RINQ
Concept of a Ruby Integrated Query Language

Sten Friedrich

Technische Fachhochschule Berlin
"University of Applied Sciences"

sten.friedrich@gmail.com

29th February 2008

Abstract

This paper introduces a Ruby Integrated Query Language for heterogeneous data sources. The concept is based on a research into different query languages regarding general and specific criteria for query language development and the possibilities of dynamic programming languages like Ruby. Native integration in the programming language and the capability to use it with heterogeneous data sources are the main goals. With this concept the impedance mismatch on the application layer can be resolved.

1 Introduction

After a lot of research into query languages and persistence of objects I finished my bachelor's thesis about creating a query language for object-databases in Ruby in January 2008. This paper is a summary of the results. In conclusion I took the basic concept of LINQ as the basis for developing a Ruby Integrated Query language for heterogeneous data sources. Before defining the new query language I analysed the current situation in application development and persistence.

2 Current situation in application development

The introduction of the object-oriented programming paradigm at the application layer created an impedance mismatch between programming and query languages and an impedance mismatch between the application and persistence layer.

The following table gives an overview of the current situation in application development.

Tool		**ORACLE**		**JDBC**		**Hibernate**		**db4o**		**LINQ**		**AOQL**	
Data storage		RDB		RDB		RDB		ODB		heterogeneous		abstract	
		S	**M**	**S**	**M**	**S**	**M**	**S**	**M**	**S**	**M**	**S**	**M**
Application layer	**Application programming**	PL/SQL	rel	Java	oo	Java	oo	Java	oo	C#	oo	any PL	oo
	Programmer			**IS**	**IM**	**IS**						**IS**	
	Queries	PL/SQL	rel	SQL	rel	HQL	oo	Java (NQ)	oo	C# (LINQ)	oo	AOQL	oo
	Mapping (automatic)					**IS**	**IM**	🟩		**IS**	**IM**		
Persistence layer	**Queries**	PL/SQL	rel	SQL	rel	SQL	rel	SODA	oo	e.g. SQL	e.g. rel	AOQL	abstract (oo)

S – programming / query language
M – data model
RDB – relational database
ODB – object database
oo – object-oriented
rel – relational
PL – programming language

Impedance mismatch:
IS – between programming and query language
IM – between object-oriented and relational data model

Colors:
yellow – relational environment
blue – object-oriented environment
cream white – abstract environment

Table 1: Current situation in application development

As you can see, the programmer can choose from a mixture of many different programming languages and databases. To define a query language you need to analyse this situation and create a basic concept.

3 How to create a query language?

Query languages have to meet a lot of requirements to be a complete query language. There are general criteria and criteria for easy learning and acceptance. In order to create such a language you have to analyse some existing query languages on the basis of these requirements. The results are shown at table 2 and 3.

3.1 General criteria

Table 2 is based on the criteria list of Saake, Schmitt and Türker in the book "Objektdatenbanken Konzepte, Sprachen, Architekturen".

The list is expanded by a few points to meet the latest developments needs in the field of query languages.

Criterion	SQL	HQL	JPQL	db4o	LINQ	AOQL
Ad-hoc queries (can be formulated without an application)	X	-	-	X	-	X
Generic operators (a few simple operators)	X	X	X	-	X	X
Application independence -> Data independence (independent from the application and data)	X	-	-	X	-	X
Declarative (can be formulated without imperative code)	X	X	X	-	X	X
Set-oriented (applicable for sets of data)	X	X	X	X	X	X
Orthogonality (every combinations that makes sense should be allowed)	X	X	X	X	X	X
Efficiency (efficient Operations)	X	X	X	X	X	X
Expandability (can be used with new types)	X	X	X	X	X	X
Conceptual closure (Result can be consistently presented in the data model)	X	X	X	X	X	X
Adequacy (use all constructs of the data model)	X	X	X	X	X	X
Completeness (at least the power of relational query languages)	X	X	X	-	X	X
Security (terminable => finite result)	X	X	X	-	-	-
Optimizable (automatically optimizable)	X	X	X	X	X	X
Formal Semantics (convertable to formal algebra => formal construct)	X	X	X	X	X	X
Limiting (incomplete computation)	X	X	X	X	X	X
Flexibility (new userspecific operators)	-	-	-	X	X	X
(userspecific implementation)	-	-	-	X	X	-
Composition -> Optimizable (query pieces can be orthogonally composed)	X	X	X	X	X	X
Native queries (native in programming language)	-	-	-	X	X	-
Universal (usable for heterogeneous data sources)	-	-	-	-	X	X

