Semantic Discovery & Integration of Urban Data Streams

Feng Gao, Ali Intizar and Alessandra Mileo
Smart City Applications - Overview

- Emergency & Disaster Management
- Inter-Agency Collaboration
- Critical Infrastructure Management
- Information Management
- Citizen Services & Immigration Control
- Law Enforcement
- Public Administration Services

* http://www.nec.com
Smart City Applications - IoE

Internet of Things
(assets, devices, mobile and static sensors)

Internet of People
(social networking, crowd-sourcing, workflow)

Internet of Services
(cloud-based solutions, processes, tools and operations)

Internet of Data
(Linking of Open Data)
Smart City Applications - Infrastructure

- Physical Sensors
Smart City Applications - Infrastructure

- Mobile/Wearable Sensors

[Images of wearable sensors and a smartphone with various sensors connected.]
Smart City Applications - Infrastructure

- Virtual Sensors (Social Media)
Smart City Applications - City Pulse

Virtualisation

Large-Scale Data Analysis

Applications

Internet of People

Internet of Things

Knowledge Based

Reliability Testing

API

BRASOV
Smart City Applications - Challenges

Federation of Heterogeneous Data Streams

Applications

API

Large-Scale Analysis

Knowledge Based

Reliability Testing
Smart City Applications - Challenges

- Real-time Information Extraction and Event Detection
- Virtualisation
- Large-Scale Data Analysis
- Knowledge Based
- Reliability Testing
- Internet of People
- Internet of Things
- Applications

- API
Smart City Applications - Challenges

Virtualisation

Internet of People

Internet of Things

Knowledge Based

Reliability Testing

Reliable Information Processing

Large-Scale Data Analysis

Applications

API
Smart City Applications - Challenges

Federation of Heterogeneous Data Streams

Internet of People

Internet of Things

Large-Scale Analysis

Knowledge Based

Reliability Testing

Applications

API

19/10/2014
Smart City Applications - Challenges

- Virtualisation

- Federation of heterogeneous data streams

- Processing user’s queries in terms of requirements rather than hard-bind queries

- Optimal data source selection while taking users constraints and preferences into account

- Automated composition of primitive data services/streams into complex events

- Automated generations of queries from the complex event’s composition plan
ACEIS - Features

- **Automated Complex Event Implementation System**
- Enables users to provide requirements rather than hard bind streams
- Automatically discovers the relevant data streams
- Selects optimal data stream after evaluating user’s constraints and preferences
- On demand data federation using complex event patterns
- Transformation of complex event into stream queries
The Complex Event Service Ontology (CES ontology) is an extension of OWL-S ontology.

CES ontology is used together with SSN (Semantic Sensor Network) ontology, SSN is used to describe the sensor aspects.

An Event Service is described with a *Grounding* and an *Event Profile*.

1. Groundings specify how to access and interact with event services.
2. Event Profiles describe the events provided by the services with *Patterns* and *Non-Functional Properties* (NFP).

An Event Request is specified as an incomplete Event Service description, without concrete service bindings.
Complex Event Service Ontology – Overview (2/2)

Namespaces:
- default: <http://www.insight-centre.org/ces#>
- rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- owls: <http://www.daml.org/services/owl-s/1.2/Service.owl#>
- owls-sp: <http://www.daml.org/services/owl-s/1.2/ServiceParameter.owl#>

Legend:
- Class
- Object property
- subClassOf
- Data property
Complex Event Service Ontology – Event Pattern

Namespaces:
- default: <http://www.insight-centre.org/ces#>
- rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- owls: <http://www.daml.org/services/owl-s/1.2/Service.owl#>

Legend:
- Class
- Class (unimplemented)
- Object property
- subClassOf
A sensor service description is annotated as:

\[ s_{\text{desc}} = (t_d, g, q_d, P_d, \text{Fol}_d, f_d) \]

- **type**
- **grounding**
- **QoS**
- **Observed Properties**
- **Feature Of Interest**

