

V3 Message Building Strategies

Understanding And Populating
HL7 V3 PHIN Messages

Dale Nelson
Zed-Logic Informatics, LLC

Agenda

- Overview
- HL7 Basics
- Processing Basics

Overview

- Scope
 - V3 PHIN message construction
 - Specifically PHIN Notification Message
 - Applicable to all

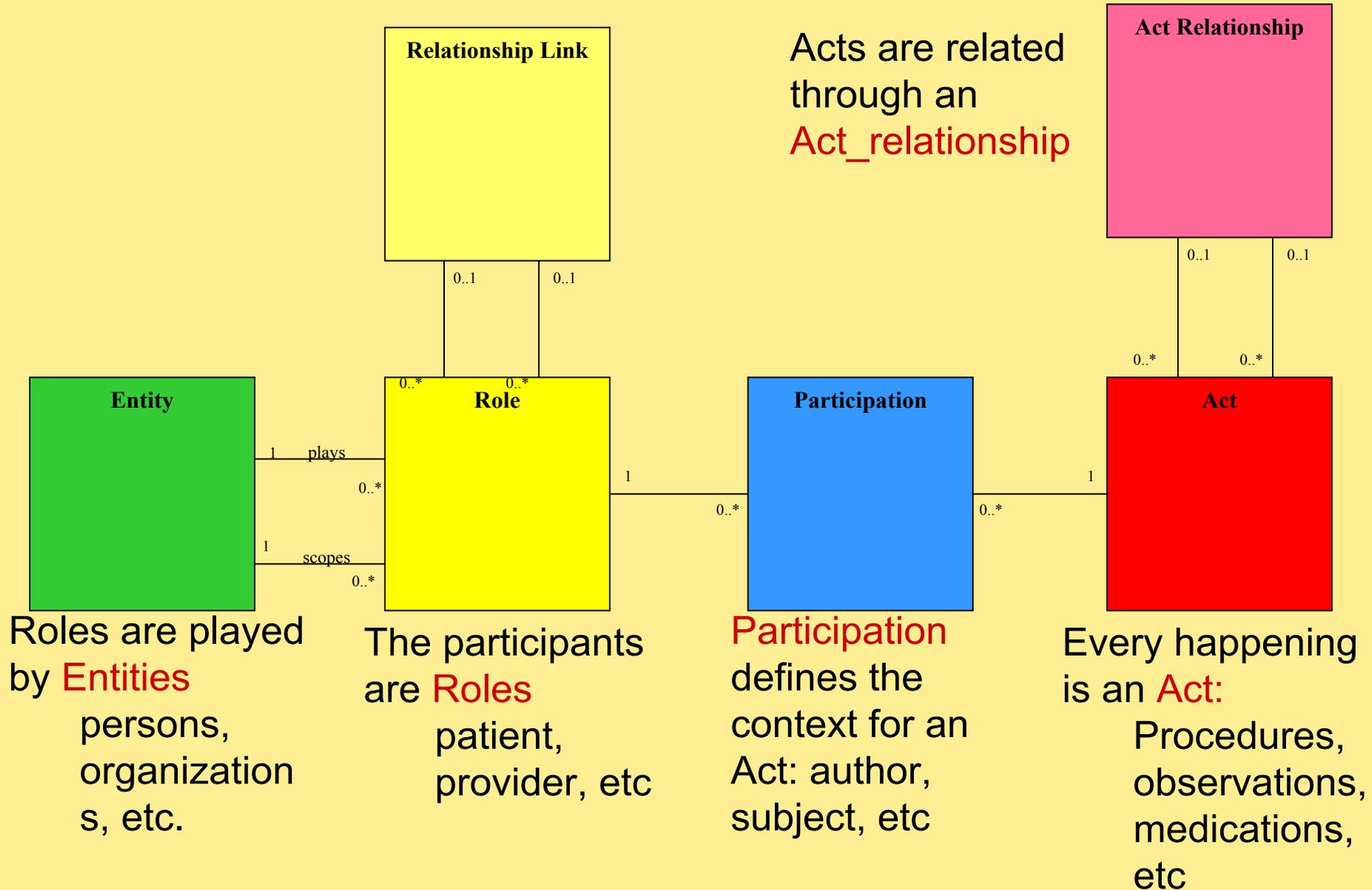
General Comments

- CDC PHIN is an Early Adaptor of V3 Messaging Standards
 - Will learn and refine process over time
 - Experience will feed back into HL7 process and model
 - Part of distinguished group
 - ~ 30 early adaptor efforts underway worldwide
- MANY ways to build messages
 - There is no “best” way
 - Best practices still open for discussion
- Current Tools easily broken
 - Very complex schemas
 - Feedback process already has resulted in schema simplification
 - Beware the phrase “supports W3C schema”
- Vendor/tool agnostic
- RIM is an ABSTRACT model for healthcare
 - Logical Architecture - deals with objects
 - Requires a Logical Data Model for persistence
 - Physical Model may be VERY different
 - Could be used as a first approximation in a reference implementation

Also

- Messages represent object instance serializations in the Refined Domain Model
 - Objects = Identity + Data + Meaning
 - They do NOT represent only data structures
- Messaging Process is
 - Not Data Messaging (Moving data from A to B)
 - Info Model Messaging (HL7 V3)
 1. Map Data model to Information Model (RIM)
 2. Transport message
 3. Process Information Model (receiver)

HL7 RIM Core Classes



HL7 Methodology Steps

- From RIM to XML Instance
 - Refine, clone, constrain, walk, format
- XML Knowledge IS NOT ENOUGH
 - Need Domain Knowledge to map into messages (Implementation Guides!)

