Global InfoTek, Inc.

Contributions to a Semantically Based Intelligence Analysis Enterprise Workflow System

Robert C. Schrag
Jon Pastor
Chris Long
Eric Peterson
Mark Cornwell,
Lance A. Forbes
Stephen Cannon

Global InfoTek, Inc.
SET Corp.
Solutions Made Simple, Inc.

ONTOLOGY FOR THE INTELLIGENCE COMMUNITY ***(OIC)***

21 October 2009

Tangram (2006–2008)

IARPA program goals:

- Prototype a surveillance and alerting system to counter the terrorist threat.

→ *Automate analytical workflows with "plug-and-play" algorithmic components.*

- Automatically select the "best" component in a given component class, based on:
  - Data profiling
  - Component execution profiling
  - Machine learning-based performance prediction
- Execute workflows on a massive scale using grid computing.

Our Contributions

- Uniformly accessible semantic store conforming to an enterprise-wide ontology
- Logic programming-based, forward-chaining query language for components to access data from the store

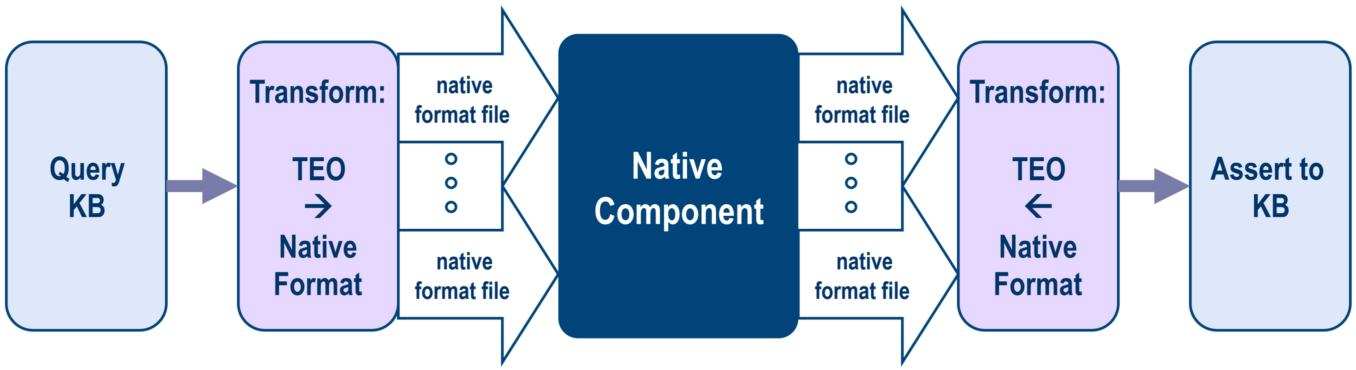
→ **Software toolkit to streamline introduction of additional legacy software components as semantically interoperable workflow building blocks**

→ Branching context representation to organize workflow components' analytical hypotheses

Toolkit

- Allows a knowledgeable user to “wrap” a newly installed component for workflow operation, quickly
- Provides a compact, declarative specification
- Covers certain widely used input / output formats:
  - Comma-separated value (CSV) files
  - Any delimited text
- Built to work with AllegroGraph, from Franz, Inc.