Listing 1. Traffic sensor service description:

```turtle
:sampleTrafficSensor a ssn:Sensor, ces:PrimitiveEventService;
  owls:presents :sampleProfile ;
  owls:supports :sampleGrounding ;
  ssn:observes [ a ces:AverageSpeed ;
    ssn:isPropertyFor :Seg_1] ,
  [ a ces:VehicleCount ;
    ssn:isPropertyFor :Seg_1] ,
  [ a ces:EstimatedTime ;
    ssn:isPropertyFor :Seg_1] .
:sampleProfile a ces:EventProfile ;
  owls:serviceCategory [ a ces:TrafficReportService ;
    owls:serviceCategoryName "traffic_report"^^xsd:string].
```
ACEIS - Architecture

Knowledge Base
- QoI/QoS
- Stream Description
- Data Mgmt, Indexing, Caching

Application Interface
- User Input
- Event Request

ACEIS Core
- Resource Management
  - Resource Discovery
  - Event Service Composer
- Composition Plan
- Adaptation Manager

Data Federation
- Query
- Results
- Subscription Manager
- Query Transformer
- Query Engine

Constraint Violation

Semantic Annotation

Data Store

IoT Data Stream

Social Data Stream
Similarly, a sensor service request is annotated:

\[ s_r = (t_r, P_r, Fol_r, f_r, \text{pref}, C) \]

A sensor service description is denoted as

\[ s_{\text{desc}} = (t, g, q_d, P_d, Fol_d, f_d), \]

where \( t \) is the sensor event type, \( g \) is the service grounding, \( q_d \) is a QoS vector describing the QoS values, \( P_d \) is the set of \( \text{ObservedProperties} \), \( Fol_d \) is the set of \( \text{FeatureOfInterests} \) and \( f_d: P_d \rightarrow Fol_d \) is a function correlating observed properties with their feature-of-interests. Similarly, a sensor service request is denoted

\[ s_{\text{sr}} = (t_r, q_r, P_r, Fol_r, f_r, \text{pref}, C). \]

Compared to \( s_{\text{desc}} \), \( s_{\text{sr}} \) do not specify service groundings, \( q_r \) represents the constraints over QoS metrics, \( \text{pref} \) represents the QoS weight vector specifying users' preferences on QoS metrics and \( C \) is a set of functional constraints on the values of \( P_r \).

\[ :\text{sampleRequest} \ a \ \text{ssn:Sensor, ces:EventRequest}; \]
\[ \ \text{owsls:presents} \ :\text{requestProfile} ; \]
\[ \ \text{ssn:observes} \ [ \ a \ \text{ces:EstimatedTime}; \]
\[ \ \text{ssn:isPropertyFor} \ :\text{Seg}_1] ; \]
\[ \ \text{ces:hasConstraint} \ [ \ \text{rdf:type} \ \text{ces:NFPConstraint}; \]
\[ \ \text{ces:onProperty} \ \text{ces:Availability}; \]
\[ \ \text{ces:hasExpression} \]
\[ \ [ \ \text{emvo:greaterThan} \ "0.9"^{\text{xsd:double}}], \]
\[ \ [ \ \text{rdf:type} \ \text{ces:NFPConstraint}; \]
\[ \ \text{ces:onProperty} \ \text{ces:Accuracy}; \]
\[ \ \text{ces:hasExpression} \]
\[ \ [ \ \text{emvo:greaterThan} \ "0.9"^{\text{xsd:double}}]. \]

\[ :\text{requestProfile} \ a \ \text{ces:EventProfile}; \]
\[ \ \text{owsls:serviceCategory} \ [ \ a \ \text{ces:TrafficReportService}; \]
\[ \ \text{owsls:serviceCategoryName} \ "\text{traffic_report}"^{\text{xsd:string}}]. \]
Similarly, a sensor service request is annotated:

\[ s_r = (t_r, P_r, \text{FoI}_r, f_r, \text{pref}, C) \]

A sensor service description is denoted as

\[ s_{\text{desc}} = (t_d, g, q_d, P_d, \text{FoI}_d, f_d) \]

where \( t \) is the sensor event type, \( g \) is the service grounding, \( q_d \) is a QoS vector describing the QoS values, \( P_d \) is the set of \( \text{ObservedProperties} \), \( \text{FoI}_d \) is the set of \( \text{FeatureOfInterests} \) and \( f_d: P_d \to \text{FoI}_d \) is a function correlating observed properties with their feature-of-interests.

