L3 Model

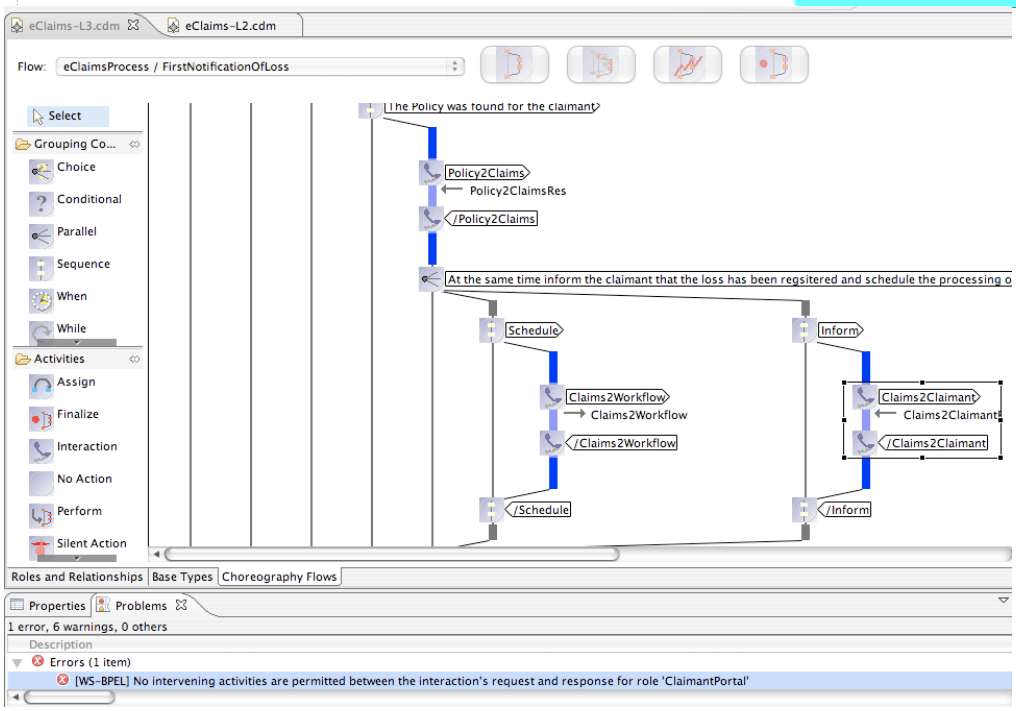


Example

π^4

L3 Model against which we test

Testing against platform

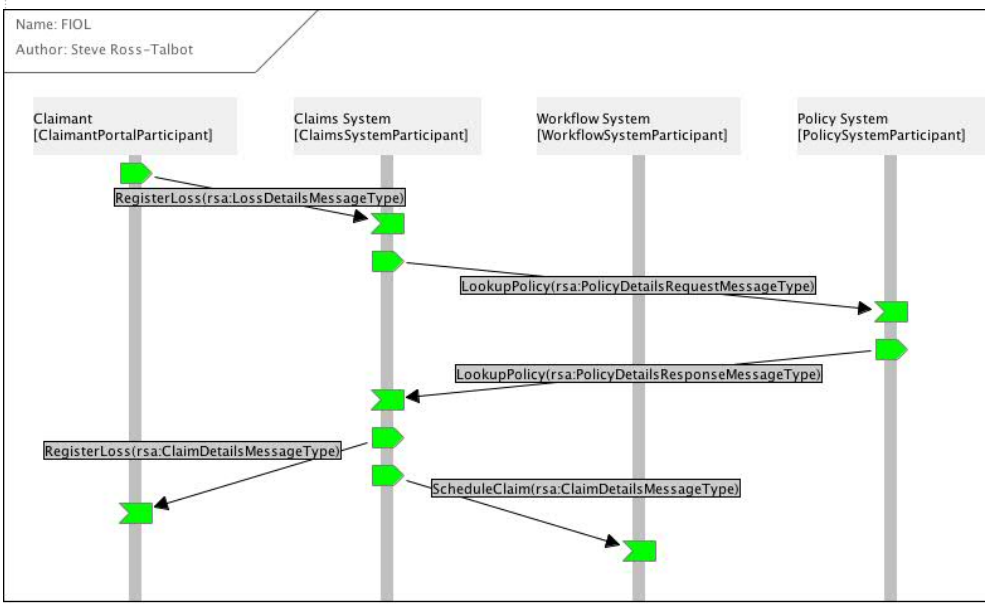


Example

π^4

Testing L3 against R3 requirements

Testing against requirements

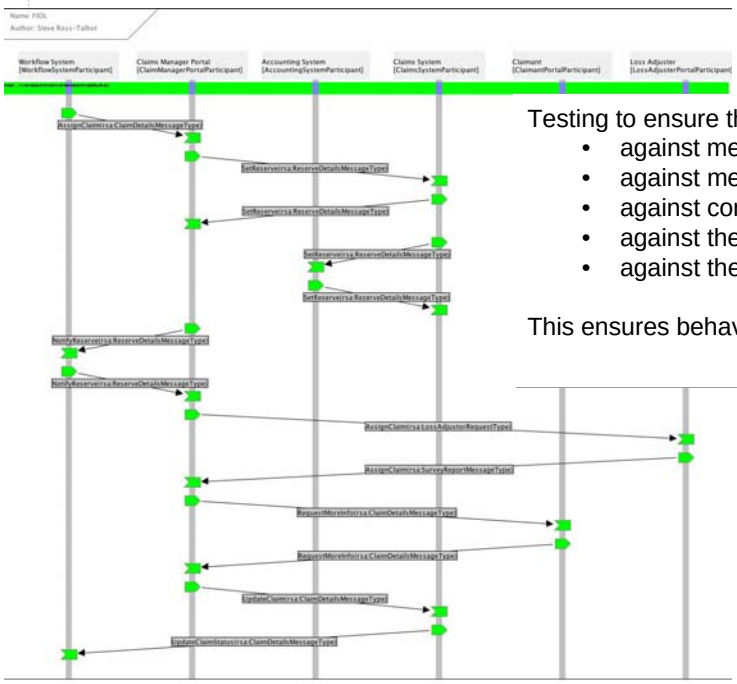


Example

π^4

Testing L3 against R3 requirements

Testing against requirements



Testing to ensure that the mode is correct:

- against message order,
- against message type,
- against correlation identities,
- against the target platform(s),
- against the information model.

This ensures behavior is correct against requirements

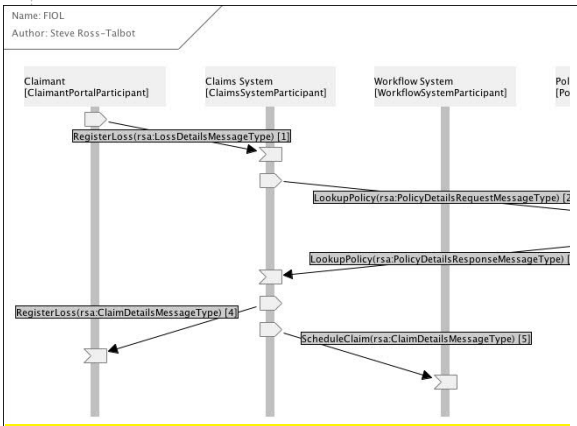
Example

π^4

Generating R4 requirements

Technical Context

Services
Correlations
Types



Ref	Identity Tokens	Identity Values	Query Ex	
[1]	PolicyRefToken	RSA-Motor-ABC123	//rsa:Polic	
[2]	PolicyRefToken	RSA-Motor-ABC123	//rsa:PolicyRef/text()	PolicyDetailsRequestMessage.xml
[3]	PolicyRefToken	RSA-Motor-ABC123	//rsa:PolicyRef/text()	PolicyDetailsMessage.xml
		RSA-Motor-ABC123	//rsa:PolicyRef/text()	
[4]	PolicyRefToken ClaimRefToken	RSA-Motor-Claim-3	//rsa:ClaimRef/text()	LossAcknowledgementMessage.xml
		RSA-Motor-ABC123	//rsa:PolicyRef/text()	
[5]	PolicyRefToken ClaimRefToken	RSA-Motor-Claim-3	//rsa:ClaimRef/text()	ScheduleClaimMessage.xml

Generation of artefacts may include any of the following:

- A UML model for behavior
  - State machine diagrams
  - Activity diagrams
  - UML meta model
- BPMN
  - For each process
- WSDL
  - For each service
- BPEL
  - For each process
- HTML
- Sequence diagrams
  - With message examples
  - With correlation identity specifications

