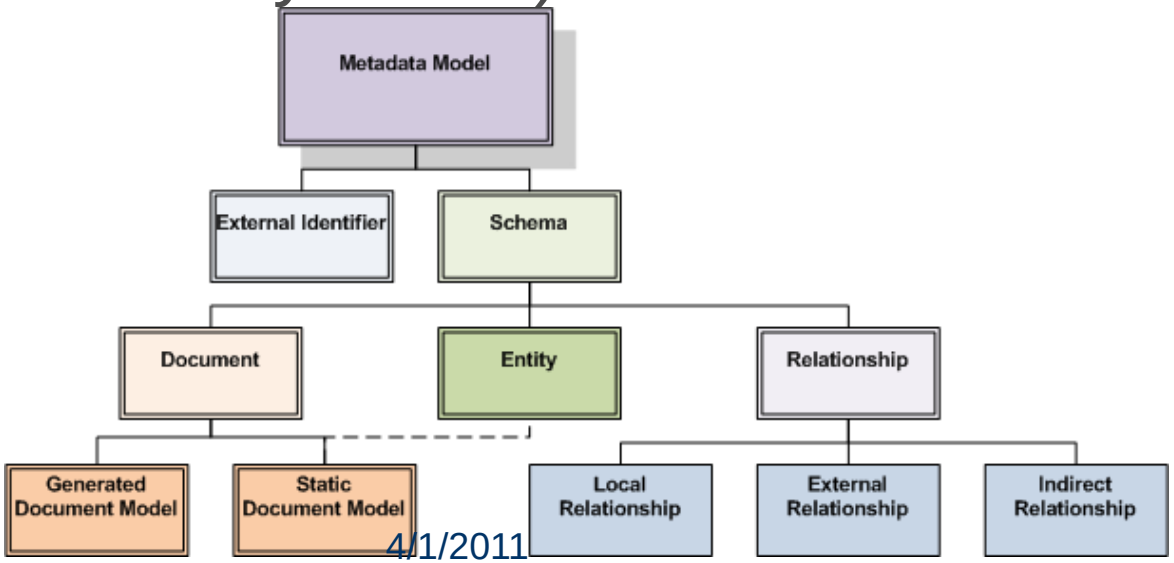
Security Automation Content Repository Demo

Dave Waltermire
Stephen Sill
Paul Cichonski
Glen Strickland

THE XML SCHEMA META MODEL

A Meta Model Based Approach

- Provides an abstraction over XML schema
- Defines the primitives of the meta model and how they relate to an XML schema instance (e.g OVAL, XCCDF, OCIL, CPE, Vulnerability Model)



Meta Model - Entity

- Represents a type in an XML schema (e.g. OVAL definition, OVAL Variable, XCCDF Group, XCCDF Rule, etc.)
- Entities can be used as building blocks for deconstructing and reconstructing XML instances

Meta Model - Relationship

- Describes an arc between two entities or between an entity and an external identifier

- Entity-to-Entity
  - Local Relationships define intra schema relationships
  - External Relationships define inter-schema relationships
- Entity-to-External Identifier
  - Indirect Relationships define relationships to external identifiers (boundary objects)

- Examples:
  - Local: An OVAL Definition has a criterion that references a state
  - Local Compositional: An XCCDF Group contains an XCCDF Rule
  - External: An XCCDF Rule has a check that references an OVAL definition
  - Indirect: An OVAL definition references a CVE

