Bounds: Expressing Reservations about Incoming Data

Martin G. Skjæveland    Audun Stolpe

Presented by Kjetil Kjernsmo

COLD 2013
22 October 2013
• Protect a dataset from unwanted amendments
### Synopsis

- Protect a dataset from unwanted amendments
- Don’t mess (literally) with my dataset, please.
• Protect a dataset from unwanted amendments
• Don’t mess (literally) with my dataset, please.
• *Boundz*: Vocabulary for expressing reservations about incoming data
Synopsis

- Protect a dataset from unwanted amendments
- Don’t mess (literally) with my dataset, please.
- **Boundz**: Vocabulary for expressing reservations about incoming data
- Theory: Based on *bounded homomorphisms*
When do you want to protect a dataset from unwanted amendments?
When do you want to protect a dataset from unwanted amendments?

- Suppose you have a triple store, and
When do you want to protect a dataset from unwanted amendments?

- Suppose you have a triple store, and
- you’d like it to be community-curated
When do you want to protect a dataset from unwanted amendments?

- Suppose you have a triple store, and
- you’d like it to be community-curated
- i.e., you’d like to invite people and institutions to contribute data
Motivation

When do you want to protect a dataset from unwanted amendments?

- Suppose you have a triple store, and
- you’d like it to be community-curated
- i.e., you’d like to invite people and institutions to contribute data
- but you’d like to protect certain elements of structure
What structure would you want to protect?

- the ontology axioms in your dataset
- you decide the ontology, they contribute the data
- a particular vocabulary or ontology
- the core ontology or internal bookkeeping vocabulary should be stable
- a set of facts
- metadata about the dataset is your business only
- whole parts of the dataset or data in a particular namespace
- don’t touch the finished/perfect/imported parts
What structure would you want to protect?

It could be many things, e.g.,

• the ontology axioms in your dataset
• you decide the ontology, they contribute the data
• a particular vocabulary or ontology
• the core ontology or internal bookkeeping vocabulary should be stable
• a set of facts
• metadata about the dataset is your business only
• whole parts of the dataset or data in a particular namespace
• don’t touch the finished/perfect/imported parts
What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable
What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable
- a set of facts
What structure?

What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable
- a set of facts
  - metadata about the dataset is your business only
What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable
- a set of facts
  - metadata about the dataset is your business only
- whole parts of the dataset or data in a particular namespace
What structure would you want to protect?

It could be many things, e.g.,

- the ontology axioms in your dataset
  - you decide the ontology, they contribute the data
- a particular vocabulary or ontology
  - the core ontology or internal bookkeeping vocabulary should be stable
- a set of facts
  - metadata about the dataset is your business only
- whole parts of the dataset or data in a particular namespace
  - don’t touch the finished/perfect/imported parts
Examples

- Ontology hijacking:
  
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says `dcterms:subject rdfs:subClassOf ex:topic`
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected

- Breach of contract:
  
  - say your triple store advertises a particular vocabulary, i.e., API,
  - used to drive faceted browsing or search
  - then incoming data interferes with it
  - which affects the uses of the API
Examples

- Ontology hijacking:
  - say your ontology uses the dcterms:subject property
Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says
    
    ```
    dcterms:subject rdfs:subClassOf ex:topic
    ```

- Breach of contract:
  - say your triple store advertises a particular vocabulary, i.e., API,
  - used to drive faceted browsing or search
  - then incoming data interferes with it
  - which affects the uses of the API
Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says
    ```
    dcterms:subject rdfs:subClassOf ex:topic
    ```
  - as a result the number of statements that are inferred will increase
Examples

- **Ontology hijacking:**
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says
    
    \[
    \text{dcterms:subject} \quad \text{rdfs:subClassOf} \quad \text{ex:topic}
    \]
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected
Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says
    
    `dcterms:subject rdfs:subClassOf ex:topic`
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected

- Breach of contract:
Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says
    
    \[
    \text{dcterms:subject \ rdfs:subClassOf \ ex:topic}
    \]
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected

- Breach of contract:
  - say your triple store advertises a particular vocabulary, i.e., API,
Examples

- **Ontology hijacking:**
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says
    
    $\text{dcterms:subject rdfs:subClassOf ex:topic}$
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected

- **Breach of contract:**
  - say your triple store advertises a particular vocabulary, i.e., API,
  - used to drive faceted browsing or search
Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says
    
    \[\text{dcterms:subject} \ rdfs:subClassOf \ \text{ex:topic}\]
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected

- Breach of contract:
  - say your triple store advertises a particular vocabulary, i.e., API,
  - used to drive faceted browsing or search
  - then incoming data interferes with it
Examples

- Ontology hijacking:
  - say your ontology uses the `dcterms:subject` property
  - then incoming data says
    
    \[
    \text{dcterms:subject} \ rdfs:\text{subClassOf} \ \text{ex:topic}
    \]
  - as a result the number of statements that are inferred will increase
  - entailments over unrelated data will be affected

- Breach of contract:
  - say your triple store advertises a particular vocabulary, i.e., API,
  - used to drive faceted browsing or search
  - then incoming data interferes with it
  - which affects the uses of the API
Examples cont.

- Skewing content:
Skewing content:

- say your triple store contains election results
Examples cont.

- Skewing content:
  - say your triple store contains election results
  - à la data.gov.uk and openelectiondata.org
Skewing content:

- say your triple store contains election results
- à la data.gov.uk and openelectiondata.org
- then incoming data makes erroneous claims about existing elections
Skewing content:

- say your triple store contains election results
- à la data.gov.uk and openelectiondata.org
- then incoming data makes erroneous claims about existing elections
- resulting in noisy data
Skewing content:

- say your triple store contains election results
- à la data.gov.uk and openelectiondata.org
- then incoming data makes erroneous claims about existing elections
- resulting in noisy data
- lowering overall data quality and re-usability
Examples cont.

- Skewing content:
  - say your triple store contains election results
  - à la data.gov.uk and openelectiondata.org
  - then incoming data makes erroneous claims about existing elections
  - resulting in noisy data
  - lowering overall data quality and re-usability
  - breaks trust
What is Boundz?

- A vocabulary for expressing reservations about incoming data

Vocabulary elements:
- Fine-grained restrictions on types and predicates with Boundzs
- Exceptions, i.e., what to do in case of a violation
  - abort
  - accept parts of the incoming data
  - ignore only violating triples

Data exchange, Payloads, Validation results

See [http://sws.ifi.uio.no/vocab/boundz](http://sws.ifi.uio.no/vocab/boundz) for details

Publish, re-use and combine bounds
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*

- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with Bounds
  - Exceptions, i.e., what to do in case of a violation
  - abort
  - accept parts of the incoming data
  - ignore only violating triples

- Also Data exchange, Payloads, Validation results

- See http://sws.ifi.uio.no/vocab/boundz for details
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:

See http://sws.ifi.uio.no/vocab/boundz for details

- Publish, re-use and combine bounds
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with Bounds

See [http://sws.ifi.uio.no/vocab/boundz](http://sws.ifi.uio.no/vocab/boundz) for details
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with Bounds
  - Exceptions, i.e., what to do in case of a violation

See [http://sws.ifi.uio.no/vocab/boundz](http://sws.ifi.uio.no/vocab/boundz) for details

Publish, re-use and combine bounds
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with Bounds
  - Exceptions, i.e., what to do in case of a violation
    - abort

See [http://sws.ifi.uio.no/vocab/boundz](http://sws.ifi.uio.no/vocab/boundz) for details

Publish, re-use and combine bounds
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with Bounds
  - Exceptions, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data

See http://sws.ifi.uio.no/vocab/boundz for details

Publish, re-use and combine bounds
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with Bounds
  - Exceptions, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data
    - ignore only violating triples