Example

π^4

Generating R4 requirements

WSDL Contract

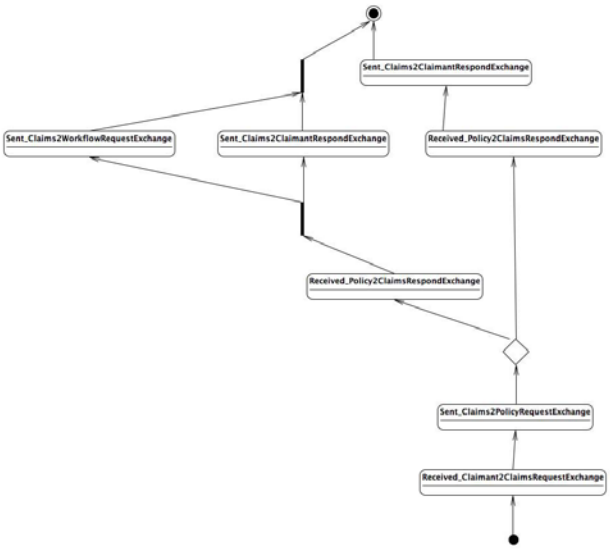
Types and functions

ClaimsSystemBehavior.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org
www.cognizant.com" name="ClaimsSystemBehavior" targetNamespace="http://www.cognizant.co
    <message name="">
        <documentation>
            pi4soa: User defined definitions
        </documentation>
    </message>
    <portType name="ClaimsSystemBehaviorPort">
        <operation name="SetReserve">
            <input message="tns:null" name="ClaimManager2ClaimsSystemRequestExchange"/>
            <output message="tns:null" name="ClaimsSystem2ClaimManagerRespondExchange"/>
        </operation>
        <operation name="UpdateClaimStatus">
            <input message="tns:null" name="Workflow2ClaimsSystemRequestExchange"/>
        </operation>
        <operation name="UpdateClaim">
            <input message="tns:null" name="ClaimsManager2ClaimSystemRequestExchange"/>
        </operation>
        <operation name="UpdateReserve">
            <input message="tns:null" name="ClaimManager2ClaimsSystemRequestExchange"/>
        </operation>
```

UML State Machine

Behavior (order)

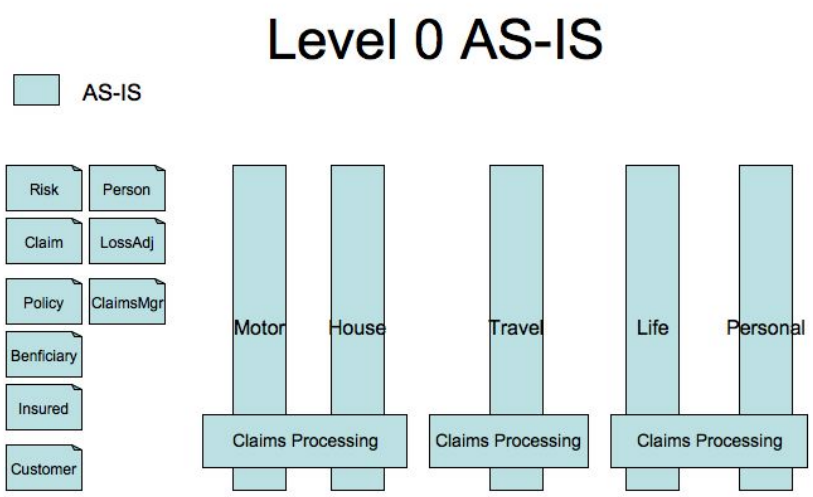


Cognizant

Passion for building stronger businesses

An exercise

AS-IS Picture



AS-IS Requirements

Level	Requirement
R0	1. Implementation of a cross functional eClaims process 2. Key information entities include: Beneficiary, Claim, Claimant, Policy, Survey Report 3. Key constraints include:Portal access is required for: Loss Adjustor, Claimant, Claims Manager
R1	1. A claim must have been made before any further processing of a claim can proceed. 2. Once a claim has been made and validated (see FirstNotificationOfLossFree.scn) the claim may be subject to any number of status enquiries by any authorised entity (Loss Adjustor, Claimant, Claims Manager). 3. Once a claim has been made and validated (see FirstNotificationOfLossFree.scn) the claim may be processed by any authorised entity (Loss Adjustor, Claimant Claims Manager) where processing advances the status of the claim to a conclusion. 4. A claim is deemed to have terminated once it reached a conclusion. 5. A conclusion may result in a payment to one or more parties.

