Optique™

Project N°:	**FP7-318338**
Project Acronym:	**Optique**
Project Title:	**Scalable End-user Access to Big Data**
Instrument:	**Integrated Project**
Scheme:	**Information & Communication Technologies**

Deliverable D1.3
Joint Phase 2 Evaluation and Phase 3 Requirement Analysis

Due date of deliverable:	(T0+24)
Actual submission date:	November 14, 2014

Start date of the project: **1st November 2012**	Duration: **48 months**
Lead contractor for this deliverable:	**UiO**
Dissemination level:	**PU** – Public

Final version

Executive Summary:
Joint Phase 2 Evaluation and Phase 3 Requirement Analysis

This document summarises deliverable D1.3 of project FP7-318338 (Optique), an Integrated Project supported by the 7th Framework Programme of the EC. Full information on this project, including the contents of this deliverable, is available online at `http://www.optique-project.eu/`.

D1.3 reports on the results of the usability and performance evaluation process during Project Year 2 and on the requirements for Project Year 3.

List of Authors

Dimitris Bilidas (UoA)
Martin Giese (UiO)
Davide Lanti (FUB)
Marius Mikalsen
Ralf Möller (TUHH)
Christian Neuenstadt (TUHH)
Rudolf Schlatte (UiO)
Martin G. Skjæveland (UiO)
Ahmet Soylu (UiO)

Contents

List of Requirements

Chapter 1

Introduction

This report contains the result of the second year usability and performance evaluation and the third-year requirements elicitation activities of the Optique project. The evaluation activities are presented in Chapter 2, subdivided into usability evaluation (Section 2.2) and performance evaluation (Section 2.1).

Chapter 3 presents the result of the requirements elicitation process, subdivided into end-user requirements (Section 3.1), scientific requirements (Section 3.2) and project-internal requirements (Section 3.3). The end-user requirements were gathered during the end-user workshops held in cooperation with Work Packages 8 and 9, the scientific requirements are per the Optique Description of Work.

Chapter 2

Evaluation

This chapter describes the evaluation activities that the project undertook during the second project year. Section 2.1 summarizes the results of performance evaluation of different components of the Optique system, Section 2.2 describes the results of a number of usability studies of the visual query interface.

2.1 Performance Evaluation

Section 2.1.1 summarizes published results on performance evaluation of OBDA systems. Further discussion on performance in Ontop can also be found in Deliverable D6.2. Section 2.1.2 contains a discussion of the WP9 *query catalog*. Section 2.1.3 contains an evaluation of streaming data processing.

2.1.1 The NPD Benchmark for OBDA Systems

To properly evaluate the performance of OBDA systems, benchmarks tailored for the specific requirements and use-cases for such systems are needed. OWL benchmarks [3, 1, 7], which have been developed to test the performance of generic SPARQL query engines, however, fail at

- exhibiting a complex real-world ontology,
- providing challenging real-world queries,
- providing large amounts of real-world data, and the possibility to test a system over data of increasing size, and
- capturing important OBDA-specific measures related to the rewriting-based query answering approach in OBDA.

In order to overcome these shortcomings, the Optique project with WP6 proposed in [9] the first benchmark specifically tailored for OBDA systems, which is based on an semantically annotated version of the Norwegian Petroleum Directorate (NPD) FactPages,[1] containing ontology, mappings, and queries [14]. In particular, the queries have been selected by users of the NPD FactPages, whereas the dataset consists of a MySQL database instance obtained starting from real-world data. Some of these queries and the dataset are not directly suitable for use as an OBDA benchmark. The reasons are that (i) most of the queries are not supported by any currently implemented OBDA system, and (ii) the dataset is too small for any useful analysis. As for point (i), we removed unsupported constructs from the original queries (e.g., aggregate functions) whenever this was possible, while for point (ii) we implemented a *generator* able to produce datasets of increasing sizes starting from characteristics learned from an initial seed, mappings, and ontology. The result of this work is what we presented under the name of *NPD Benchmark*, and it is available online.[2]

[1] `http://factpages.npd.no/factpages/`

[2] `https://github.com/ontop/npd-benchmark`

O1 The ontology should include rich hierarchies of classes and properties.

O2 The ontology should contain a rich set of axioms that infer new objects and could lead to inconsistency, in order to test the reasoner capabilities.

M1 The mappings should be defined for elements of most hierarchies.

M2 The mappings should contain redundancies, and sub-optimal SQL queries to test optimizations.

Q1 The query set should be based on actual user queries.

Q2 The query set should be complex enough to challenge the query rewriter.

D1 The virtual instance should be based on real-world data.

D2 The size of the virtual instance should be tunable.

S1 The languages of the ontology, mapping, and query should be standardized, i.e., based on the standards OWL, R2RML, and SPARQL, respectively.

Table 2.1: Benchmark requirements for ontologies, mappings, queries, datasets and languages.