Table 2: General criteria

3.2 Specific criteria for easy learning and acceptance

SQL is an approved language for queries. To follow the principles of SQL will guarantee the acceptance of the new language. Table 3 shows an overview of analysis.

Query functions	**Examples**	**SQL**	**HQL**	**JPQL**	**db4o**	**LINQ**	**AOQL**
Domain	FROM	X	X	X	X	X	X
Restriction	WHERE	X	X	X	X	X	X
Conditional operators	BETWEEN, IN, LIKE	X	X	X	-	-	X
Projection	SELECT	X	X	X	-	X	X
Creation of new types	SELECT NEW	X	-	-	-	X	X
Collections	SELECTMANY	-	-	-	-	X	-
Combination (Path expressions)	. (Dot notation)	-	X	X	X	X	X
Combination (Joins)	JOIN	X	X	X	-	X	X
Ordering	ORDER BY	X	X	X	X	X	-
Grouping	GROUP BY	X	X	X	-	X	-
Grouping with condition	HAVING	X	X	X	-	X	-
Subselects	SUBSELECT	X	X	X	X	X	X
Quantifiers	EXISTS, ALL, ANY	X	X	X	-	X	X
Aggregation	COUNT, SUM, MIN	X	X	X	-	X	X
Set operations	DISTINCT, UNION	X	X	X	-	X	-
Element operations	FIRST, LAST, SINGLE	X	X	X	-	X	-
Partitioning	TAKE, SKIP	-	-	-	-	X	-
Concatenation	CONCAT	-	-	-	-	X	-
Equality	SEQUENCEEQUAL	-	-	-	-	X	-
Conversion	TOARRAY, TOLIST	-	-	-	-	X	-
Generation	RANGE, REPEAT	-	-	-	-	X	-

Table 3: Specific criteria for easy learning and acceptance

3.3 Role of the programming language

The relation to the programming language is an important aspect in creating a query language. It can be natively integrated in the programming language or combined with the data store. Native integration delivers type safety, refactoring and early error detection to the programmer. The combination with the data store delivers data independence of the application layer. If you want to integrate the query language natively in the programming language (viz. defining query operators), the programming language has to achieve some conditions in order to implement e.g. projection, expandability, closures and deferred execution. Ruby meets this demands because of its dynamic, flexibility, dynamic typing and so on. It can handle code as data with its blocks for methods and you can expand types at run time. With all this features Ruby is a great candidate for integrating queries natively into a programming language.

3.4 Conclusion

The result of analysing the current situation in application development and existing query languages is, that the important impedance mismatch for the application development is the mismatch at the application layer. Integrating the query language natively into the programming language is the main goal to resolve this impedance mismatch. The mismatch to the persistence layer can be resolved with automatic mapping or native data storage in object databases. That's why the query language should be integrated in the programming language and should work with heterogeneous data sources e.g. main memory, relational or object databases, xml, filesystem and so on. LINQ combines those two important things. For this reason, the LINQ concept will be the inspiration for the Ruby Integrated Query language.

4 RINQ - Ruby Integrated Query Language

4.1 Conception

The new query language in Ruby is called RINQ, like Ruby Integrated Query and is natively integrated in the programming language, has sql-like query operators and supports heterogeneous data sources. RINQ is available for types which include the module Enumerable. Furthermore it provides two interfaces RExpressionTree and ROODB for data source specific implementation of the query operators. Table 4 shows this concept.