Processing Basics

- Focus on Sender

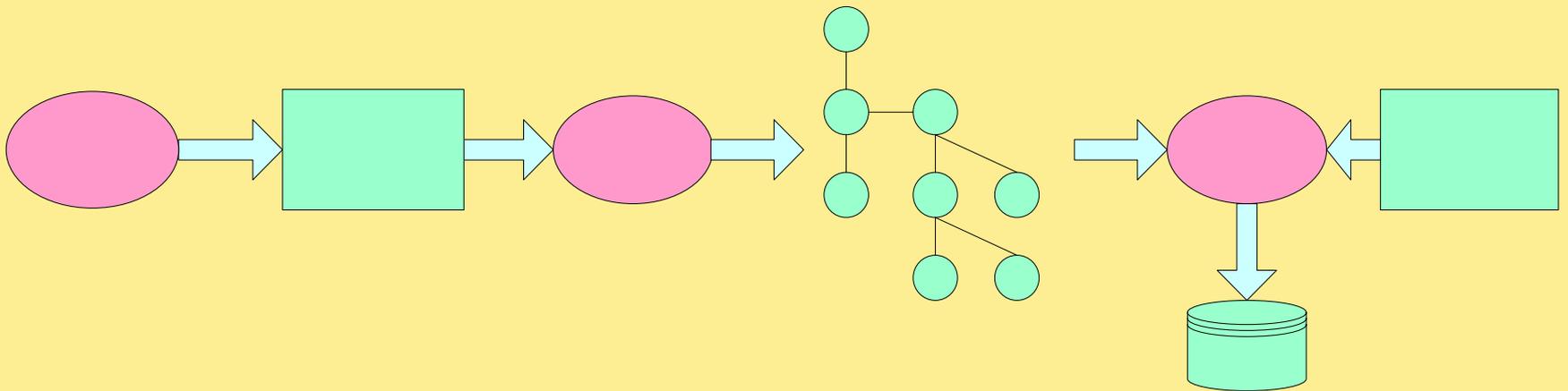
Client (Sending) Tasks

- Build Message
- Send Message
 - Use system such as PHIN-MS

Receiver Tasks

- Receive payload via PHIN-MS
 - Unwrap
- Parse (and possibly validate) payload
 - SAX
 - JAX-B
 - Other (Castor, etc)
 - .NET
 - Integration engine / product
- Apply message & semantics to application

Receiver Processing



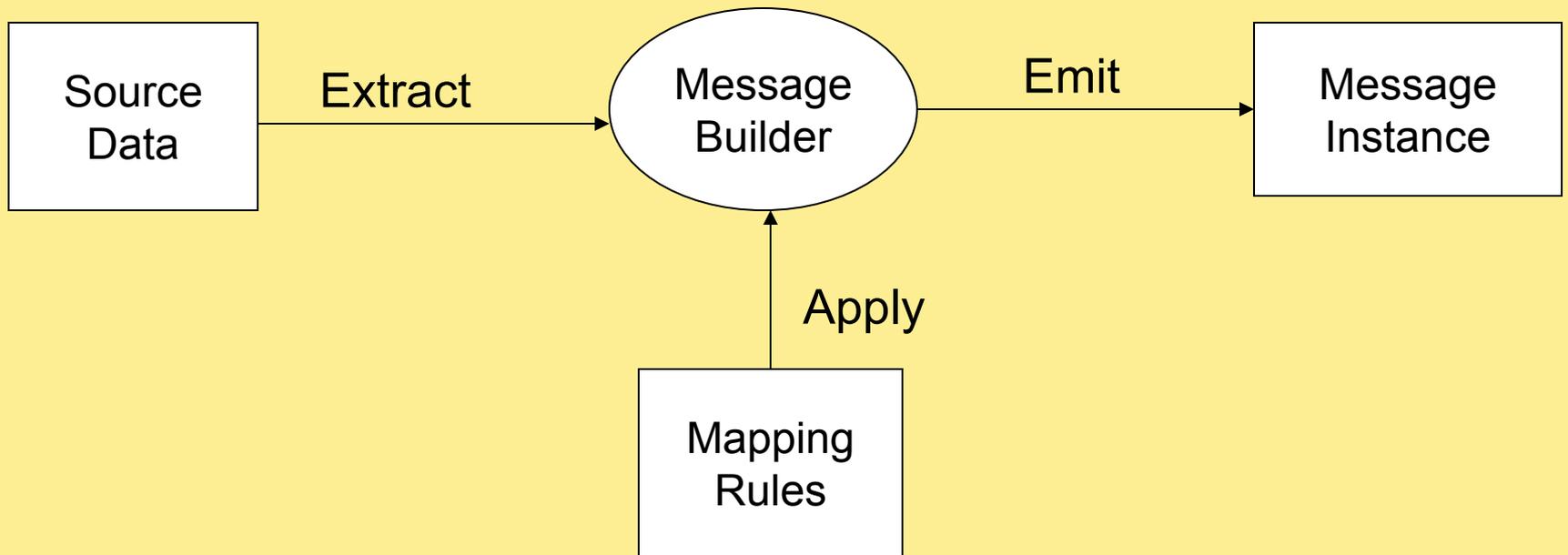
Receiver Processing

- Role-based behavior
 - Application Role = Archivist, but can be active role, such as order manager, etc.
- Constraint processing
 - There are many things you can do in an instance that are valid by the schema, but not valid by HL7
 - IVL_TS (high, low, center, width)
 - II (extension, root)
 - Future tool (OCL) to apply constraints in schema to message
- If objects instantiated
 - Use inheritance to process
 - Can trigger “work” via detection of state changes in focal objects
 - Workflow
 - Rules Reasoning
 - Simple Repository population