See [http://sws.ifi.uio.no/vocab/boundz](http://sws.ifi.uio.no/vocab/boundz) for details

Publish, re-use and combine bounds
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with Bounds
  - Exceptions, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data
    - ignore only violating triples
  - Also Data exchanges, Payloads, Validation results

See [http://sws.ifi.uio.no/vocab/boundz](http://sws.ifi.uio.no/vocab/boundz) for details

Publish, re-use and combine bounds
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with * Bounds
  - Exceptions, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data
    - ignore only violating triples
  - Also *Data exchanges, Payloads, Validation results*
    - See [http://sws.ifi.uio.no/vocab/boundz](http://sws.ifi.uio.no/vocab/boundz) for details
What is Boundz?

- A vocabulary for expressing reservations about incoming data
- not what shape incoming data should have
- but what shape it *should not have*
- Vocabulary elements:
  - Fine-grained restrictions on types and predicates with Bounds
  - Exceptions, i.e., what to do in case of a violation
    - abort
    - accept parts of the incoming data
    - ignore only violating triples
  - Also Data exchanges, Payloads, Validation results
    - See [http://sws.ifi.uio.no/vocab/boundz](http://sws.ifi.uio.no/vocab/boundz) for details
- Publish, re-use and combine bounds
Boundz’ Capabilities

- No validation in the “traditional” sense, e.g.,
  - \( x \) is an integer and \( 0 \leq x \leq 100 \)
  - \( x \) must have \( n \) y’s
  - \( x \leq y \)
  - but validation of data relative to receiving dataset, e.g.,
    - Do not add more superclasses to my dataset
    - New foaf:knows relationships must relate new persons only
    - Adding new data must not rearrange existing data
    - Write-protecting (parts of) a dataset
Boundz’ Capabilities

- No validation in the “traditional” sense, e.g.,
  - \( x \) is an integer and \( 0 \leq x \leq 100 \)
Boundz’ Capabilities

- No validation in the “traditional” sense, e.g.,
  - $x$ is an integer and $0 \leq x \leq 100$
  - an $x$ must have $n$ $y$'s
No validation in the “traditional” sense, e.g.,
- $x$ is an integer and $0 \leq x \leq 100$
- an $x$ must have $n$ $y$'s
- $x \leq y$
Boundz’ Capabilities

- No validation in the “traditional” sense, e.g.,
  - \( x \) is an integer and \( 0 \leq x \leq 100 \)
  - an \( x \) must have \( n \) \( y \)'s
  - \( x \leq y \)

- but validation of data relative to receiving dataset, e.g.,
Boundz’ Capabilities

- No validation in the “traditional” sense, e.g.,
  - $x$ is an integer and $0 \leq x \leq 100$
  - an $x$ must have $n$ $y$'s
  - $x \leq y$

- but validation of data relative to receiving dataset, e.g.,
  - Do not add more superclasses to my dataset
Boundz’ Capabilities

- No validation in the “traditional” sense, e.g.,
  - $x$ is an integer and $0 \leq x \leq 100$
  - an $x$ must have $n$ $y$'s
  - $x \leq y$

- but validation of data relative to receiving dataset, e.g.,
  - Do not add more superclasses to my dataset
  - New foaf:knows relationships must relate new persons only
**Boundz’ Capabilities**

- No validation in the “traditional” sense, e.g.,
  - $x$ is an integer and $0 \leq x \leq 100$
  - An $x$ must have $n$ $y$'s
  - $x \leq y$

- But validation of data relative to receiving dataset, e.g.,
  - Do not add more superclasses to my dataset
  - New `foaf:knows` relationships must relate new persons only
  - Adding new data must not rearrange existing data
Boundz’ Capabilities

- No validation in the “traditional” sense, e.g.,
  - $x$ is an integer and $0 \leq x \leq 100$
  - an $x$ must have $n$ $y$'s
  - $x \leq y$

- but validation of data relative to receiving dataset, e.g.,
  - Do not add more superclasses to my dataset
  - New foaf:knows relationships must relate new persons only
  - Adding new data must not rearrange existing data
  - Write-protecting (parts of) a dataset
Bounds control what new triples can be added to a dataset $E$ based on the elements in $E$.