Wrapping a Legacy Workflow Component

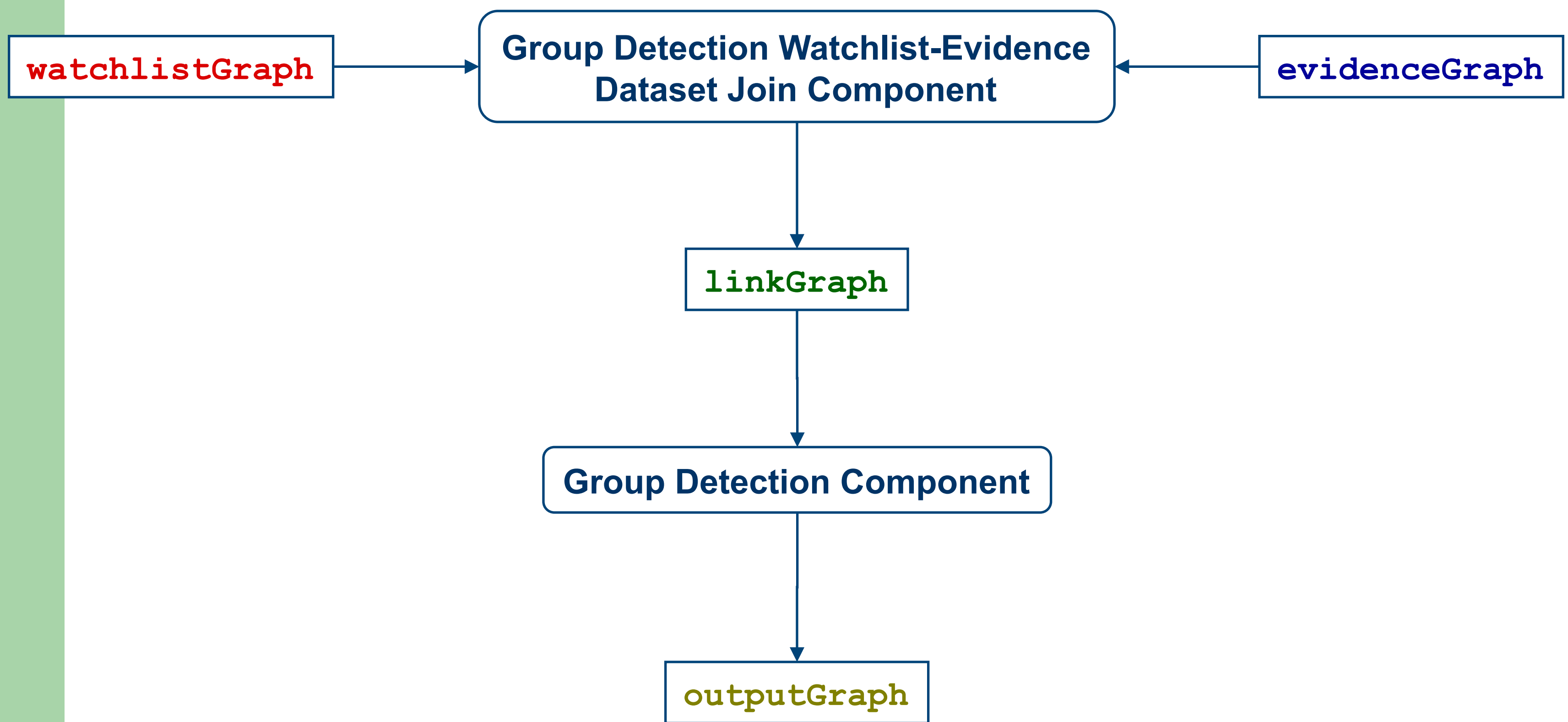


GITI's ToolKit supports three native component interface styles.

- **Automatic:** Delimited text files (implemented), XML files (notionally designed)
- **Semi-automatic:** Ntriples files
- **Custom** Lisp code

GITI
Global InfoTek, Inc.

Workflow Use Case



KB Query Component & Query Forms

```
(defKB-query-component
     group-detection-watchlist-evidence-dataset-join-component
     (DataJoinProcess)
  ((query (q- ?Event !rdf:type !teo:TwoWayCommunicationEvent ?evidenceGraph)
          (q- ?Event !teo:sender ?sender ?evidenceGraph)
          (q- ?Event !teo:receiver ?receiver ?evidenceGraph)
          (q- ?sender !rdf:type !teo:Person ?evidenceGraph)
          (q- ?receiver !rdf:type !teo:Person ?evidenceGraph)
          (q- ?sender !rdf:type !teo:Person ?watchlistGraph)
          (q- ?receiver !rdf:type !teo:Person ?watchlistGraph)
          (a- ?Event !rdf:type !teo:TwoWayCommunicationEvent ?linkGraph)
          (a- ?Event !teo:deliberateActor ?sender ?linkGraph)
          (a- ?Event !teo:deliberateActor ?receiver ?linkGraph)
          (a-- ?sender !rdf:type !teo:Person ?linkGraph)
          (a-- ?receiver !rdf:type !teo:Person ?linkGraph)))
```

- **q-** = query / find in graph.
- **a-** = assert / add to graph.
- **a--** = assert / add to graph (omitting any duplicate assertions).

→ *Forward chain from input graphs / datasets to output graphs / datasets.*

GDA Native Input and Output CSV Files

Native GDA Input:

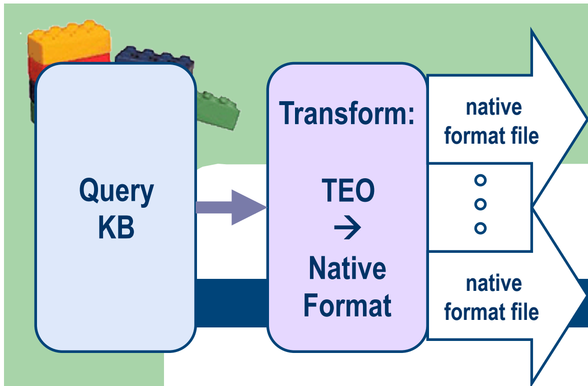
```
Ev-1194,In-10381
Ev-709,In-15840
Ev-709,In-36232
Ev-38749,In-4938
Ev-38749,In-48834
Ev-34121,In-3007
Ev-34121,In-35214
Ev-65474,In-21371
Ev-65474,In-19354
Ev-23484,In-39017
Ev-23484,In-16809
...
```

Native GDA Output:

```
group,entity
G0,In-10096
G0,In-15840
G0,In-19354
G0,In-19540
G0,In-19625
G0,In-21371
G0,In-28719
G0,In-37201
G0,In-37733
G0,In-38634
G0,In-47910
G1,In-1002
...
```