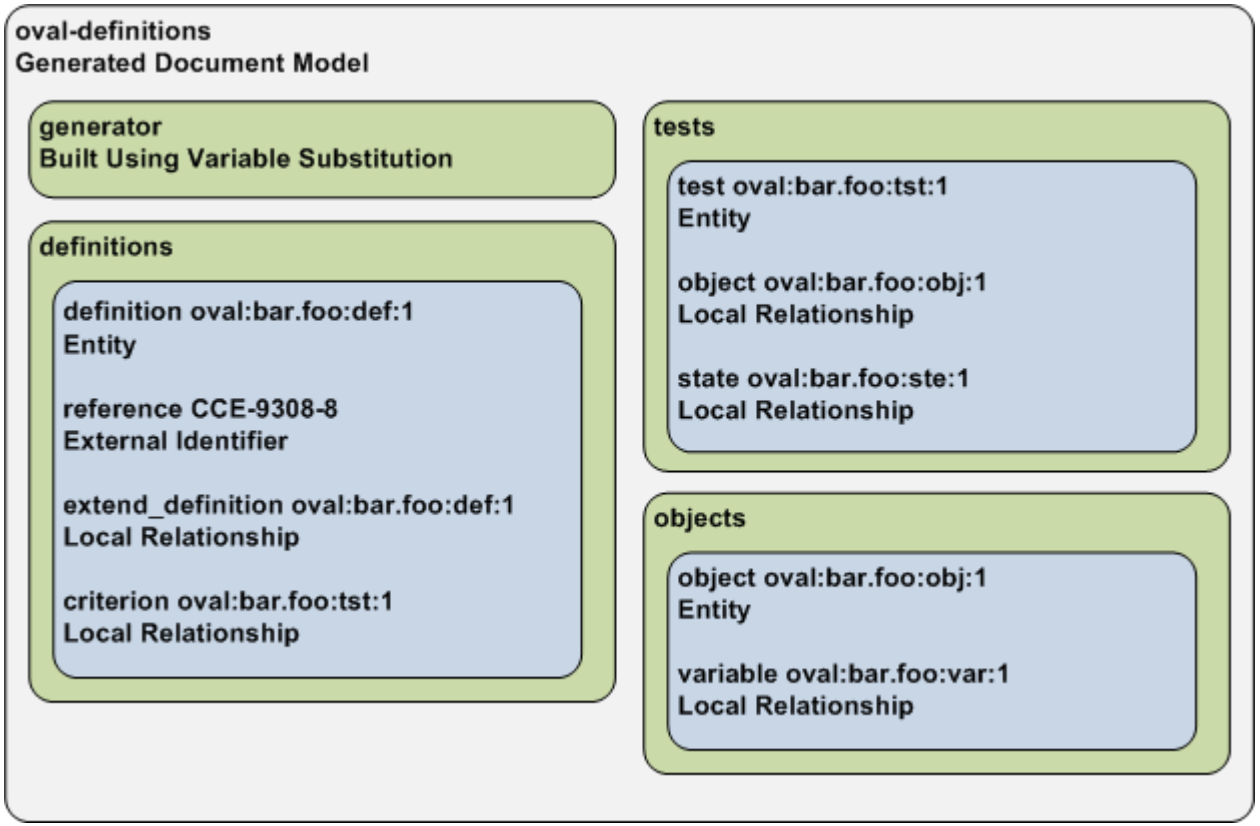
Meta Model - Document

- Represents a collection of entities
- Generated Documents: Non-indexed documents that are simple containers of entities (e.g. oval_definitions, cpe-list, NVD vulnerability data, etc.)
- Static Documents: Indexed documents, that are Entities in themselves, that contain other entities (e.g. XCCDF Benchmark)

Meta Model Use

- Enables identification of documents, entities and relationships
- Supports decomposition of XML instances for persistence
- Enables reconstitution of persisted content into XML instances