- Can express this with patterns:
  - If an incoming triple does not exist in $E$,
  - and matches the pattern, then add it.
Theoretical background

- **Bounded homomorphism**
Theoretical background

- **Bounded homomorphism**
- between incoming and existing data
Theoretical background

- **Bounded homomorphism**
- between incoming and existing data
- 20 generic bounds in total
• *Bounded homomorphism*
• between incoming and existing data
• 20 generic bounds in total
• arranged in lattice according to strength
Theoretical background

- **Bounded homomorphism**
- between incoming and existing data
- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:

\[
\begin{align*}
(S &
\quad (S \lor P \lor O) \\
(S &
\quad (S \land P \land O)
\end{align*}
\]

max.: nothing

min.: anything
Theoretical background

- **Bounded homomorphism**
- between incoming and existing data
- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:
  - any element

```
max.: nothing

min.: anything

stronger bounds, smaller permissible amendments

```

```
(T)
```

```
(S ∨ P ∨ O)
```

```
n, n, n
```

```
n, n, a
```

```
n, a, n
```

```
a, n, n
```

```
n, n, a
```

```
n, a, n
```

```
a, n, a
```

```
(S)
```

```
a, n, n
```

```
a, n, a
```

```
n, n, a
```

```
n, a, n
```

```
a, n, a
```

```
(P)
```

```
(S ∧ P ∧ O)
```

```
a, a, n
```

```
a, a, a
```

```
⊥
```
Theoretical background

- **Bounded homomorphism**
- between incoming and existing data
- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:
  - any element
  - new, element must be new: $n \notin E$
Theoretical background

- *Bounded homomorphism*
- between incoming and existing data
- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:
  - any element
  - new, element must be new: $n \not\in E$
- Controls what new triples can be added to a dataset

![Diagram showing the lattice structure of the generic bounds](attachment:diagram.png)

- max.: nothing
- min.: anything
- stronger bounds, smaller permissible amendments

For details, see paper.
Theoretical background

- **Bounded homomorphism**
- between incoming and existing data
- 20 generic bounds in total
- arranged in lattice according to strength
- Pattern legend:
  - any element
  - new, element must be new: \( n \notin E \)
- Controls what new triples can be added to a dataset
- For details, see paper.
• Do not add superclasses (but new subclasses allowed)
Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: \(\langle n, \text{rdfs:subClassOf}, a \rangle\)
• Do not add superclasses (but new subclasses allowed)
  • Bound: $\langle n, \text{rdfs:subClassOf}, a \rangle$
  • New subclass axioms must have new subjects
• Do not add superclasses (but new subclasses allowed)
  • Bound: \( \langle n, \text{rdfs:subClassOf}, a \rangle \)
  • New subclass axioms must have new subjects
  • i.e., adding new subclasses to existing classes is OK
Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: \( \langle n, \text{rdfs:subClassOf}, a \rangle \)
  - New subclass axioms must have new subjects
  - i.e., adding new subclasses to existing classes is OK
  - but old subjects in subclass axioms are not allowed,
Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: \(\langle n, \text{rdfs:subClassOf}, a \rangle\)
  - New subclass axioms must have new subjects
  - i.e., adding new subclasses to existing classes is OK
  - but old subjects in subclass axioms are not allowed,
  - i.e., no new superclasses to existing classes
Do not add superclasses (but new subclasses allowed)

- Bound: \( \langle n, \text{rdfs:subClassOf}, a \rangle \)
- New subclass axioms must have new subjects
- i.e., adding new subclasses to existing classes is OK
- but old subjects in subclass axioms are not allowed,
  - i.e., no new superclasses to existing classes
- and no new subclass relationships between existing classes.
Example

