Model Importer Implementation

Architecture

Unknown macro: 'drawio'

Importer Purpose

The importer will create and update nodes. Nodes will be filled with properties and relations to exactly match the third-party model. Aspect and definitions should have been created previously.

Available methods in BImporter

```
public void doExecute(Context cx)
```

This method executes the import in a thread and fill the timestamps (last attempt, success and failure) of the component.

The following methods will be called (in order):

- initImport()
- processImport()
- finishImport()

This method will also manage importer status. It is called by Niagara action only (no automatic execution).

There is theoretically no need to force its call or to override this method.

```
protected abstract void processImport() throws Exception;
```

This method contains the core the import. You can use the following methods to manipulate nodes:

- createNode()
- updateNode()
- moveNode()
- deleteNode()

Then engine will process these operations to import the nodes.

This theoretically is the only one method you have to override in your own connector.

```
public void log(BLog log)
```

This method sends a log through the Active logs system. See Logs.

```
protected void initImport() throws Exception
```

This method initializes engine, scope and operations list. All of these are available as protected variables.

```
protected void finishImport() throws Exception
```

This method submits operations to engine, wait for them to be executed and manages the state of the importer depending on the result. It also initialize any node that has been changed (so the modification can be taken into account by tags and strategies).

```
protected void createNode(BDefinition definition, String id, String displayName, Map<String, BValue> tags)
```

This method sends an operation to engine to create a node in a definition and fill its tags.

- Definition: definition to add the node
- Id: unique identifier of the node. It is advised to create ids with only alpha numeric characters, starting with a letter.
- DisplayName: display name of the node
- Tags: tags of the node (string, numeric, boolean, etc.). It is better to match with NodeTags (previously created in the definition)

Behavior in case of duplicate ids: if a node has already the same id, we consider that this is the same node and we merge tags of both nodes. So if you execute 2 consecutive imports, data will be merged (and not duplicated).

```
protected void updateNode(String id, String displayName, Map<String, BValue> tags)
```

This method sends an operation to engine to update the tags of an existing node. The node is identified using its id. A tag cannot be removed, this function will only add or replace a tag.

```
protected void moveNode(String id, String idFrom, String idTo)
```

This method sends an operation to change the ascendant of a node. Reminder: the relation between a node and its ascendant is "b:isIn". null can be passed as parameter "idFrom" if node had previously no ascendant. null can be passed as parameter "idTo" to make the node orphan.

```
protected void deleteNode(String id)
```

This method sends an operation to delete an existing node identified by its id.

```
protected void addRelation(String idFrom, String idTo, String relationId, Map<String, BValue> tags)
```

This method sends an operation to create a relation with a given id between two nodes identified by their id. A relation can be tagged. It is better to match with

```
protected void deleteRelation(String idFrom, String idTo, String relationId)
```

This method sends an operation to delete an existing relation between two nodes.