Automatic Interface

```
(defWrapped-component GDA-component-TerroristGroup (GroupDetectionProcess)
  :native-input-CSV-file-specs
    (("links.csv"
      :query (query (q- ?E !teo:deliberateActor ?P ?linkGraph))
      :query-type select
      :headerline nil
      :text-delimiter ","
      :query-template (?E ?P)))
  :native-output-CSV-file-specs
    (("groups.csv"
      :query (query (a- ?G !teo:orgMember ?P ?outputGraph)
                    (a-- ?G !rdf:type !teo:TerroristGroup ?outputGraph)
                    (a-- ?P !rdf:type !teo:Terrorist ?outputGraph))
      :headerline t
      :CSV-template (?G ?P)
      :namespace-template ("http://anchor/teo#" "http://anchor/teo#")))
  :native-component-directory "GDA_DISTRIBUTION"
  :native-component-command-name "gda_applic"
  :native-component-command-arguments ("links" "links.csv"))
```



Automatic Input Mechanism

Native GDA Input File:

```
Ev-1194,In-10381
Ev-709,In-15840
Ev-709,In-36232
Ev-38749,In-4938
Ev-38749,In-48834
Ev-34121,In-3007
Ev-34121,In-35214
Ev-65474,In-21371
Ev-65474,In-19354
Ev-23484,In-39017
Ev-23484,In-16809
...
```

General Query Conjunct:

```
(q- ?E              !teo:deliberateActor ?P              ?linkGraph)
```

Instantiated Query Conjunct:

```
(q- !teo:Ev-1194 !teo:deliberateActor !teo:In-10381 ?linkGraph)
```

General Query Template: `(?E ?P)`

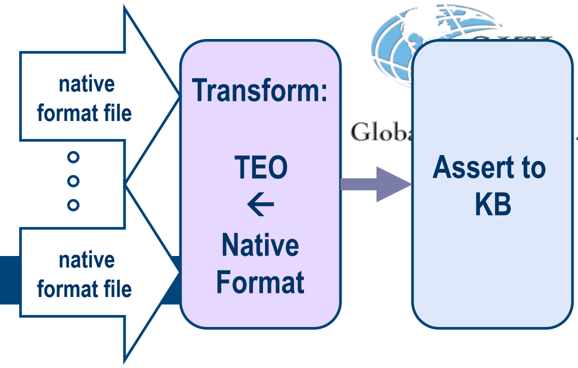
Instantiated Query Template: `(!teo:Ev-1194 !teo:In-10381)`

Calling the Native Component (Automatic Interface)

Directory:	Command-name:	Command-arguments:
`$GU_CORE/GDA_DISTRIBUTION`	`gda_applic`	`links links.csv`

Native Component

Automatic Output Mechanism



Query Conjuncts:

Gen. `(a- ?G !teo:orgMember ?P ?outputGraph)`

Inst. `(a- !teo:G0 !teo:orgMember !teo:In-10096 ?outputGraph)`

Gen. `(a-- ?G !rdf:type !teo:TerroristGroup ?outputGraph)`

Inst. `(a-- !teo:G0 !rdf:type !teo:TerroristGroup ?outputGraph)`

Gen. `(a-- ?P !rdf:type !teo:Terrorist ?outputGraph)`

Inst. `(a-- !teo:In-10096 !rdf:type !teo:Terrorist ?outputGraph)`

Native GDA Output File:

```
group,entity
G0,In-10096
G0,In-15840
G0,In-19354
G0,In-19540
G0,In-19625
G0,In-21371
G0,In-28719
G0,In-37201
G0,In-37733
G0,In-38634
G0,In-47910
G1,In-1002
...
```

General CSV / Query Template: `(?G ?P)`

Instantiated CSV Template: `(G0 In-10381)`

Instantiated Query Template: `(!teo:G0 !teo:In-10381)`

Semi-automatic Interface

```
(defWrapped-component GDA-component-TerroristGroup (GroupDetectionProcess)
  :declared-input-queries
    ((query (q- ?E !teo:deliberateActor ?P ?linkGraph)))
  :declared-output-queries
    ((query (a- ?G !teo:orgMember ?P ?outputGraph)
            (a-- ?G !rdf:type !teo:TerroristGroup ?outputGraph)
            (a-- ?P !rdf:type !teo:Terrorist ?outputGraph)))
  :native-component-directory "GDA"
  :native-component-command-name "ntriples-GDA-wrapper.sh"
  :native-component-command-arguments ())
```

Query KB

Global InfoTek, Inc.

Semi-automatic Input Mechanism

Query Conjunct:

```
(q- ?E                              !teo:deliberateActor
?P ?linkGraph)
```

Input Ntriples File (`./linkGraph`):

```
<http://anchor/teo#Ev-1194> <http://anchor/teo#deliberateActor>
<http://anchor/teo#In-10381> .

<http://anchor/teo#Ev-52532> <http://anchor/teo#deliberateActor>
<http://anchor/teo#In-37997> .

...
```

Calling the Native Component (Semi-automatic Interface)

Directory: `$GU_CORE/GDA`

Command-name: `ntriples-GDA-wrapper.sh`

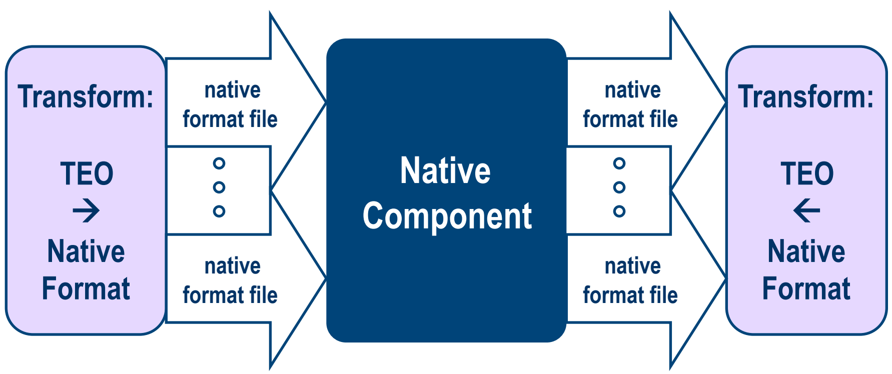
Input Ntriples File (`./linkGraph`):