Data	(type)	**Enumerable**		**RexpressionTree**		**ROODB**		**other**	
Storage	(example)	HS		RDB		ODB		file system	
		S	**M**	**S**	**M**	**S**	**M**	**S**	**M**
Application layer	**Application programming**	Ruby	oo	Ruby	oo	Ruby	oo	Ruby	oo
	Programmer								
	Queries	Ruby (RINQ)	oo	Ruby (RINQ)	oo	Ruby (RINQ)	oo	Ruby (RINQ)	oo
API	**Input**	RINQ methods		RINQ methods		RINQ methods		RINQ methods	
	Output	Query result	oo	Expression tree	oo	SODA	oo	???	oo
	Mapping (automatic)			**IS**	**IM**				
Persistence layer	**Queries**			SQL	rel.	SODA	oo	???	???

S – programming / query language
M – data model
RDB – relational database
ODB – object database
oo – object-oriented
rel – relational

Impedance mismatch:
IS – between programming and query language
IM – between object-oriented and relational data model

Colors:
yellow – relational environment
blue – object-oriented environment
cream white – abstract environment

Table 4: Concept of RINQ

RINQ can appear in two kinds, as query operators (methods) or as query expressions (DSL-like). Ruby already contains keywords such as select, which are implemented for enumerable data types. That's why it is necessary to extend the new RINQ query operator keywords with a "q", because they should be valid for different types. That means, select in SQL is equivalent to qselect in RINQ.

4.2 Query operators (query methods) - language definition

Example

```
customernames = customers.
                qwhere   {|c| c.address.city == "London"}.
                qselect  {|c| {:firstname => c.firstname,
                               :lastname  => c.lastname}}.
                qorderby {|c| c.lastname}
```

Meta operators

- ::= (assignment operator)
- OR (exclusive OR)
- `|` (not an exclusive OR like Backus-Naur-Form but parameter definition in a Ruby code block)
- `[]` (not an optional construct like Backus-Naur-Form but an Array - because Ruby has only one code block for the query-expressions)

Meta Elements

Basic element is the "collection" as a synonym for instances of queryable objects of type Enumerable, RExpressionTree or ROODB.

```
collection          ::=  enumerable-  OR  roodb-  OR
                         rexpressiontree-object

source              ::=  collection
sourceElem          ::=  element of the collection

second              ::=  collection of source type
secondElem          ::=  element of the second collection

inner               ::=  collection of any type
innerElem           ::=  element of the inner collection

result              ::=  collection of other than source type
resultElem          ::=  element of the result collection

other               ::=  object of any type

expr_XxxElem        ::=  expression with the Xxx element
```

```
expr_XxxElem_YyyElem
                    ::=  expression with the Xxx element and
                                         the Yyy element

predicate           ::= expr_SourceElem

selector            ::= expr_SourceElem  OR
                        {:newElem_attrib1 => expr_SourceElem,
                         :newElem_attrib2 => expr_SourceElem,
                        ...}

numericSelector     ::= expr_SourceElem  OR
                        {:newElem_attrib1 => expr_SourceElem,
                         :newElem_attrib2 => expr_SourceElem,
                        ...}

resultSelector      ::= expr_SourceElem_InnerElem  OR
                        {:newElem_attrib1 =>
                           expr_SourceElem_InnerElem,
                         :newElem_attrib2 =>
                            expr_SourceElem_InnerElem,
                        ...}

keySelector         ::= expr_SourceElem  OR
                        [expr_SourceElem,expr_SourceElem,
                           ...]

innerKeySelector    ::= expr_InnerElem   OR
                        [expr_InnerElem,expr_InnerElem, ...]

sourceValue         ::= element of source type
indexValue          ::= integer value
countValue          ::= integer value

keyComparer         ::= Proc.new{key_compare_code}

equalKeyComparer    ::= Proc.new{key_equal_compare_code}

sourceComparer      ::= Proc.new{source_compare_code}

equalSourceComparer ::= Proc.new{source_equal_compare_code}

resultComparer      ::= Proc.new{result_compare_code}
```