Sender Processing Steps

1. Capture data from disparate input sources & data models
 - Existing Repository
 - Message Feeds
 - Application Data
2. Translate to Common Information Model
 - RIM based
3. Serialize as XML message
4. Send via PHIN-MS

Extraction & Mapping



Message Builder

- Start with R-MIM model
- Build Object Graph of Class Clones
- Extract and map source data to attributes
- Translate to DOM
- Serialize
- Validate ?
- Send via PHIN-MS or equivalent
- NOTE:
 - You don't *have* to do all of this
 - You could simply hand-code XML
 - It just makes extensibility and maintenance easier

Data Extraction & Mapping

- Push Model
 - Populate the message by visiting the data (push data)
 - Data-Driven XML Binding
- Pull Model
 - Build generic message, walk, pulling data from source as needed
 - Custom DOM application
 - Class Clone model
- Push-Pull Model
 - Populate intermediate representation by visiting data, then
 - Build generic message, walk, pulling data from intermediate rep as needed
 - Data-driven XML Binding + Class Clone model

Push (1)

Data-Driven XML Binding

- Extract XML from repository via tools
 - Most DB vendors provide DB schema to XML Schema generation
- Use XSLT (or similar) to translate XML instances to V3 messages
 - Probably can't be done? (reasonably)
 - V3 message semantics too complex/rich for reasonable transforms

Push (2)

Data-Driven XML Binding

- Typical of JAX-B, Castor, many vendor apps
 - Put Data into Canonical XML format
 - Extract XML from repository via tools
 - Most DB vendors provide DB schema to XML Schema generation
 - Generate (Java) classes from XML Schema
 - Unmarshall DB XML extracts to class instances
 - Manipulate class instances and serialize as XML

Data-driven

- Problems:
 - Not likely DB schema resembles R-MIM
 - Schema represents the input data model, not the V3 Message format
 - Still need to translate to V3 Message Schema
 - » Haven't saved any work

Push (3)

Data-Driven XML Binding

- Typical of JAX-B, Castor, many vendor apps
 - Generate (Java) classes from V3 Message XML Schema
 - Populate classes via ObjectFactory
 - Manipulate class instances and serialize as XML

Pull DOM

- Custom program to build DOM tree
 - Intrinsic knowledge of message structure
 - Not very re-useable
- Relatively straightforward
- Serialization easy