The OVAL Document Model with Meta Model Constructs



Process for Decomposing a Document into Entities

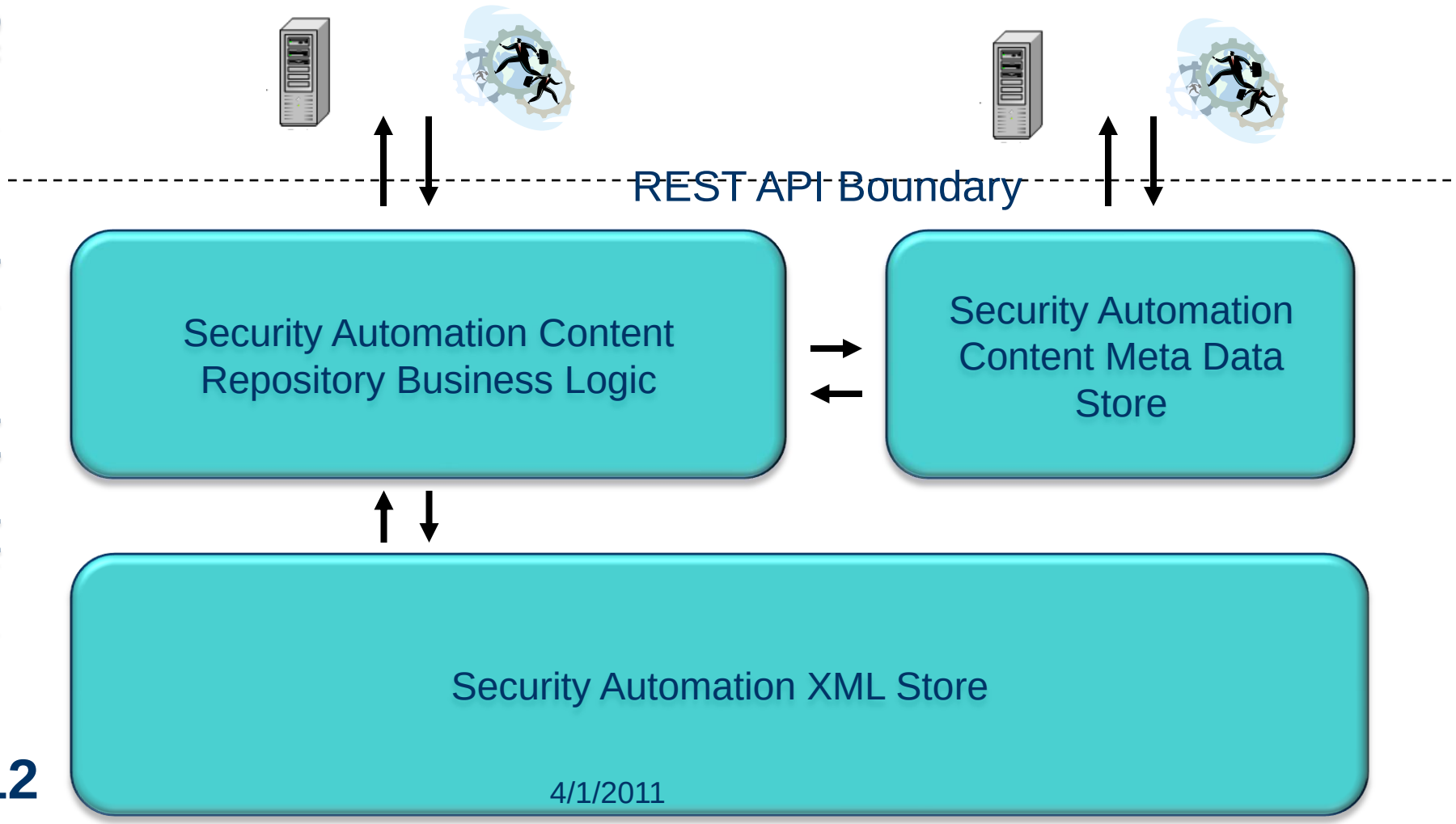
1. Identify the document model
2. Based on Meta Model information, identify the entities that exist within the document instance
3. Parse out each entity
4. For each entity:
    a. Identify the entity type
    b. Parse the entity identifying relationships and external identifiers defined for the entity within the meta model
    c. Extract each relationship
5. Persist entities and relationships
    1. Persist relationships and other metadata to the metadata store
    2. Persist content as unstructured text to the content store

Process for Generating a Document From a Collection of Entities

1. Resolve query generating result sets
2. Analyze result sets to determine document models need for output
3. Output document collection root element
4. For each document:
    a. If document type is generated:
        1. Generate the top-level document structure
        2. For each entity in the model output the entity
    b. If the document type is static:
        a. Output the entity
    c. For each entity, output any compositional entities

THE REPOSITORY ARCHITECTURE

Current Architecture



DEMO

FUTURE ACTIVITIES

Next Steps

- Complete support for Compositional Schema Models (e.g. XCCDF)
- Handle schema oriented versioning
- Handle entity versioning
- Handle XML signatures
- Implement a federated content repository model
  - Define a repository communication model
    - Service endpoints for querying and publication
    - XML Models
    - APIs leveraging services
  - Determine a method to support repository resolution / registration