Syntax

RINQ: RESTRICTION Operator

```
[sourceElem]= source.qwhere {|sourceElem| predicate}
```

RINQ: PROJECTION Operators

```
[resultElem]= source.qselect {|sourceElem| selector}
[innerElem] = source.qselectmany {|sourceElem| selector}
[resultElem]= source.qselectmany {|sourceElem|
                                  [selector,resultSelector]}
```

RINQ: JOIN Operators

```
[resultElem]= source.qjoin (inner)
  {|sourceElem,innerElem| [keySelector,innerKeySelector,
                           resultSelector]}

[resultElem]= source.qjoin (inner,equalKeyComparer)
  {|sourceElem,innerElem| [keySelector,innerKeySelector,
                           resultSelector]}
# qgroupjoin analog
```

RINQ: ORDERING Operators

```
[orderdSourceElem]= source.qorderby
                           {|sourceElem| keySelector}
[orderdSourceElem]= source.qorderby(KeyComparer)
                           {|sourceElem| keySelector}
# qorderbydesc analog
[orderdSourceElem]= orderdSource.qthenby
                           {|orderdSourceElem| keySelector}
[orderdSourceElem]= orderdSource.qthenby(KeyComparer)
                           {|orderdSourceElem| keySelector}
# qthenbydesc analog
[reverseSourceElem]= source.qreverse
```

RINQ: GROUPING Operators

```
{key => [sourceElem]}= source.qgroupby
                       {|sourceElem| keySelector}
{key => [sourceElem]}= source.qgroupby (equalKeyComparer)
                       {|sourceElem| keySelector}
{key => [resultElem]}= source.qgroupby
                       {|sourceElem| [keySelector,selector]}
{key => [resultElem]}= source.qgroupby (equalKeyComparer)
                       {|sourceElem| [keySelector,selector]}
```

RINQ: QUANTIFIERS

```
bool= source.qcontains (other)
bool= source.qcontains (other, equalSourceComparer)
```

```
bool= source.qany {|sourceElem| predicate}
bool= source.qall {|sourceElem| predicate}
```

RINQ: AGGREGATE Operators

```
int = source.qcount
int = source.qcount    {|sourceElem| predicate}
numeric= numericSource.qavg
numeric= source.qavg   {|sourceElem| numericSelector}
numeric= numericSource.qsum
numeric= source.qsum   {|sourceElem| numericSelector}
sourceElem=source.qmin
sourceElem=source.qmin(sourceComparer)

resultElem=source.qmin               {|sourceElem| Selector}
resultElem=source.qmin(resultComparer){|sourceElem| Selector}
# qmax analog
```

RINQ: SET Operators

```
[sourceElem]= source.qdistinct
[sourceElem]= source.qdistinct(equalSourceComparer)
[sourceElem]= source.qunion(second)
[sourceElem]= source.qunion(second,equalSourceComparer)
# qintersect and qexcept analog
```

RINQ: ELEMENT Operators

```
[sourceElem]= source.qfirst
[sourceElem]= source.qfirst          {|sourceElem| predicate}

[sourceElem]= source.qfirstordefault
[sourceElem]= source.qfirstordefault {|sourceElem| predicate}
# qlast and qlastordefault analog
```

RINQ: PARTITIONING Operators

```
[sourceElem]= source.qtake      {|sourceElem| countValue}
[sourceElem]= source.qtakewhile {|sourceElem| predicate}
# qskip and qskipwhile analog
```

RINQ: CONCATENATION Operator

```
[sourceElem]= source.qconcat(second)
```

RINQ: EQUALITY Operator

```
bool= source.qcollectionequal(second)
bool= source.qcollectionequal(second,equalSourceComparer)
```

4.3 Query expressions (DSL) - language definition

To increase the abstraction level of the new query language you can create and implement it as a DSL (Domain specific language). This can be realised in Ruby with Metaprogramming. Rubys open syntax and dynamic concept offer good premises. Peter Vanbroekhoven gives a tutorial about that in his presentation[1]. With Metaprogramming it would also be possible to get rid of the "q" in the query operator method names. Query expressions have been made as abstract keywords which call the query methods of RINQ. They are not fully implemented yet.