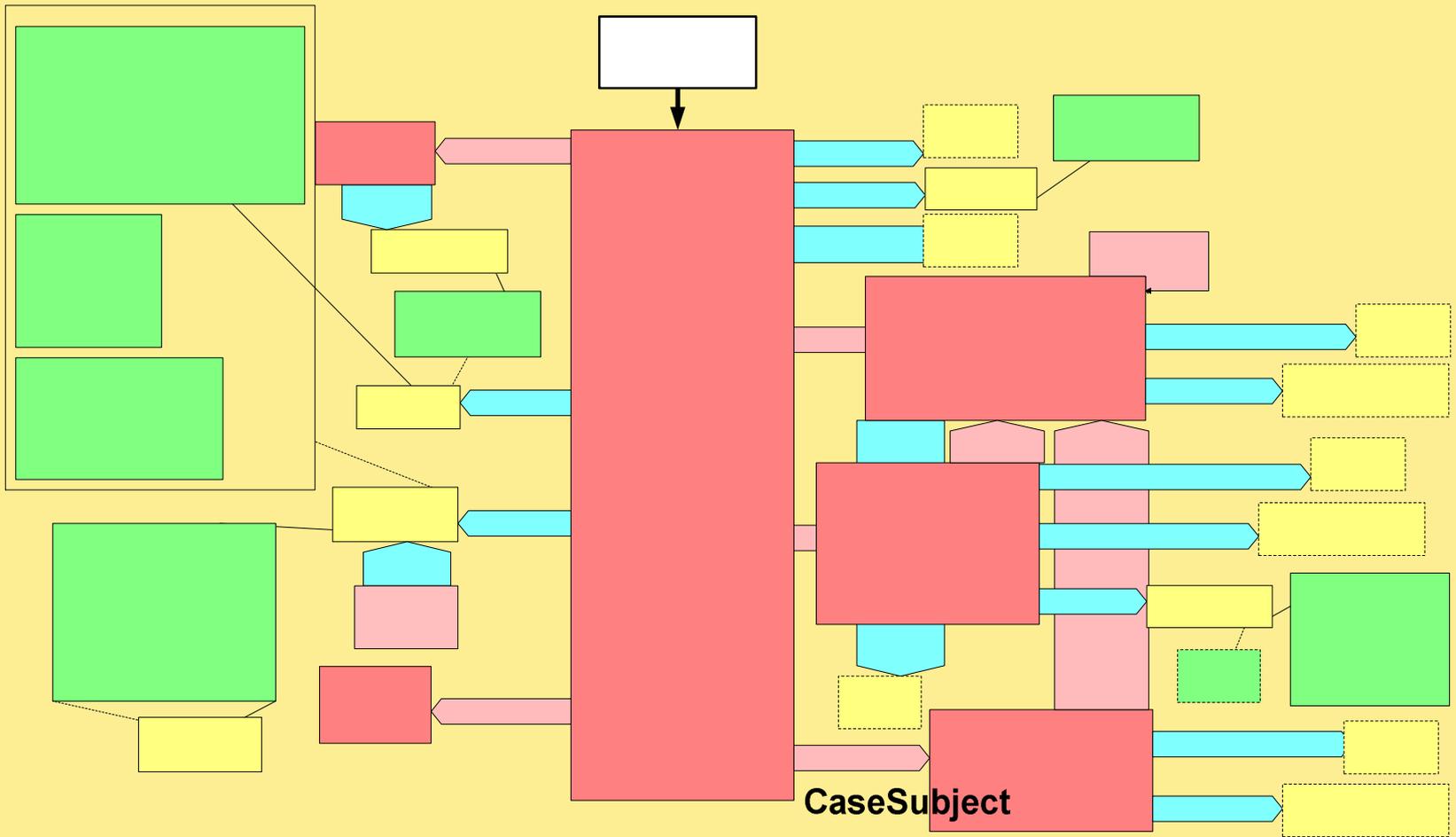
Pull, Push-Pull Object Graph Methods

- Java Classes
 - Get, set methods on attributes
 - Constructors to set default values
 - Methods to access associated classes

HL7 Java SIG

- The goal of the Java SIG
 - to develop a software API for the parsing and building of messages.
 - Build a message given a RIM object graph that has already been populated with message content.
 - Parse a given message creating and populating a RIM object graph as required to contain the message content.

PHIN Notification R-MIM



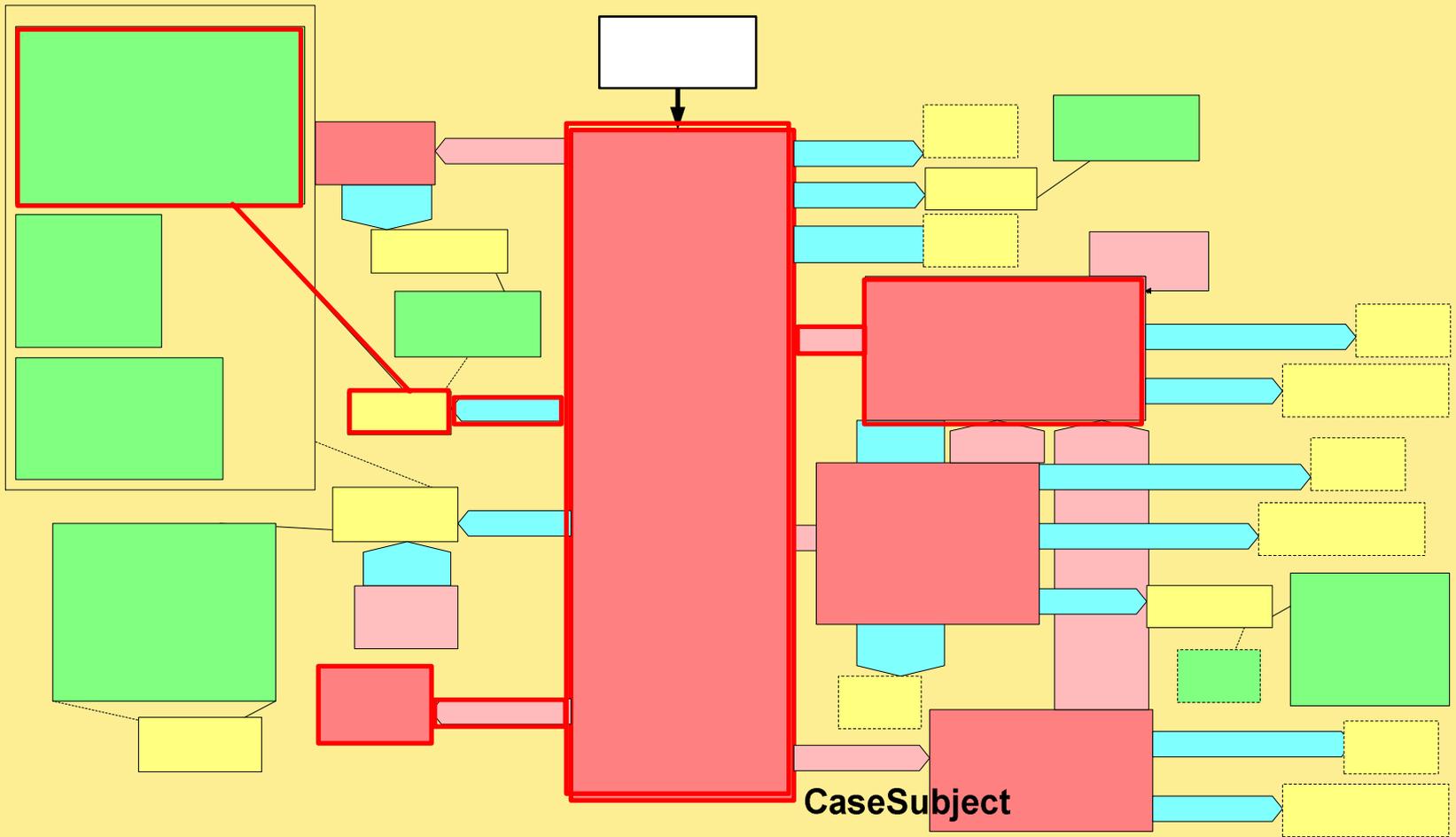
Person

classCode*: <= *PSN*

determinerCode*: <= *INSTANCE*

id:

But we only use this much for “Msg A”



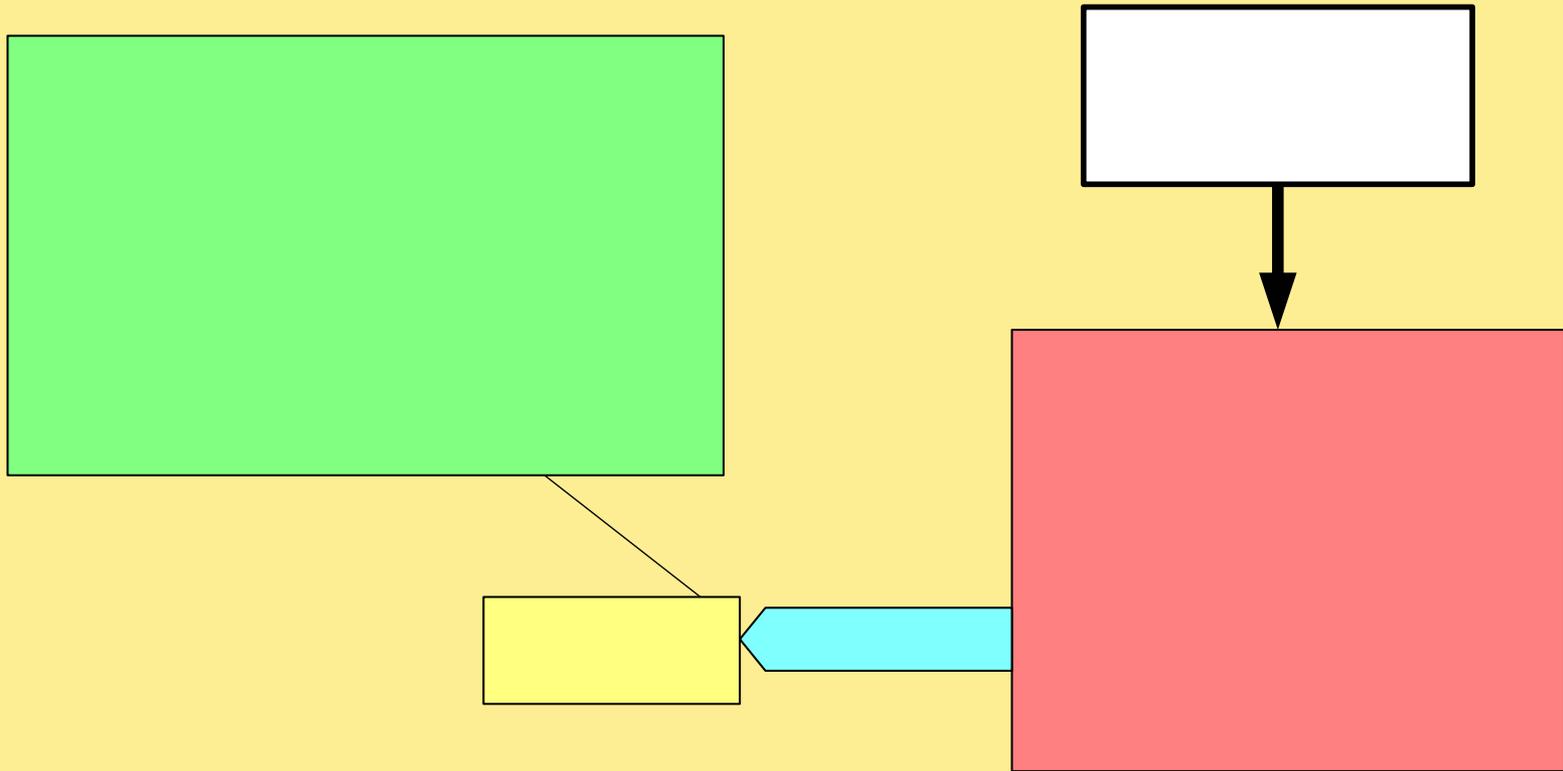
Person

classCode*: <= PSN

determinerCode*: <= INSTANCE

id:

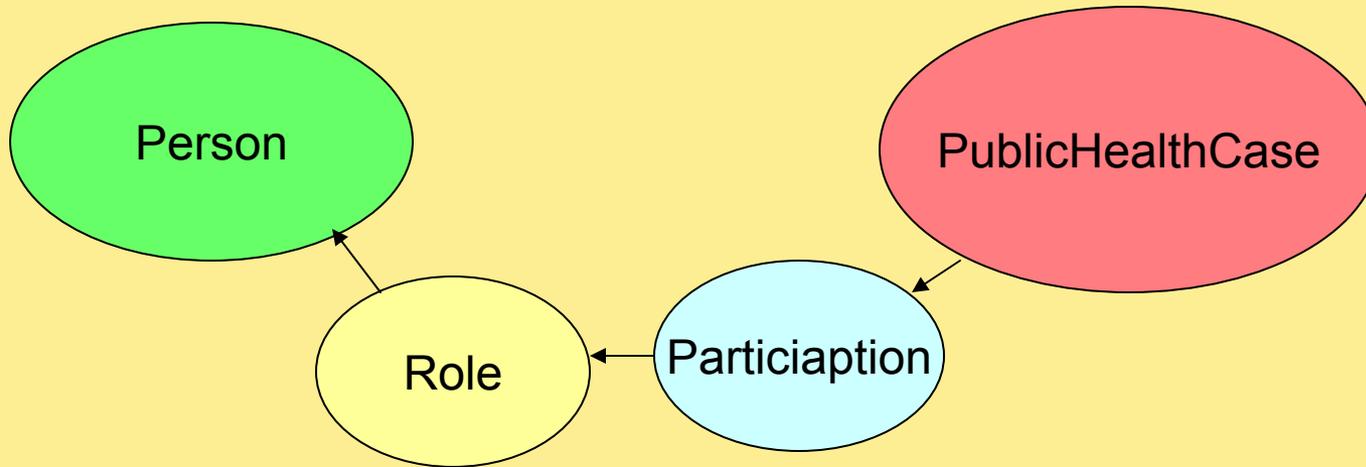
R-MIM Snippet of Interest



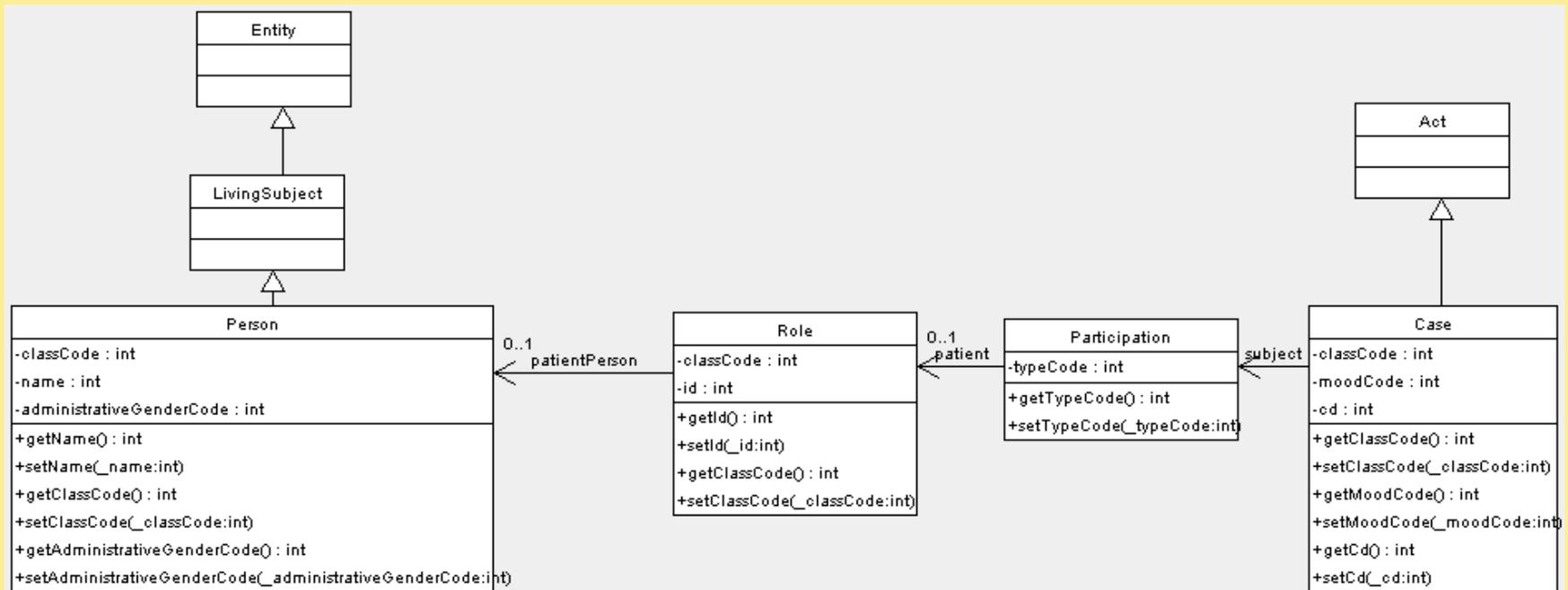
Sample Snippet

```
<publicHealthCase>
  <id/>
  <code codeSystem="2.16.840.1.114222.4.19.999" code="10101" displayName="Hepatitis C, acute"/>
  <text>general comments</text>
  <statusCode code="active"/>
  <effectiveTime>
    <low value="20030108"/>
  </effectiveTime>
  <activityTime>
    <low value="20030107"/>
  </activityTime>
  <detectionMethodCode nullFlavor="UNK"/>
  <transmissionModeCode nullFlavor="UNK"/>
  <component3 typeCode="COMP">
    <observationProcess>
      <code codeSystem="2.16.840.1.114222.4.19.999" code="INV128"/>
      <value xsi:type="CV" codeSystem="2.16.840.1.114222.4.19.222" code="Y"/>
    </observationProcess>
  </component3>
  <subject2 typeCode="SBJ">
    <patient>
      <patientPerson>
        <administrativeGenderCode codeSystem="2.16.840.1.114222.4.19.999" code="M"/>
        <birthTime value="20000101"/>
        <deceasedInd value="false"/>
        <addr use="PST">
          <postalCode>68501</postalCode>
        </addr>
        <raceCode codeSystem="2.16.840.1.114222.4.19.999" code="2106-3"/>
        <raceCode codeSystem="2.16.840.1.114222.4.19.999" code="2108-9"/>
        <ethnicGroupCode codeSystem="2.16.840.1.114222.19.999" code="2100-0"/>
      </patientPerson>
    </patient>
  </subject2>
</publicHealthCase>
```

Model in UML?