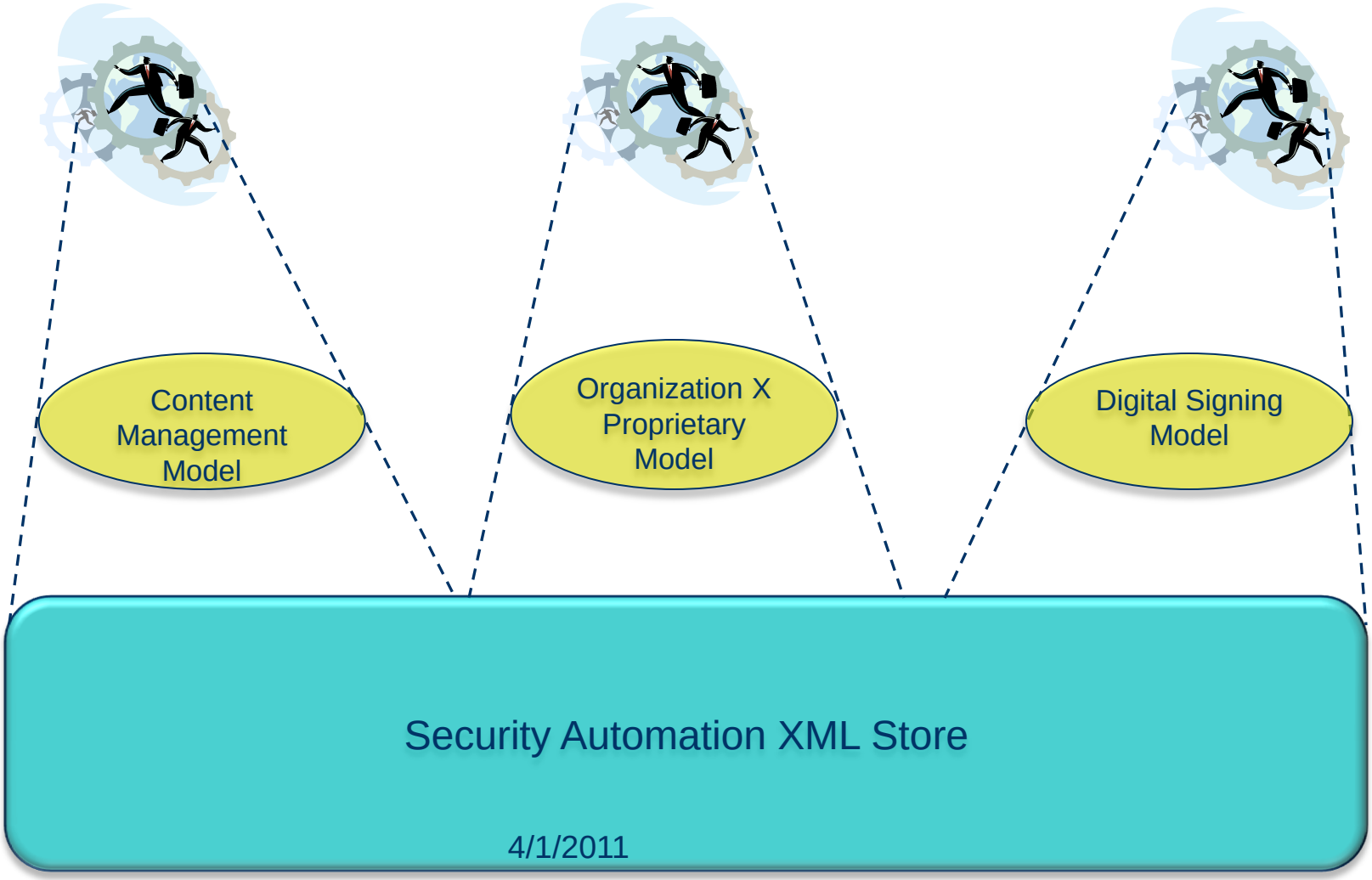
Metadata store may hold metadata adhering to disparate viewpoints of security automation data

- Metadata model described in previous slides is the core model required for operation of repository (i.e, defines the primitives).
- Repository users may create distinct models capturing alternative viewpoints of automation data.
  - Automation data can be annotated with these new models in the same way.
  - Alternative models may support additional capabilities.