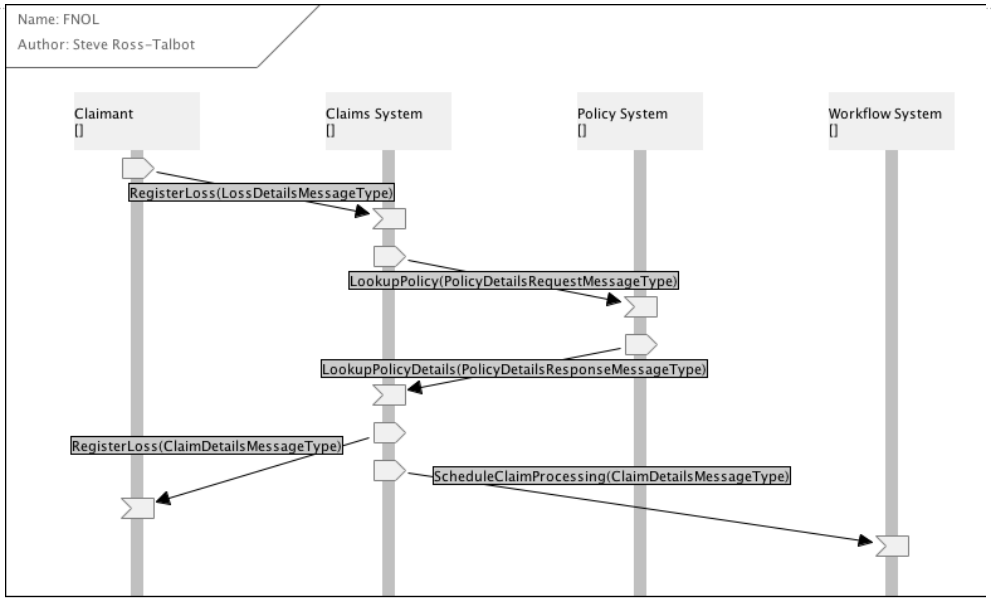
AS-IS Requirements

Level	Requirement
R2	1. Reuse Legacy Policy system, Workflow system, Accounting System and Claim system
R3	1. A claim must be submitted by a claimant with a policy number for the claim or sufficient details to extract a policy number (Name, Address, Post Code) from the Policy System. 2. A claim is always validated against a policy number. A claim with no valid policy number is an invalid claim and is rejected. 3. A claim is given a unique claim reference number after it has been shown to be valid against a policy and this is always used after issue. 4. All claims, except for the first indication of loss will include both claim and policy reference. 5. We shall use WSDL1.1 6. We shall use BPEL

Exercise

π^4

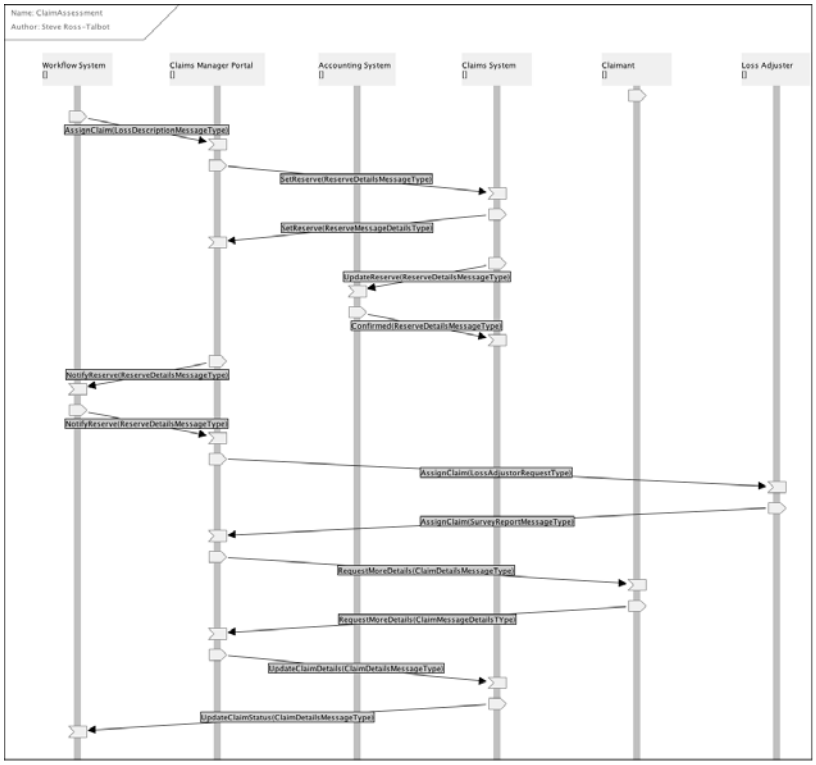
AS-IS Requirements



Exercise

π^4

AS-IS Requirements



Cognizant
Passion for building

Exercise

π^4

What you will need to do

1. Load the eclipse environment into your laptop from the memory sticks
2. You will find
   a) R2 and unbound R3 sequence diagrams
   b) Set of example messages
   c) L0 AS-IS picture and an L0 O-BE picture
3. Create an L1 model and generate an html view of that model
4. Create an L2 model and generate an html view of that model
5. Create an L3 model
6. Copy the sequence diagrams, bind them to the L3 model
7. Test the model agains the bound sequence diagrams

π^4

The Future

BPMN2

- BPMN2
  - Combines both local models and global models
  - Incorporates the benefits of WS-CDL
  - Based on petri-net theory

  - +ve's
    - Local and global behaviors
    - Industry support

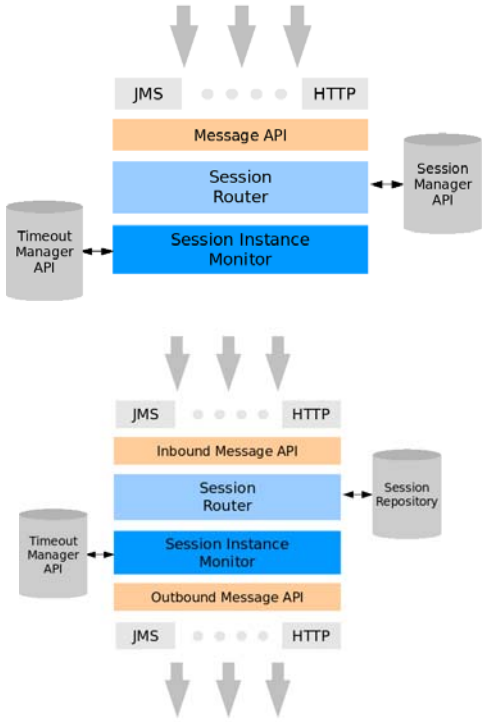
  - -ve's
    - Roles are on the outside not on the inside and complexity is based on interaction and action not on roles