UML Class Generation



Generated Class (Case)

```
public class Case extends Act {
    // attributes
    private CS classCode;
    private CS moodCode;
    private CV cd;
    // associations
    public Participation participation;
    // access methods for associations
    public Participation getParticipation() {
        return participation; }
    public void setParticipation(Participation participation) {
        this.participation = participation; }

    // operations
    public CS getMoodCode() {
        return moodCode; }
    public void setMoodCode(CS _moodCode) {
        moodCode = _moodCode; }
    public CS getCd() {
        return cd; }
    public void setCd(CS _cd) {
        cd = _cd; }
}
```

Generated Class (Person)

```
public class Person extends LivingSubject {
    // attributes
    private CS classCode;
    private AD address;
    private CV administrativeGenderCode;
    // operations
    public AD getAddress() {
        return address; }

    public void setAddress(AD _address) {
        address = _address; }

    public CS getClassCode() {
        return classCode; }

    public void setClassCode(CS _classCode) {
        classCode = _classCode; }

    public CV getAdministrativeGenderCode() {
        return administrativeGenderCode; }

    public void setAdministrativeGenderCode(CV _administrativeGenderCode) {
        administrativeGenderCode = _administrativeGenderCode; }
} // end Person
```

Pseudo -Code

```
Context context = ContextFactory.newContext("... identification of input data ...");
```

```
Case msg = msgFactory.newCase();  
msg.setCd(dtFactory.newCS(codeSystem, "629808"));
```

```
Participation p = msgFactory.newParticipation();  
Role r = msgFactory.newRole();  
Person psn = msgFactory.newPerson();
```

```
psn.setAddress(dtFactory.newAD("zip", context.getPostalCode()));  
psn.setAdministrativeGenderCode(dtFactory.newCV(codeSystem,  
    context.getGenderCode()));  
r.setPatientPerson(psn);           // attach person to role  
p.setPatient(r);                   // attach role to participation  
msg.setParticipation(p);           // attach participation to Case
```

Data Mapping

- How do we get the data to populate the objects?
Careful application of Implementation Guides
Always need a “context” (Case ID, Obs Id, etc)
 - Pull Model
 - Use custom logic to obtain data (e.g., JDBC, ADO...)
 - Push-Pull
 - Push relevant data to intermediate source
 - Use Data driven XML Binding
 - Pull data from intermediate graph

Gotchas

- Datatypes
 - Special handlers needed for many datatypes (Complex & composite types)
 - TS
 - “20030515T104500” => “20030515104500-0500”
 - “15-May-03 10:45:00” => “20030515104500-0500”
 - IVL_TS
 - “12-May-1987 to 12-May-1988” =>

```
<effectiveTime>
    <low value="198705122000" inclusive="true"/>
    <high value="198705122130" inclusive="true"/>
</effectiveTime>
```
 - RTO_PQ_PQ
 - “15 mg/dL” =>

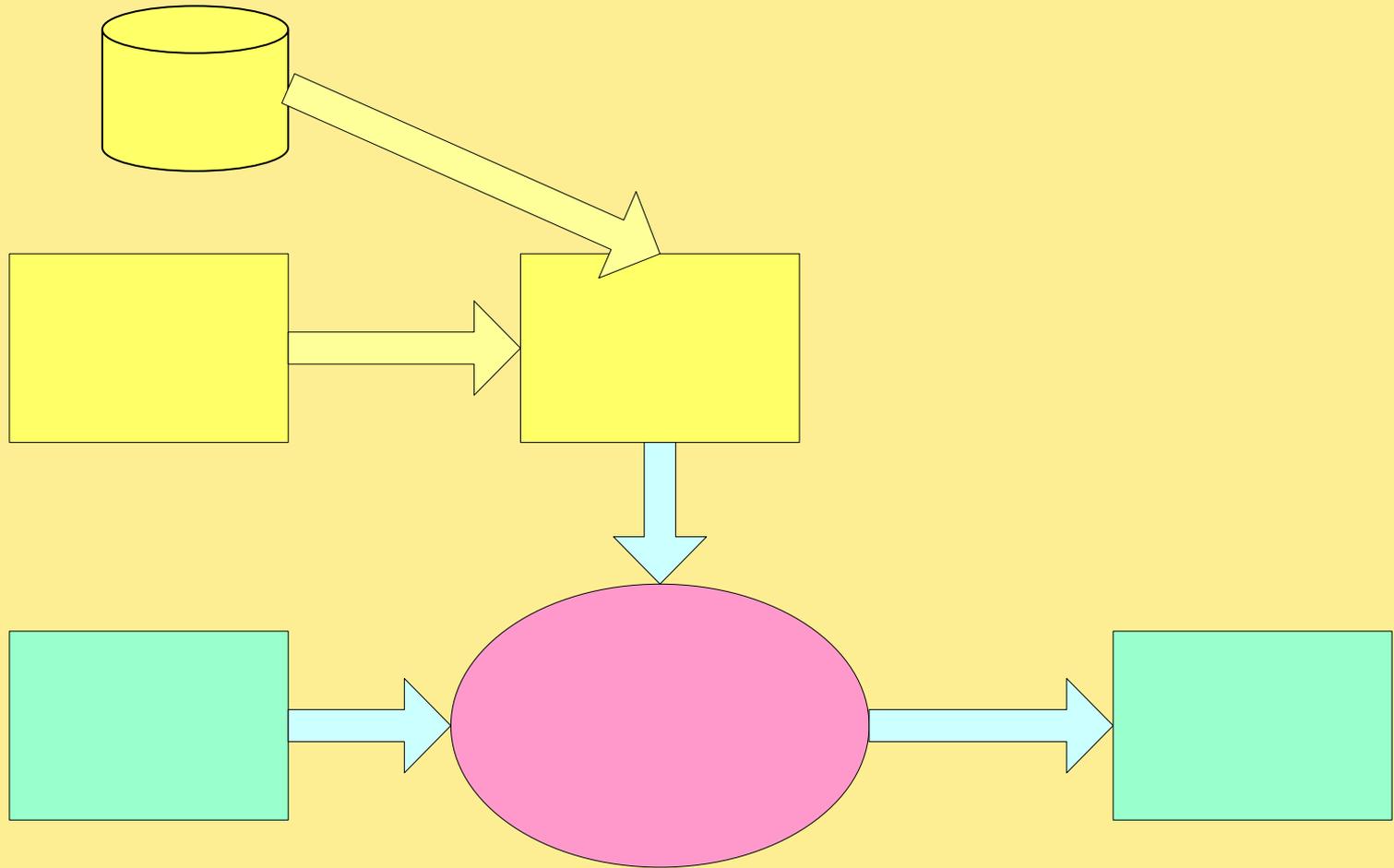
```
<foo>
  <numerator value="15" unit="mg" />
  <denominator value="1" unit="dL" />
</foo>
```
 - RTO_MO_PQ "USD189.95:1"

```
<foo>
  <numerator value="189.95" currency="USD" />
  <denominator value="1" unit="" />
</foo>
```
 - Null Values
 - <<null>> => <foo nullflavor="UNK"/>