Example of an alternative metadata model

- One useful model may describe classes and relationships applicable to content management.
  - This model would capture classes and relationships useful when writing generic creation and update code.
  - At an abstract level, code must know how a node under inspection for update is related to other nodes in a graph.
- Potential Classes:
  - SharedData, UnSharedData, RootNode, LeafNode.
- Potential Relationships:
  - isRelatedTo (transitive relationship to capture all possible nodes – optimization for faster querying of potential conflicts).

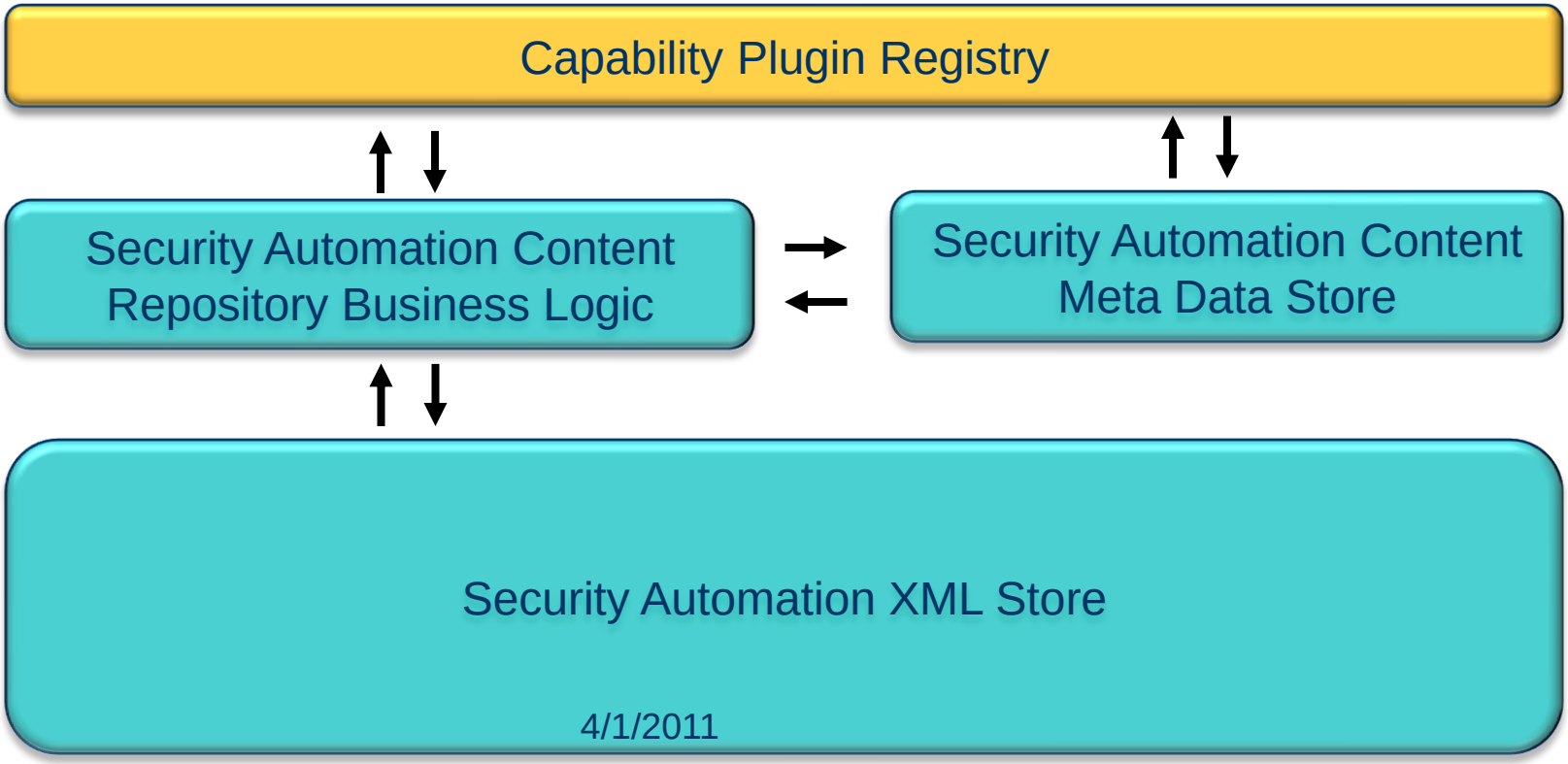
Each metadata model supports an alternative lens through which to view the same data – different lenses support disparate data management use cases



