netymon pty ltd

ABN 56 113 088 145

Kowari Proposal

PROPOSED FIX FOR SYMBOLIC-TRANSFORMATION

netymon pty ltd

Unit 3 / 13 Denham St Annerley 4103 Australia - +61 7 3392 1296 - +61 414 517 882

Abstract

It is proposed that Kowari be enhanced

1. Refactor Constraint to add explicit access to the constraint's model element
2. Removal of ConstraintResolutionHandler from ConstraintDescriptor and it's replacement with a resolver delegating handler for custom constraints. (Thank-you Alex Hall [NGC] for proposing this elegant fix)
3. Renaming all query evaluation methods and classes to refer to evaluation in place of resolution.
4. Itql Compound Syntax, to support convenient expression of reified queries in itql

Summary of Problem

INTRODUCTION

Kowari is a non-relational database management system optimised for the store of RDF graphs. The Kowari team released version 1.1 at the end of 2005. This version included enhancements to Kowari's 'ResolverSPI' (funded by the Australian Defence Science and Technology Organisation) that permit Kowari to be used as an information integration platform.

The original ResolverSPI, originally released as 1.1pre1, was deliberately limited in its scope. Specifically it was limited to integrating external sources of rdf-like data with Kowari. It was always understood that this would be insufficient to support integrating arbitary data and would need to be extended if it was to fulfil the ultimate goal of making Kowari a powerful EII platform. In 1.1 we have implemented the first version of a full Resolver SPI capable of integrating arbitary datastores, using non-rdf datamodels, into traditional rdf queries.

BASIC KOWARI QUERIES

Kowari is traditionally queried by either access to a programmatic SessionAPI or a query language iTQL which is parsed internally into calls on the Session API. To illustrate, an example of a simple query requesting the suburb portion of an address for an individual represented by the uri <example:fred> follows:

```
select $subur from <test:model>
where <example:fred> <hasAddress> $addr and
```

$addr <inSuburb> $suburb

This will be parsed into a Java object representing the following structurewhich will be passed into a Session object for evaluation:

```
QUERY :
    SELECT-LIST := [ $suburb ]
    MODEL-EXPRESSION :
        MODEL-RESROURCE : <test:model>
    CONSTRAINT-EXPRESSION :=
        CONSTRAINT-CONJUNCTION (
            CONSTRAINT := [ <example:fred> <hasAddress> $addr ]
            CONSTRAINT := [ $addr <inSuburb> $suburb ]
        )
```

This will result in the following processing:

1. Kowari engine obtains the Resolver associated in the system-model with the uri <test:model>
2. The query engine passes each CONSTRAINT to Resolver.resolve() which returns the result as a Tuples object
3. The two tuples are then joined corresponding to the ConstraintConjunction which itself corresponds to the and operation expressed in the iTQL query
4. The result is then wrapped as an Answer for transmission back to theclient

S Y M B O L I C T R A N S F O R A M T I O N

An intuitive model behind SymbolicTransformation is that of Reified-RPC. We represent external data as a function of N in-parameters to M out-parameters. Possibly the most trivially intuitive of these functions would be basic addition.

In		*-i>*	*Out*
1	1	-a>	2
1	2	-b>	3
2	1	-c>	3
2	2	-d>	4
...	...	...	...

We recognise that any function of this nature can be mapped to rdf in multiple ways, however at the very least the above can be mapped to the following graph:

:_a	<add:lhs>	“1”
:_a	<add:rhs>	“1”
:_a	<add:sum>	“2”
:_b	<add:lhs>	“1”
:_b	<add:rhs>	“2”
:_b	<add:sum>	“3”
:_c	<add:lhs>	“2”
:_c	<add:rhs>	“1”
:_c	<add:sum>	“3”
:_d	<add:lhs>	“2”
:_d	<add:rhs>	“2”
:_d	<add:sum>	“4”
...	...	...

which can therefore be queried in iTql (or a SessionAPI Query object). ie.

```
select $sum from <add:model>
where
  $_bn <add:lhs> "1" and
       $_bn <add:rhs> "2" and
       $_bn <add:sum> $sum;
```

PREPRATORY WORK

Experience with the development of resolvers with the query rewriting API has

uncovered several issues with the new ResolverSPI.