Similarly, a sensor service request is denoted

\[ s_r = (t_r, q_r, P_r, \text{FoI}_r, f_r, \text{pref}, C) \]

Compared to \( s_{\text{desc}} \), \( s_r \) do not specify service groundings, \( q_r \) represents the constraints over QoS metrics, \( \text{pref} \) represents the QoS weight vector specifying users' preferences on QoS metrics and \( C \) is a set of functional constraints on the values of \( P_r \).

A sensor service description is considered a match for \( s_r \) if all of the following three conditions are true:
Stream Discovery – Matching Condition

A sensor service description $S_d$ matches a service request $S_r$ *iff* the following three conditions are true:

1. $t_r$ subsumes $t_d$:

2. $q_d$ satisfies $C$:

3. $\forall p_1 \in P_r, \exists p_2 \in P_d \Rightarrow T(p_1)$ subsumes $T(p_2) \land f_r(p_1) = f_d(p_2)$:
Stream Discovery – Matching Condition

A sensor service description $S_d$ matches a service request $S_r$ *iff* the following three conditions are true:

1. $t_r$ subsumes $t_d$:
   Requested service type is same or a super-type of provided service type.

2. $q_d$ satifies $C$:
   Quality-of-service properties of the provided sensor service satisfy the constraints specified in the service request.

3. $\forall p_1 \in P_r, \exists p_2 \in P_d \Rightarrow T(p_1)$ subsumes $T(p_2) \land f_r(p_1) = f_d(p_2)$:
   Provided sensor service observes all requested physical properties from the requested feature-of-interest (a geographical location or physical entity from which the observations are made).
Stream Integration – Complex Event Service

- A Complex Event Service (CES) integrates different sensor streams to detect complex events based on event patterns.

- An Event Pattern describes the correlations of integrated streams.

```turtle
:SampleEventRequest a ces:EventRequest;  
   owls:presents :SampleEventProfile.

:SampleEventProfile rdf:type owls:EventProfile;  
   ces:hasPattern [ rdf:type ces:And, rdf:Bag;  
      rdf:_1 :locationRequest;  
      rdf:_2 :seg1CongestionRequest;  
      rdf:_3 :seg2CongestionRequest;  
      rdf:_4 :seg3CongestionRequest;  
      ces:hasWindow "5"^^xsd:integer].
```
Stream Integration – Matching condition

Discovery and composition of complex event services are based on matching event patterns (and aggregated NFP values).

Procedure:
1. Derive canonical forms of event patterns of CESs.
2. Apply tree isomorphism algorithms over the canonical event patterns and the requested event pattern to identify reusable or equivalent event patterns.
3. Generate all possible composition plans.
4. Aggregate NFPs based on event patterns and compare aggregated NFP values against requested constraints to filter out unsatisfied composition plans.
5. Rank the remaining composition plans based on preferences (soft constraints).
Canonical Event Pattern (1/2)

Create complete event patterns

```
<table>
<thead>
<tr>
<th>ES1</th>
<th>Or</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES2</td>
<td></td>
</tr>
<tr>
<td>ES3</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>ES2</th>
<th>And</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td></td>
</tr>
<tr>
<td>e2</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>ES3</th>
<th>Seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>e3</td>
<td></td>
</tr>
<tr>
<td>e4</td>
<td></td>
</tr>
</tbody>
</table>

```

getCompletePattern()
Canonical Event Pattern (2/2)

Create reduced event patterns

- **Sequential Lift**
  - Example: 
    - Original: `SEQ SEQ SEQ e1 e2 e2 e3`
    - Lifted: `SEQ e1 e2 e2 e3`

- **Sequential Merge**
  - Example: 
    - Original: `SEQ e1 e2 e3`
    - Merged: `SEQ e1 e2 e3`