Input CSV File (`./links.csv`):

Output Ntriples File (`./outputGraph`):

Output CSV File (`./groups.csv`):

```
$GU_CORE/GDA_DISTRIBUTION gda_applic links links.csv
```



Semi-automatic Output Mechanism

Assert to KB

Query Conjuncts:

```
(a- ?G !teo:orgMember
?P ?outputGraph)

(a-- ?G !rdf:type
!teo:TerroristGroup ?outputGraph)

(a-- ?P !rdf:type
!teo:Terrorist ?outputGraph)
```

Output Ntriples File (`./outputGraph`):

```
<http://anchor/teo#G0> <http://anchor/teo#orgMember>
<http://anchor/teo#In-10096> .

<http://anchor/teo#G0> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://anchor/teo#TerroristGroup> .

<http://anchor/teo#In-10096> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://anchor/teo#Terrorist> .

...
```

Custom Lisp Code Interface

```
(defWrapped-component GDA-component-TerroristGroup (GroupDetectionProcess)
  :declared-input-queries
    ((query (q- ?E !teo:deliberateActor ?P ?linkGraph)))
  :declared-output-queries
    ((query (a- ?G !teo:orgMember ?P ?outputGraph)
            (a-- ?G !rdf:type !teo:TerroristGroup ?outputGraph)
            (a-- ?P !rdf:type !teo:Terrorist ?outputGraph)))
  :inner-wrapper-body (GDA-component-multi-inner-wrapper-body
                       linkGraph
                       outputGraph
                       !teo:Terrorist
                       !teo:TerroristGroup
                       GDA-component-TerroristGroup))
```

Any Evaluable Lisp Expression

Calling the Native Component (Custom Interface)

Global InfoTek, Inc.

Any Evaluable Lisp Expression

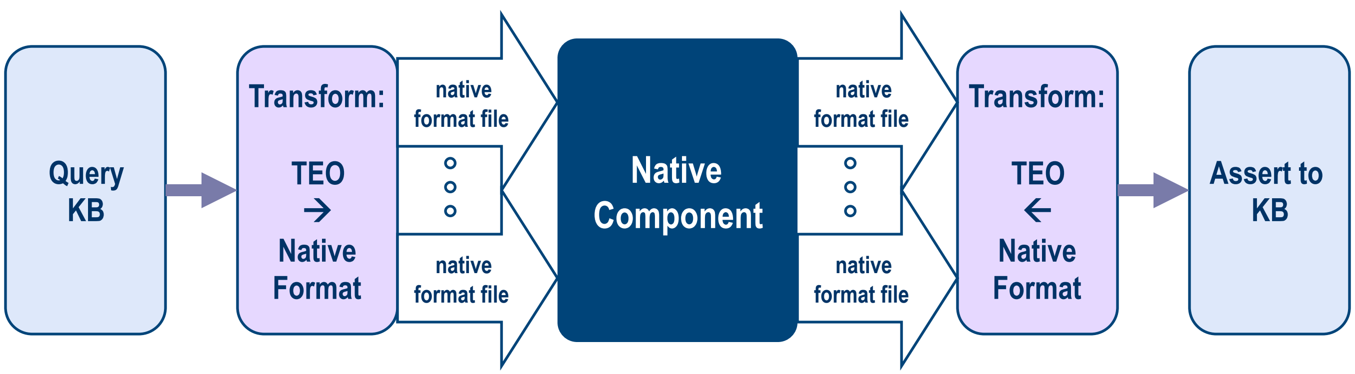
Input Graph (`?linkGraph`):

Input CSV File (`./links.csv`):

Output Graph (`?outputGraph`):

Output CSV File (`./groups.csv`):

```
$GU_CORE/GDA_DISTRIBUTION gda_applic links links.csv
```



GDA Native Input XML File

Native ORA Input:


```
<?xml version="1.0" standalone="yes">
<DynamicNetwork>
  <MetaMatrix>
    <nodes>
      <nodeset type="Agent" name="Agent" id="Agent">
        <node id="In-32764"></node>
        <node id="In-22466"></node>
        …
      </nodeset>
    </nodes>
    <networks>
      <graph sourceType="Agent" targetType="Agent" id="communication">
        <edge source="In-3741" target="In-35250" type="double" value="1"></edge>
        <edge source="In-47379" target="In-45163" type="double" value="1"></edge>
        …
      </graph>
    </networks>
  </MetaMatrix>
</DynamicNetwork>
```