1. Many SymbolicTransformer's require explicit access to the model on a constraint, and the ability to map a model URL to a model-type. This ability is currently implicit in the transformers access to the system-model. However since the development of the API additional work done to resolve model URI/URN conflation has rendered this insufficient to perform the mapping, and an explicit mechanism is required.
2. The current requirement for a custom Constraint to implement its own ConstraintResolutionHandler has encouraged the complete resolution of these constraints within the handler. This is contary to the evaluation design of kowari where all resolution happens within the Resolver.resolve() method. This has flow-on effects by bypassing Kowari's security layer. All primitive constraint resolution should be delegated to the appropriate Resolver by DatabaseSession via the QueryEvaluationContext.
3. Part of the reason for the confusion between Constraint Handling and Resolution is the use of the phrase 'Resolution' to refer both to graph operations and query-calculus reductions. The use of the term Evaluate/Evaluation is preferred for the latter, reserving Resolve/Resolution for graph operations.
4. A trivial 3-arity relation requires 3 primitive constraints to be included in the query. In general a query against an N-arity relationrequires N constraints per relational-constraint. As it is expected that users will

desire the ability to express multiple relational-constraints in a single query, this is considered undesirable. To aleviate this itis proposed that we extend the iTql query syntax with syntactic sugar based on SPARQL/Turtle to simplify these constraints.

It is also recommended that time be spent writing an abstract transformer that can provide declarative transformation of conjunctive compound constraints such as those discussed above. This work will substantially simplify the development of most symbolic transformers

Proposal

It is proposed that Kowari be enhanced

1. Refactor Constraint to add explicit access to the constraint's model element
2. Removal of ConstraintResolutionHandler from ConstraintDescriptor and it's replacement with a resolver delegating handler for custom constraints. (Thank-you Alex Hall [NGC] for proposing this elegant fix)
3. Factoring of the localization code in LocalQueryResolver.resolve into a localizeConstraint() method on ConstraintDescriptor.
4. Renaming all query evaluation methods and classes to refer to evaluation in place of resolution.
5. Modifying SymbolicTransformation or MutableLocalQuery to provide a context to the transform method that permits the mapping of modelURI's to modelTypeURI's.
6. Itql Compound Syntax, to support convenient expression of reified queries in itql

EXPLICIT MODEL

Currently the query evaluator makes an implicit assumption that the 4th element of a constraint is the model. This hasn't been true for a long time and there are several kludges in the evaluator to compensate for this. With the introduction of custom constraints it is no longer possible to isolate these kludges from the SPI; so it's time to remove them and fix it properly.

Constraint needs a method getModelElement().

CONSTRAINT-DESCRIPTOR REFACTOR

It was never intended that ConstraintResolutionHandlers would resolve their constraints directly. The design intent was for the iTQL query calculus to be reduced by the LocalQueryResolver ultimately to an Tuples Algebra expression that could be resolved directly. Primitive values within the Algebra are simple constraints/operations to be resolved against a triple-graph. ConstraintResolutionHandler (now ConstraintExpressionEvaluator) will still be required, as this represents the evaluation of both simple and complex constraints (conjunctions, disjunctions, etc). Alex Hall's insight was that there is currently no existing or anticipated simple constraint that doesn't share the same behaviour. So it is proposed that access to this be removed from custom constraints, and the common behaviour factored into a common PrimitiveConstraintExpressionEvaluator.

LOCALIZATION REFACTOR

Currently LocalQueryResolver.resolve hard-codes the construction of a localized ConstraintImpl from its argument, and other kludges exist for the other hard-coded constraint types. This should be factored out into a seperate interface analogous to the model-rewrite logic on ConstraintDescriptor. The existance of custom constraints mean we can no longer hard-code kludges to handle this logic.

RENAMING REFACTOR

Straight forward renaming.

Currently	*After Rename*
LocalQueryResolver.resolve	LocalQueryEvaluator.evaluate
ConstraintResolutionHandler.resolve	ConstraintExpressionEvaluator.evaluate
ModelResolutionHandler.resolve	ModelExpressionEvaluator.evaluate
ConstraintOperations.resolveConstraintExpression	ConstraintOperations.evaluateConstraintExpression
ConstraintOperations.resolveModelExpression	ConstraintOperations.evaluateModelExpression
LocalQuery.resolve (excluding Tuples LocalQuery.resolve(Constraint))	LocalQuery.evalute

SYMBOLIC TRANSFORMATION REFACTORING

This involves refactoring the API to make access to the modelURL→ modelTypeURL mapping explicit. Implementing this will require refactoring the context passed to SymbolicTransformation to provide access to the mapping

ITQL COMPOUND SYNTAX

This would involve adding two forms of syntactic sugar to the iTql parser.

```
[ <a1> <v1>, <a2> <v2> : <v3> in <m> ] =>
    $_t <a1> <v1> and
    $_t <a2> <v2> and
    $_t <a2> <v3>
and
  { <s> <a1> <v1>, <a2> <v2> : <v3> in <m> }=>
    <s> <a1> <v1> and
    <s> <a2> <v2> and
    <s> <a2> <v3>
```

the key to the first expansion being the guarentee that $_t is unusedand unreferencable by other constraints in the query.