Example

```
customerfirstnames = query do
    qfrom :c => customers
    qwhere c.lastname == "Mitchel"
    qselect c.firstname
end
```

Meta Elements

```
source              ::= collection
expr_SourceElem     ::= expression with the source element
name                ::= symbol
newElem_attrib      ::= symbol
predicate           ::= expr_SourceElem
selector            ::= expr_SourceElem |
                        (expr_SourceElem ,)* |
                        (newElem_attrib => expr_SourceElem ,)*
```

Syntax

```
query-expression    ::= qfrom-clause query-body
query-body          ::= query-body-clause* final-query-clause
query-body-clause   ::= (qfrom-clause | qwhere-clause)
qfrom-clause        ::= qfrom name => source
qwhere-clause       ::= qwhere predicate
qselect-clause      ::= qselect selector
final-query-clause  ::= qselect-clause
```

5 Can abstract and language integrated queries be combined?

RINQ represents the trend to integrate the query language into the programming language, but the impedance mismatch between the application and persistence layer still exists. With RINQ you can't create application independent queries. This is only possible with a data storage integrated query language (maybe with imperative elements like PL/SQL). To realise this approach the Object Database Technology Working Group (ODBTWG) released a White Paper[2] about an Abstract Object

[1] cf. [Va07]
[2] cf. [Ca07]

Query Language based on (SBQL) a Stack-Based Query Language with a Stack-Based Data Storage Architecture. The research for SBA and SBQL was done by Prof. Kazimierz Subieta and his students at the Polish-Japanese Institute of Information Technology, Warsaw[3].

5.1 AOQL - Abstract Object Query Language

AOQL is an abstract query language that is combined with an abstract data store. The connection with the programming language is an unsolved problem. Table 5 gives an overview.

Data source (abstract) ASx: Abstract store of type x			**AS0**		**AS1**		**AS2**		**AS3**	
			relational, XML, simple persisted objects (no inheritance)		AS0 + classes and static inheritance		AS1 + dynamic object roles and dynamic inheritance		(AS1 or AS2) + encapsulation (public properties)	
			S	**M**	**S**	**M**	**S**	**M**	**S**	**M**
Application layer		**Application programming**	any PL	oo	any PL	oo	any PL	oo	any PL	oo
		Programmer	**IS**		**IS**		**IS**		**IS**	
		Parser	**IS**		**IS**		**IS**		**IS**	
		Queries	AOQL	oo	AOQL	oo	AOQL	oo	AOQL	oo
Persistence layer	**Abstract data source**	**Queries**	AOQL	oo	AOQL	oo	AOQL	oo	AOQL	oo
		Mapping (automatic)	**IS**	**IM**	**IS**					
	Real data source	**Queries**	e.g. SQL (RDB)	rel.	e.g. SODA (OODB)	oo	??? (???)	oo	??? (???)	oo

based on SBQL (Stack Based Query Language)

S – programming / query language
M – data model
RDB – relational database
ODB – object database
oo – object-oriented
rel – relational
PL – programming language

Impedance mismatch:
IS – between programming and query language
IM – between object-oriented and relational data model

Colors:
yellow – relational environment
blue – object-oriented environment
cream white – abstract environment

Table 5: AOQL - Abstract Object Query Language

5.2 AOQL and RINQ together - a vision

The combination of RINQ and AOQL with its abstract approach to query and store data could be a possible solution to resolve the impedance mismatch at the application layer as well as the impedance mismatch between the application and the persistence layer. Table 6 shows a possible variant.