- Do not add superclasses (but new subclasses allowed)
  - Bound: $\langle n, \text{rdfs:subClassOf}, a \rangle$
  - New subclass axioms must have new subjects
  - i.e., adding new subclasses to existing classes is OK
  - but old subjects in subclass axioms are not allowed,
  - i.e., no new superclasses to existing classes
  - and no new subclass relationships between existing classes.
  - Prevents basic ontology-hijacking, protects vocabulary.
Example

- New `foaf:knows` relationships must relate new persons
Example

- New foaf:knows relationships must relate new persons
- or, “keep away from my friend graph”
Example

- New `foaf:knows` relationships must relate new persons
- or, “keep away from my friend graph”
  - \( \langle n, \text{foaf:knows}, n \rangle \)
Example

- New `foaf:knows` relationships must relate new persons
- or, “keep away from my friend graph”
  - \( \langle n, \text{foaf:knows}, n \rangle \)
  - Both subject and object of `foaf:knows` triples must be new
Example

- New foaf:knows relationships must relate new persons
- or, “keep away from my friend graph”
  - \( \langle n, \text{foaf:knows}, n \rangle \)
  - Both subject and object of foaf:knows triples must be new
  - Keeping amendments separated from receiver’s data
• Adding new data must not re-arrange existing data
Example

- Adding new data must not re-arrange existing data
  - \((n, a, a)\)
  - \((a, n, a)\)
  - \((a, a, n)\)

Disallows \((a, a, a)\), i.e., adding new triples with old elements only

Prevents skewing of existing data and noise

Implements "conservative extensions" for RDF graphs

Good fit for OWL LD (Linked Data) profile

\[ \text{OWL LD} = \text{OWL RL} \cap \text{single triple statements} \]
Adding new data must not re-arrange existing data

- \(\langle n, a, a \rangle\)
- \(\langle a, n, a \rangle\)
- \(\langle a, a, n \rangle\)

- Disallows \(\langle a, a, a \rangle\)
Example

- Adding new data must not re-arrange existing data
  - \( \langle n, a, a \rangle \)
  - \( \langle a, n, a \rangle \)
  - \( \langle a, a, n \rangle \)
- Disallows \( \langle a, a, a \rangle \)
- i.e., adding new triples with old elements only
• Adding new data must not re-arrange existing data
  • \(\langle n, a, a \rangle\)
  • \(\langle a, n, a \rangle\)
  • \(\langle a, a, n \rangle\)
• Disallows \(\langle a, a, a \rangle\)
• i.e., adding new triples with old elements only
• Prevents skewing of existing data and noise
• Adding new data must not re-arrange existing data
  • $\langle n, a, a \rangle$
  • $\langle a, n, a \rangle$
  • $\langle a, a, n \rangle$
  • Disallows $\langle a, a, a \rangle$
  • i.e., adding new triples with old elements only
  • Prevents skewing of existing data and noise

• Implements “conservative extensions” for RDF graphs
Adding new data must not re-arrange existing data

- \( \langle n, a, a \rangle \)
- \( \langle a, n, a \rangle \)
- \( \langle a, a, n \rangle \)

- Disallows \( \langle a, a, a \rangle \)
- i.e., adding new triples with old elements only
- Prevents skewing of existing data and noise

- Implements “conservative extensions” for RDF graphs
- Good fit for OWL LD (Linked Data) profile
Adding new data must not re-arrange existing data

- $\langle n, a, a \rangle$
- $\langle a, n, a \rangle$
- $\langle a, a, n \rangle$
- Disallows $\langle a, a, a \rangle$
- i.e., adding new triples with old elements only
- Prevents skewing of existing data and noise

- Implements “conservative extensions” for RDF graphs
- Good fit for OWL LD (Linked Data) profile
  - $\text{OWL LD} = \text{OWL RL} \cap \text{single triple statements}$
Implementation available:

- Prototype implementation: checks bounds, computes payloads
- Complexity: $P$

More info/material on the web:

- [http://sws.ifi.uio.no/project/boundz/](http://sws.ifi.uio.no/project/boundz/)
- [http://sws.ifi.uio.no/vocab/boundz](http://sws.ifi.uio.no/vocab/boundz)
Boundz vocabulary

Simplified overview of:
http://sws.ifi.uio.no/vocab/boundz
Vocabulary example

BBC Music dataset restrictions:

1. The vocabulary that the BBC uses must not be hijacked by adding new superclasses or superproperties.

2. Adding new `foaf:made` relationships is not tolerated, unless both artist and record is new to the BBC dataset; their current library is regarded as complete with respect to the albums of enlisted artists, but is open for extensions with new artists.

3. More fanpages may be added, but an existing fanpage cannot be related to more artists.

4. No new information about existing genres may be added.

5. Also, assume the BBC keeps a special dataset about the Beatles which is not under their management, so they want to disallow any new information using only elements from this dataset. However, new information may *relate* to the Beatles dataset.

```prolog
1  ex:bbcmusic a bz:BoundedGraph ;
2  bz:hasBound bzs:RDFS ,
3  
4  \[ a bz:Aso ;  bz:predicateValue foaf:made ] ,
5  \[ a bz:o ;  bz:predicateValue mo:fanpage ;
6  \[ a bz:T ;  bz:hasException bz:ignoreViolations ] ,
7  \[ a bz:T ;  bz:subjectClass mo:Genre ] ,
8  \[ a bz:T ;  bz:objectClass mo:Genre ] .
9  ex:beatles a bz:BoundedGraph ;
10 bz:hasBound [ a bz:KKspo ] .
```
Boundz output example

1 <file:///test/test>
2   a :ExchangeSchema ;
3   :hasBound _:b1 , _:b2 , _:b3 , _:b4 , _:b5 , _:b6 , _:b7 , _:b8 ;
4   :hasSource <file:///test/uni1_0.ttl> , <file:///test/uni1_1.ttl> ;
5   :hasTarget <file:///test/uni500_2.ttl> ;
6   :outputPayload "false"^^xsd:boolean ;
7   :outputViolations "false"^^xsd:boolean ;
8   :sourceReasoning "false"^^xsd:boolean ;
9   :targetReasoning "false"^^xsd:boolean .
10
11 <http://test/test/1372168337206/31381>
12   a :Exchange ;
13   :hasPayload [ :noOfTriples "6498"^^xsd:long ] ;
14   :hasViolation [ a :Violation ;
15     :hasSource <file:///test/uni1_1.ttl> ;
16     :noOfTriples "566"^^xsd:long ;
17     :onBound _:b7 ] ;
18   :instanceOf <file:///test/test> ;
19   :timestamp "1372168337206"^^xsd:long .
20
21   _:b7 a :T ;
22   :classRestriction ub:University ;
23   :hasException :ignoreViolations ;
24   :hasRestriction ub:University ;
25   :objectClass ub:University ;
26   :objectRestriction ub:University .

Excerpt of http://sws.ifi.uio.no/project/boundz/impl/outputExchanges.ttl
SPARQL representation of bz:KKspo:

1 CONSTRUCT
2 { ?s ?p ?o .}
3 WHERE
4 { GRAPH <SOURCE>
5 { ?s ?p ?o
6   GRAPH <TARGET>
7     { { { ?s _:5 :b0 } UNION { _:b1 ?s _:b2 } UNION { _:b3 _:6 ?s } }
8      { { ?p _:3 :b4 } UNION { _:b5 ?p _:b6 } UNION { _:b7 _:4 ?p } }
9      { { ?o _:1 _:b8 } UNION { _:b9 ?o _:b10 } UNION { _:b11 _:2 ?o } }
10     }
11   }
12   MINUS
13   { GRAPH <TARGET>
14     { ?s ?p ?o }
15   }
16 }
References