    - Can it model channel passing (i.e. classic authentification)

Scribble

"Scribbling is necessary for architects, either physical or computing, since all great ideas of architectural construction come from that unconscious moment, when you do not realise what it is, when there is no concrete shape, only a whisper which is not a whisper, an image which is not an image, somehow it starts to urge you in your mind, in so small a voice but how persistent it is, at that point you start scribbling" - Kohei Honda 2007

Scribble

- Three languages in one:
  - Protocol (minimal type signature used to guarantee type safety of conversations)
  - Conversation (implements of one or more protocols, with added constraints/logic)
  - Language (integrating multiparty session types, or conversations, into general programming languages)
- Used as underlying type system for pi4soa CDL tool suite to provide conformance checking.
- Refined monitoring of end points.
- Refined runtime environment.



Scribble

Scribble

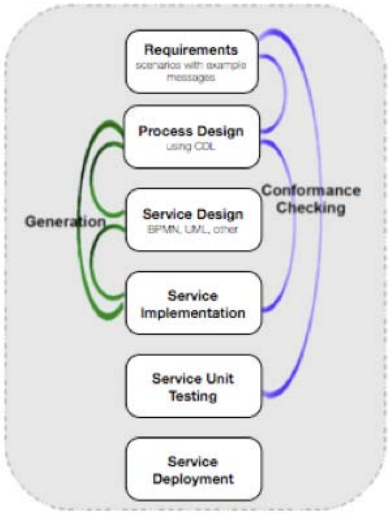
```
protocol HelloWorld
{
        role You, World;
        Hello from You to World;
}
```

WS-CDL

```
package HelloWorld {
   roleType YouRole, WorldRole;
   participantType
      You {YouRole},
   World{WorldRole};
   relationshipType
      YouWorldRel between
         YouRole and WorldRole;
      channelType WorldChannelType
      with roleType WorldRole;
   choreography Main
   {
      WorldChannelType worldChannel;
      interaction operation=hello
         from=YouRole to=WorldRole
         relationship=YouWorldRel
      channel=worldChannel
         { request messageType=Hello; }
   }
}
```

Overlord

- Uses WS-CDL outside of the scope of WS-* and within a more general SOA infrastructure.
- Design methodology and runtime support
- Compliment other development techniques and improve the resiliency and robustness of SOA-based applications, no matter how they are ultimately deployed.



π^4

Q&A