Automatic XML File Spec (Notional)

```
<?xml version="1.0" standalone="yes">
<DynamicNetwork>
  <MetaMatrix>
    <nodes>
      <nodeset type="Agent" name="Agent" id="Agent">
        <TGU-query-spec>
          <query (query (q- ?E !teo:deliberateActor ?P ?linkGraph)) />
          <query-type select-distinct />
          <query-template (?P) />
          <answer-file-line-content "<node id=&quot;?P&quot;></node>" />
        </TGU-query-spec>
      </nodeset>
    </nodes>
    <networks>
      <graph sourceType="Agent" targetType="Agent" id="communication">
        <TGU-query-spec>
          <query (query (q- ?E !teo:deliberateActor ?P1 ?linkGraph)
                        (q- ?E !teo:deliberateActor ?P2 ?linkGraph)
                        (not (upi= ?P1 ?P2))) />
          <query-type select-distinct />
          <query-template (?P1 ?P2) />
          <answer-file-line-content "<edge source=&quot;?P1&quot; target=&quot;?P2&quot;
type=&quot;double&quot; value=&quot;1&quot;></edge> />
        </TGU-query-spec>
      </graph>
    </networks>
  </MetaMatrix>
</DynamicNetwork>
```

Under the Covers, Behind the Scenes

- Error handling and trapping
- Trace mode for debugging
- Automated regression testing
- System-wide logging
- Component characterization and registration
- Stressing of AllegroGraph's remote server implementation

Tangram Data & Component Limitations

- Structured, synthetic data
- Limited space of components
  - Group detectors (4)
  - Suspicion scorers (2)
  - Pattern matchers (2)

The Wrapping Process

Wrapping steps:

- Install the wrapping toolkit.
- Install the native component so that it will be accessible to the wrapper.
- Define any KB query component(s) needed to select appropriate data from any broader dataset(s).
- Define the wrapper for the native component.
- Test both KB query and wrapped native components to ensure effective operation. We have developed and applied a testing framework that includes component concurrency (i.e., re-entrance) testing.
- Deploy the developed and tested components.

Wrapping team:

- "Installer" (of legacy components)
- "Developer" (toolkit user)
- "Tester" (wrapped component QA)
- "Scripters" (custom wrapping code)
- "Deployer" (of wrapped components

- Component "champion" …
  - Knows component's enterprise function (s)
  - Understands component operation
  - Brings exemplary use cases

- Toolkit developers (receive new requirements)

The Tangram GU Story

- Developed the toolkit during roughly six months of concentrated effort
  - Started with this presentation’s use case workflow
  - Developed progressively more automatic interfaces
  - Wrapped legacy components ourselves
  - Provided the toolkit to others
  - Wrapped components: GDA, ORA group detection algorithms, suspicion scorers based on Proximity and NetKit classifiers, LAW and CADRE pattern matchers
- Met Tangram’s usability goals
  - With the toolkit’s fully automatic interface, we can usually complete Steps 3 and 4 of the foregoing wrapping process within a single staff hour.

Representing a Dataset's Context Lineage

- We take each workflow component's execution, noted in a ProcessExecution (PE) object, as the source of the statements in any output (hypothesis) dataset.
- Lineage is manifested in the connections among datasets, process executions, and workflow executions (noted in WorkflowExecution objects).
- Incremental context representation: Upstream datasets' statements also hold in downstream datasets.

<u>WorkflowExecution</u>
hasProcessExecution*

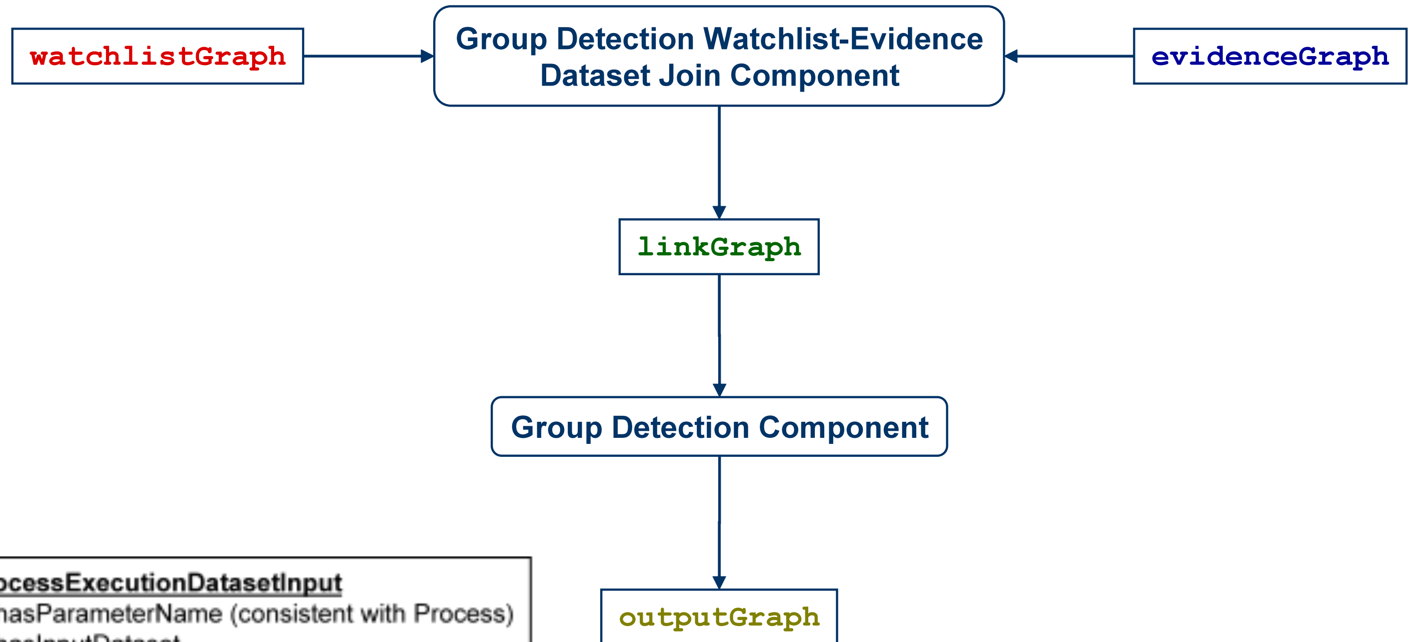
<u>ProcessExecution</u>
hasProcess (e.g., GDA)
hasPEDatasetInput*
hasPEDatasetOutput*
hasPEControlInput*