[3]cf. [URLb]

Data storage	(type) (example)		**RAS0** RDB		**RAS1** OODB		**RAS2** ???		**RAS3** ???	
			S	**M**	**S**	**M**	**S**	**M**	**S**	**M**
Application layer		**Application programming**	Ruby	oo	Ruby	oo	Ruby	oo	Ruby	oo
		Programmer								
		Queries	Ruby (RINQ)	oo	Ruby (RINQ)	oo	Ruby (RINQ)	oo	Ruby (RINQ)	oo
API		**Input**	RINQ methods		RINQ methods		RINQ methods		RINQ methods	
		Output	AOQL	oo	AOQL	oo	AOQL	oo	AOQL	oo
Persistence layer	**Abstract data source**	**Queries**	AOQL	oo	AOQL	oo	AOQL	oo	AOQL	oo
		Mapping (automatic)	**IS**	**IM**						
	Real data source	**Queries**	e.g. SQL(RDB)	rel.	e.g. OODB	oo	???	oo	???	oo

S – programming / query language
M – data model
RDB – relational database
ODB – object database
oo – object-oriented
rel – relational

Impedance mismatch:
IS – between programming and query language
IM – between object-oriented and relational data model

Colors:
yellow – relational environment
blue – object-oriented environment
cream white – abstract environment

Table 6: AOQL and RINQ together - a vision

The query operators of RINQ could be used to create AOQL queries which realise the access to the data store. So AOQL is always the interface to the data store. That means you have two options, queries with RINQ methods from the application or queries with AOQL directly. The data store has to implement and understand AOQL only and the programmer could write his queries with programming language integrated query operators. So application development could be free of impedance mismatches.

References

[BH07] DON BOX and ANDERS HEJLSBERG: *.NET Language-Integrated Query*. 2007. Url: http://msdn2.microsoft.com/en-us/library/bb308959.aspx.

[Ca07] MICHAEL CARD: *Next-Generation Object Database Standardization*. 2007. Url: http://odbms.org/experts.html#article15.

[CR06] WILLIAM COOK and CARL ROSENBERGER: *Native Queries for Persistent Objects*. 2006. Url: http://odbms.org/experts.html#article2.

[HT07] Anders Hejlsberg and Mads Torgersen: *The .NET Standard Query Operators.* 2007. Url: http://msdn2.microsoft.com/en-us/library/bb394939.aspx.

[JBC+06] Eric Jendrock, Jennifer Ball, Debbie Carson, Ian Evans, Scott Fordin and Kim Haase: *The Java^{TM} EE 5 Tutorial Third Edition.* Sun Microsystems, Inc., 2006. Url: http://java.sun.com/javaee/5/docs/tutorial/doc/?wp406229&QueryLanguage.html#wp80587.

[PEH+06] Jim Paterson, Stefan Edlich, Henrik Hörning and Reidar Hörning: *The Definitive Guide to db4o.* Apress, Berkely, CA, USA, 2006.

[SST03] Gunter Saake, Ingo Schmitt and Can Türker: *Objektdatenbanken- Konzepte, Sprachen, Architekturen.* International Thomson Publishing, 2003.

[TFH06] Dave Thomas, Chad Fowler and Andy Hunt: *Programming Ruby: The Pragmatic Programmers' Guide, Second Edition.* Pragmatic Bookshelf, October 2006.

[URLa] *Hibernate Reference Documentation Version: 3.2.2.* Url: http://www.hibernate.org/hib_docs/v3/reference/en/pdf/hibernate_reference.pdf.

[URLb] *Stack-Based Architecture (SBA) and Stack-Based Query Language (SBQL).* Url: http://www.sbql.pl/.

[Va07] Peter Vanbroekhoven: *How to create a Domain Specific Language? - Metaprogramming in Ruby.* 2007. Url: http://www.xaop.com/blog/2007/10/07/video-how-to-create-a-domain-specific-language-/.

[WW07] Ralf Westphal and Christian Weyer: *.NET 3.0 kompakt.* Spektrum Akademischer Verlag, 2007.