In the future it may be possible to extend architecture to support eclipse-like plugins for disparate capabilities

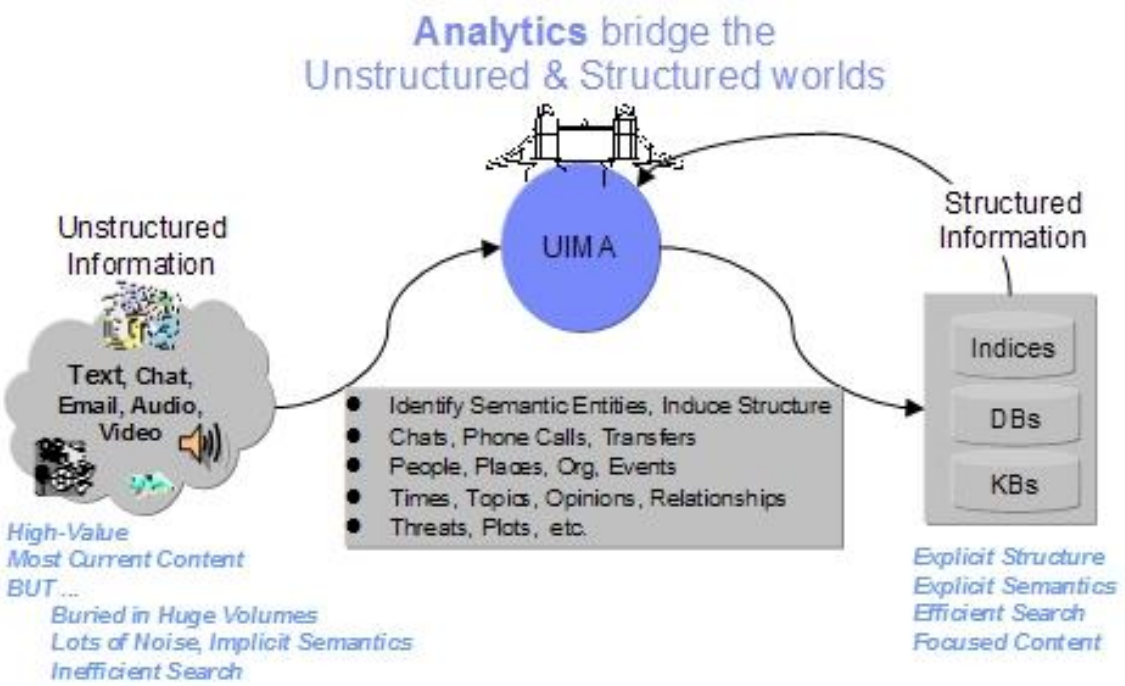
Capabilities registered with framework will support new use cases. They will provide:

1. Capability specific metadata model.
2. Annotation directives between automation data and model.
3. Code for capability business logic.



Frameworks like Apache UIMA may serve as an exemplar for future work

- UIMA focused on connecting unstructured information to structured metadata adhering to user defined models.
- Similarly, we are focused on connecting XML instance data with structured metadata.
- Learning from frameworks like UIMA may guide future efforts.



* Image taken from Apache UIMA Website (http://uima.apache.org/doc-uima-why.html)

Questions and Other Info

Questions?

Project:

http://code.google.com/p/security-automation-content-repository/

Discussion:

http://groups.google.com/group/security-automation-content-repository-discuss