<u>ProcessExecutionDatasetInput</u>
hasParameterName (consistent with Process)
hasInputDataset

<u>ProcessExecutionDatasetOutput</u>
hasParameterName (consistent with Process)
hasOutputDataset

<u>ProcessExecutionControlInput</u>
hasParameterName
hasValue

Workflow Use Case (Reminder)



WorkflowExecution
hasProcessExecution*

ProcessExecution
hasProcess (e.g., GDA)
hasPEDatasetInput*
hasPEDatasetOutput*
hasPEControlInput*

ProcessExecutionDatasetInput
hasParameterName (consistent with Process)
hasInputDataset

ProcessExecutionDatasetOutput
hasParameterName (consistent with Process)
hasOutputDataset

ProcessExecutionControlInput
hasParameterName
hasValue

Relaxing the Context Monotonicity Assumption

- Current implicit assumption:
  - A component's output graph(s) only add(s), logically, to the information in its input graph(s), never delete(s) or retract(s).
  - Not entirely practical in intelligence analysis…
    - → Different analysts pursue different lines of reasoning, using different tools, at different times
    - → Build on each other's results / hypotheses
    - → Sometimes appropriate to extend a context, sometimes to branch

Some Reasons Different Contexts May Arise

Differences in supporting data, from:

- Conflicting original data sources.
- Time-varying data conditions for a given source, such as:
  - Disbelief in something we earlier had belief in (perhaps because it had been supplied in error)
  - Belief in something we did not have belief in (perhaps because we had no data about it)

Differences in supporting analytical hypotheses, from:

- Analyst's conjecture, or "what-if" analysis (that may effect belief or disbelief in data as discussed above)
- Differences in workflow components giving rise to different answers, when:
  - A given workflow function has alternative realizations in different components.
  - A given component has alternative configurations of control parameters.

Global InfoTek, Inc.

Beyond Tangram

Our workflow component semantic interoperability solution can stand on its own, apart from much of *Tangram's more ambitious surrounding machinery.*

- *Automatically select the "best" component in a given component class, based on:*
  - *Data profiling*
  - *Component execution profiling*
  - *Machine learning-based performance prediction*
- *Execute workflows on a massive scale using grid computing.*

→ *Surrounding machinery's constraints have sometimes limited our development opportunities.*

GITI
Global InfoTek, Inc.

Workflow Editor Opportunity

watchlistGraph → Group Detection Watchlist-Evidence Dataset Join Component ← evidenceGraph

Group Detection Watchlist-Evidence Dataset Join Component → linkGraph → Group Detection Component → outputGraph

- *Analyze queries to determine dataset-wise semantic compatibility of potentially connected components.*

GITI analyzes queries to support workflow reasoning now, but in Tangram we have not had a workflow composition role.

Query Editor Opportunity

```
(defKB-query-component
     group-detection-watchlist-evidence-dataset-join-component
     (DataJoinProcess)
  ((query (q- ?Event !rdf:type !teo:TwoWayCommunicationEvent ?evidenceGraph)
          (q- ?Event !teo:sender ?sender ?evidenceGraph)
          (q- ?Event !teo:receiver ?receiver ?evidenceGraph)
          (q- ?sender !rdf:type !teo:Person ?evidenceGraph)
          (q- ?receiver !rdf:type !teo:Person ?evidenceGraph)
          (q- ?sender !rdf:type !teo:Person ?watchlistGraph)
          (q- ?receiver !rdf:type !teo:Person ?watchlistGraph)
          (a- ?Event !rdf:type !teo:TwoWayCommunicationEvent ?linkGraph)
          (a- ?Event !teo:deliberateActor ?sender ?linkGraph)
          (a- ?Event !teo:deliberateActor ?receiver ?linkGraph)
          (a-- ?sender !rdf:type !teo:Person ?linkGraph)
          (a-- ?receiver !rdf:type !teo:Person ?linkGraph)))
```

GUI to compose queries:

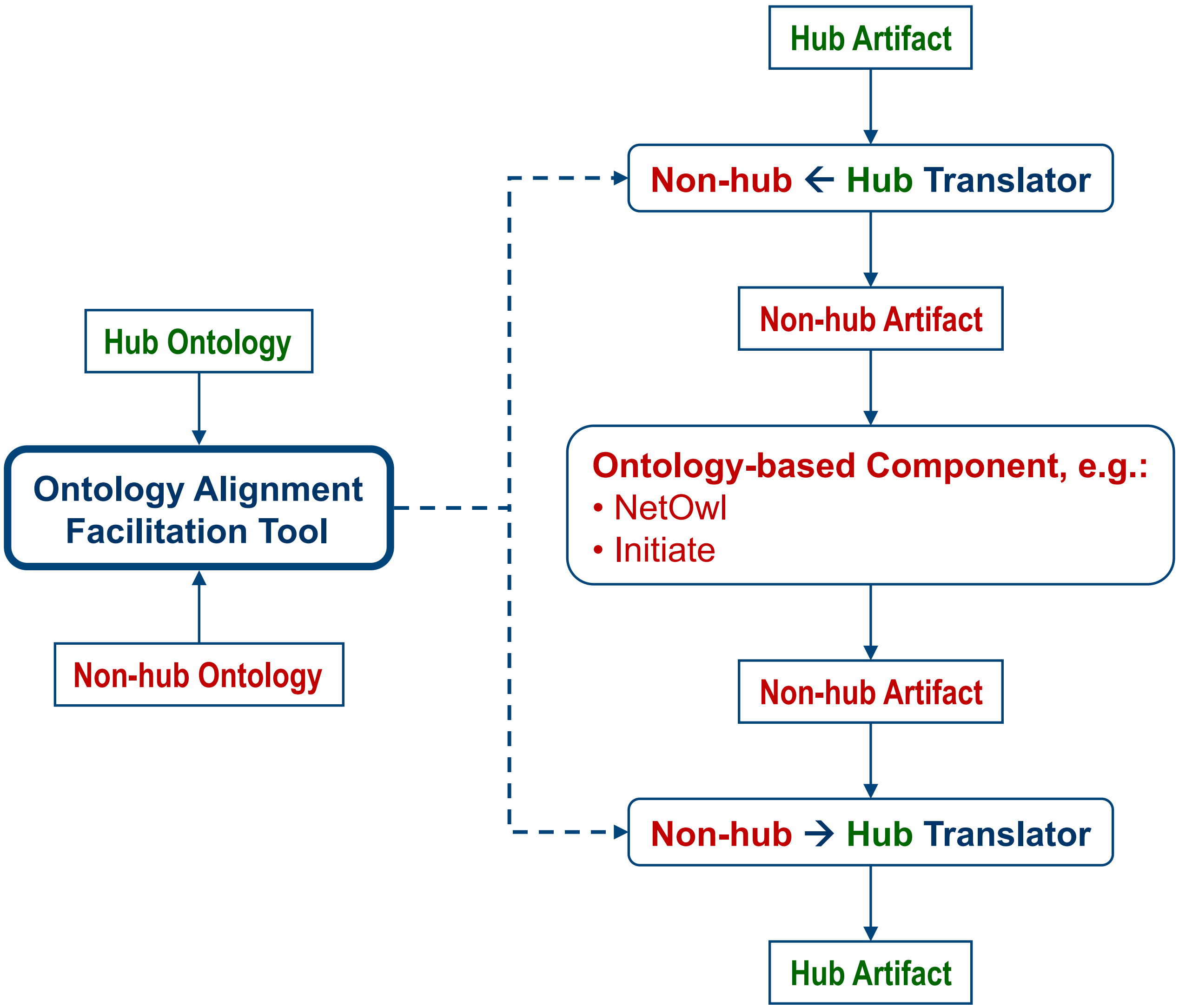
- *Ontology class / subclass browsing interface*
- *Graphical depiction of query structure*
- *Constraints from declared adjacent components, dataset connections*