- **Parallel Lift**
  - Example: 
    - Original: `AND e1 AND e1 e2`
    - Lifted: `AND e1 e1 e2`

- **Parallel Merge**
  - Example: 
    - Original: `AND e1 e1 e2`
    - Merged: `AND e1 e2`

- **Repetition Lift**
  - Example: 
    - Original: `REP x2 e1 REP x3 e2`
    - Lifted: `REP x2 e1 REP x2 e2`

- **Repetition Merge**
  - Example: 
    - Original: `REP x2 e1 REP x2 e2`
    - Merged: `REP x2 e1 e2`
Create event reusability hierarchy

Reusable relation: $R(ep_1, ep_2)$ holds if $R_d(ep_1, ep_2)$ or $R_i(ep_1, ep_2)$ holds.
Example of a composition plan:

**Stream Integration – Composition Plan**

- Query
  - OR
    - SEQ
      - e1
      - e2
      - e3

- Event Service 1
  - type = e1
  - loc = loc1
- Event Service 2
  - type = e2
  - loc = loc2
- Event Service 3
  - type = e3
  - loc = loc3
- Event Service 4
  - type = e4
  - loc = loc4

Composition Plan
- OR
  - e4
  - e3
  - loc = loc4
  - loc = loc3
ACEIS - Architecture

Knowledge Base
- QoI/QoS
- Stream Description
- Data Mgmt, Indexing, Caching

Application Interface
- User Input
- Event Request

ACEIS Core
- Resource Discovery
- Event Service Composer

Resource Management

Data Federation
- Query
- Results

Query
- Subscription Manager
- Query Transformer
- Query Engine

Constraint Validation
- Constraint Violation

Adaptation Manager

Semantic Annotation

Data Store
- IoT Data Stream
- Social Data Stream
Query Transformation – Semantic Alignment

Goal: transform the composition plan into a stream query which can be evaluated by a stream reasoning engine over RDF data streams

Requirements:
- Alignments of event pattern operators to stream query operators
- Transformation Algorithm

Alignments for CQELS query language:

<table>
<thead>
<tr>
<th>Event Pattern</th>
<th>$s_d$</th>
<th>Sequence</th>
<th>Repetition</th>
<th>And</th>
<th>Or</th>
<th>Selection</th>
<th>Filter</th>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQELS Operator</td>
<td>SGP</td>
<td>-</td>
<td>-</td>
<td>Join</td>
<td>Optional</td>
<td>Projection</td>
<td>Filter</td>
<td>Window</td>
</tr>
</tbody>
</table>

- Sequence and Repetition not supported by CQELS.
- Sensor requests mapped to Stream Graph Pattern.
- AND operator mapped to stream join.
- OR operator mapped to OPTIONAL keyword (left-outer-join).
Example of composition plan

Event textual description:

Monitor the user’s current location as well as the traffic conditions (estimated travel time) of all the 3 street segments on the chosen route.

CQELS Query Transformation Result:

```
Select ... Where {
Graph <http://purl.oclc.org/NET/ssnx/ssn#> {
?ob rdfs:subClassOf ssn:Observation
Stream <locationStreamURL> [range 5s] {
?locId ssn:observationResult ?result1.
?result1 ssn:hasValue ?value1.
?loc ct:hasLongitude ?lon. }
Stream <trafficStreamURL1> [range 5s] {
?seg1Id rdf:type ?ob. ?seg1Id ssn:observedBy ?es1.
?seg1Id ssn:observationResult ?result2.
?result2 ssn:hasValue ?value2.
?value2 ssn:hasQuantityValue ?eta1.}
Stream <trafficStreamURL2> [range 5s] {...}
Stream <trafficStreamURL3> [range 5s] {...} }
```
Conclusion

• Automated Complext Event Implementation System

• Information model for dynamic discovery and selection of sensor data streams i.e. Complex Event Ontology, Event Pattern Ontology and Event Reuest/Profile

• Algorithm to create optimal composition plan using event reusability heirachy

• Transformation of the composition plan into stream queries (CQELS)