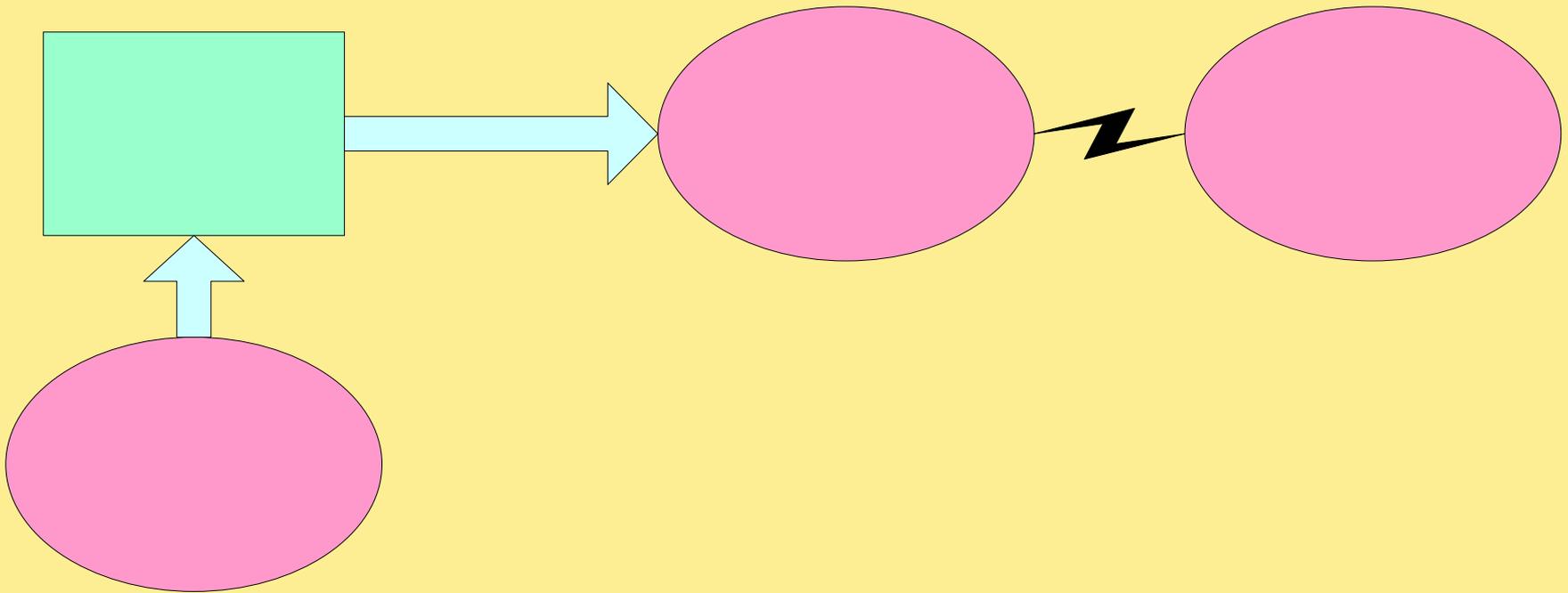
Walk graph

- **Serialize**
 - Order dependencies
 - HL7 JAVA SIG uses HMD information to drive order
 - Could use schema to define a transform
 - COTS or Open source Serializer
`org.apache.xml.serialize`

Touch Ups



Ship It



Questions?

Dale Nelson

Principal Consultant

Zed-Logic Informatics, LLC

dale@zed-logic.com