Component Editor Opportunity

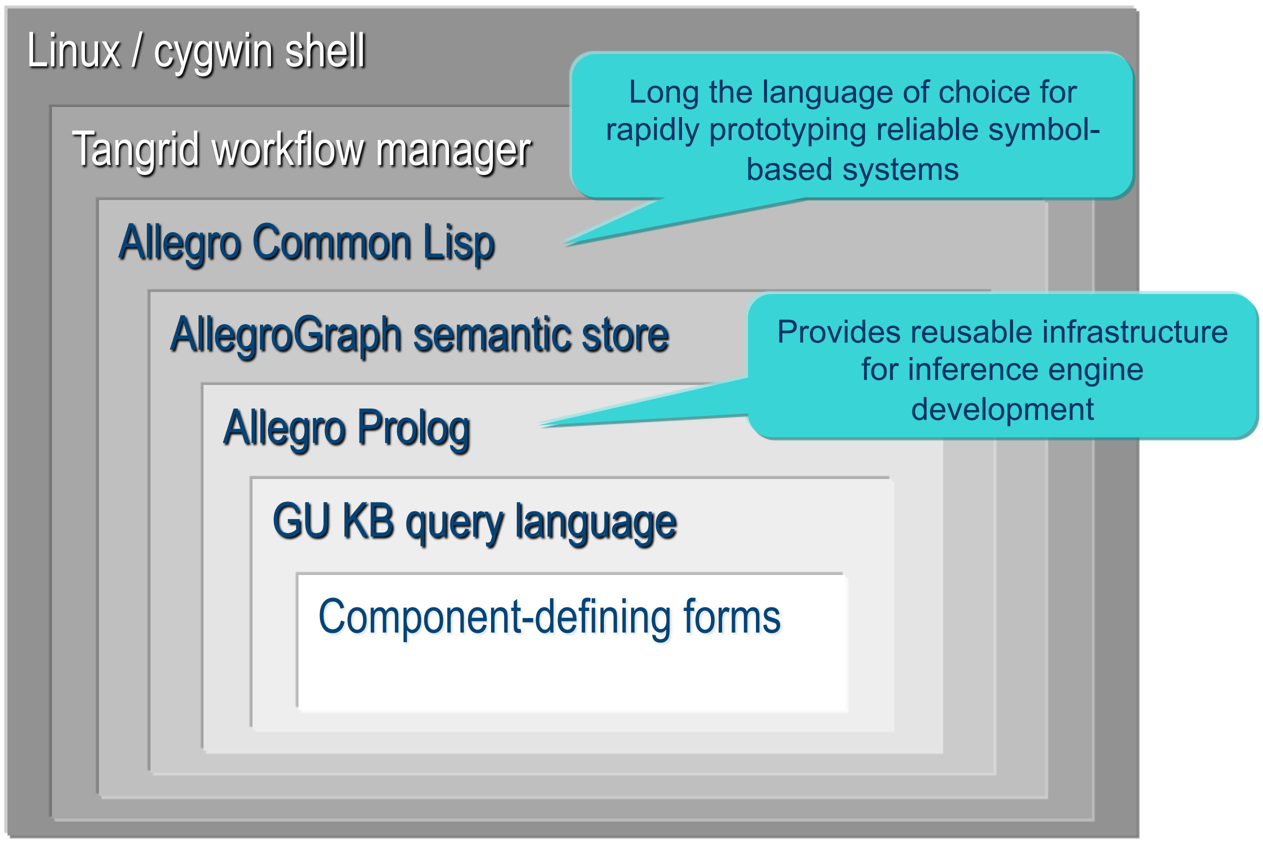
```
(defWrapped-component GDA-component-TerroristGroup (GroupDetectionProcess)
  :declared-input-queries
    ((query (q- ?E !teo:deliberateActor ?P ?linkGraph)))
  :declared-output-queries
    ((query (a- ?G !teo:orgMember ?P ?outputGraph)
            (a-- ?G !rdf:type !teo:TerroristGroup ?outputGraph)
            (a-- ?P !rdf:type !teo:Terrorist ?outputGraph)))
  :native-component-directory "GDA"
  :native-component-command-name "ntriples-GDA-wrapper.sh"
  :native-component-command-arguments ())
```

Forms-based GUI to define components, for those who definitely never want to touch anything that even looks like (Lisp) code.

Ontology Alignment Facilitation Opportunity

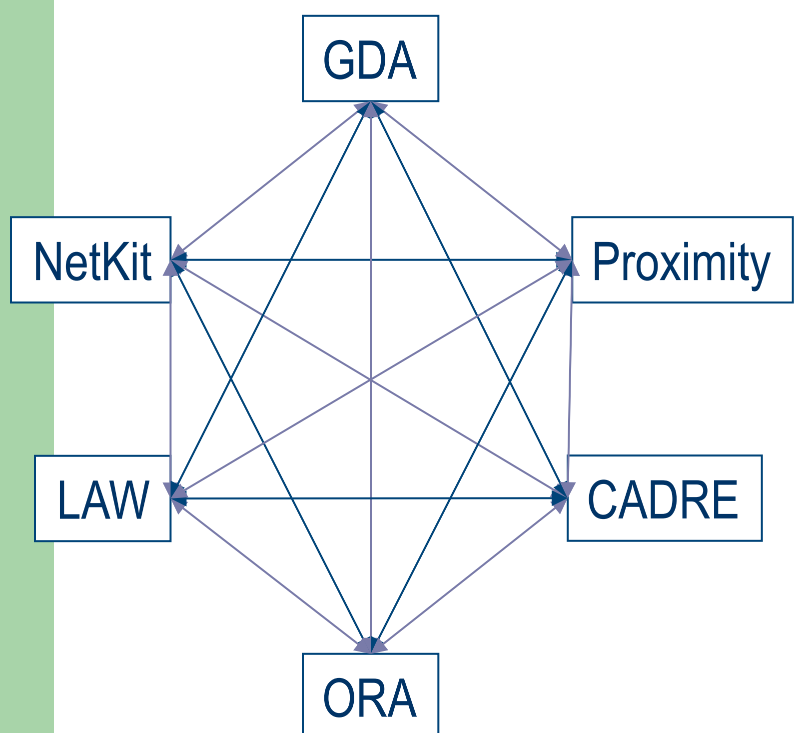


ToolKit Implementation, in System Context



Pairwise Adapters vs. “Hub” Language

$n\,(n-1) = 30$ unidirectional adapters for 6 components



$2n = 12$ unidirectional adapters for 6 components