Another contribution of our work has been to give a first "requirements-driven" characterization of meaningful benchmarks for OBDA systems. The result of this work is synthesized in Table 2.1. None of the currently available OWL benchmarks fully satisfy these requirements, whereas the NPD Benchmark satisfies all of them. This fact led to tests that gave interesting results [9] about the actual applicability of OBDA in real-world scenarios, as they mostly contradict the ones obtained by using other common benchmarks. In particular, we witnessed the fact that real-world mapping assertions are so complex that they pose a serious challenge on the performance of an OBDA system. For example, it is quite common to have classes for which a high number of SQL queries is required in order to retrieve all of their instances; this fact easily leads to an *unfolded* SQL query whose size is an exponential of the size of the original SPARQL query, as witnessed by the size of the unfolding given in terms of number of SQL queries in Table 2.2.

These observations led to the question whether it is possible to further push the optimizations performed by Ontop and which make use of metadata information. Current, yet unpublished, research suggests that this is the case when hidden dependencies on the physical data are manually imposed by domain experts, e.g., when functional dependencies are completely specified. It is often the case that these dependencies are not explicitly specified in the database schema, because they have a strong effect on the performance (e.g., SQL inserts become quite slow in presence of foreign keys). However, letting Ontop know about them (without having to change the database schema) allows the query optimization techniques to perform a much deeper optimization, often simplifying exponential queries into very short ones.

Table 2.2: Hard Queries, Rewriting And Unfolding

	Ext. reasoning disabled				Ext. reasoning enabled			
	No. of SQL queries		Time, sec.		No. of SQL queries		Time, sec.	
query	rewrite	unfold	rewrite	unfold	rewrite	unfold	rewrite	unfold
q6	—	48	—	0.1	73	1740	1.8	1.3
q9	—	570	—	0.1	1	150	0	0.03
q10	—	24	—	0.9	1	24	0	0.01
q11	—	1	—	0.1	73	870	0.03	0.7
q12	—	1	—	0.2	10658	5220	525	139

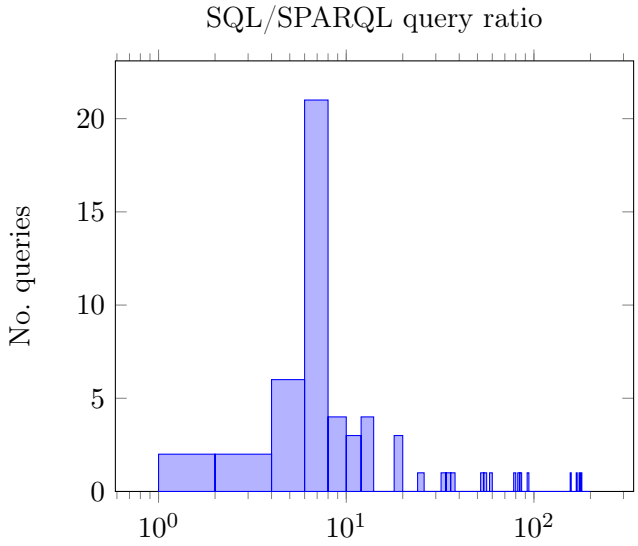

Figure 2.1: Effect of rewriting and unfolding wrt. query size. The diagram shows the number of queries according to times the increase of the generated SQL query compared to the size of the input SPARQL query.

2.1.2 Evaluation of the Statoil Query Catalog

An important part of the Optique project is to evaluate the Optique platform on real information needs collected from end-users. In the Statoil Use Case, WP9, we have collected different queries or information needs from Statoil personnel. Parts of these have been translated to SPARQL queries, in some cases, one end user information need is translated into multiple SPARQL queries that describe parts of the original need. The user information needs and SPARQL queries have been compiled into a *query catalog*, and all the SPARQL queries in the catalog can be automatically executed on different configurations of the Optique platform software, using different versions of ontologies, mappings and possibly also data sources. The query catalog is included in full in delivery D9.2, together with descriptions of the ontologies, mappings and databases that are relevant for the Statoil Use Case. Also, a test run of the query catalog using the latest ontology and mappings against the most important data store in the Statoil Use Case is included. This section presents a summarized version.

Currently, the query catalog contains 63 end-user information needs which are formulated in natural language English. Parts of these are translated into a total of 60 SPARQL queries. All of these 60 queries are executed in the test run. 11 of the queries do not return any answers when the timeout threshold is set to 3600 seconds = 1 hour, and three queries do not currently execute because of errors in the query rewrite process. Figure 2.1 shows the increase in size of queries from the input SPARQL query to the generated SQL query which is executed over the underlying database. The figure shows that a large part of the input queries are rewritten into SQL which is at most 10 times larger than the input. The largest increase in size of a query is a 179 times increase compared to the input; the possible reasons for an increase in size of this magnitude are discussed in D9.2 and D6.2.

Figure 2.2 shows the execution time of the queries in the query catalog; one plot shows the unfolding time, and the other plot query execution time. Note that the unfolding time is given in milliseconds and the execution time in seconds, i.e., the unfolding time is negligible for the total query answering time; the longest unfolding time is 290 milliseconds. As for the execution time, again with a timeout set to 3600 seconds, 23 queries return answers within 10 seconds, 16 queries return answers within between 10–100 seconds, six queries return answers within between 100–1000 seconds, and one query returns answers after more than 3000 seconds.

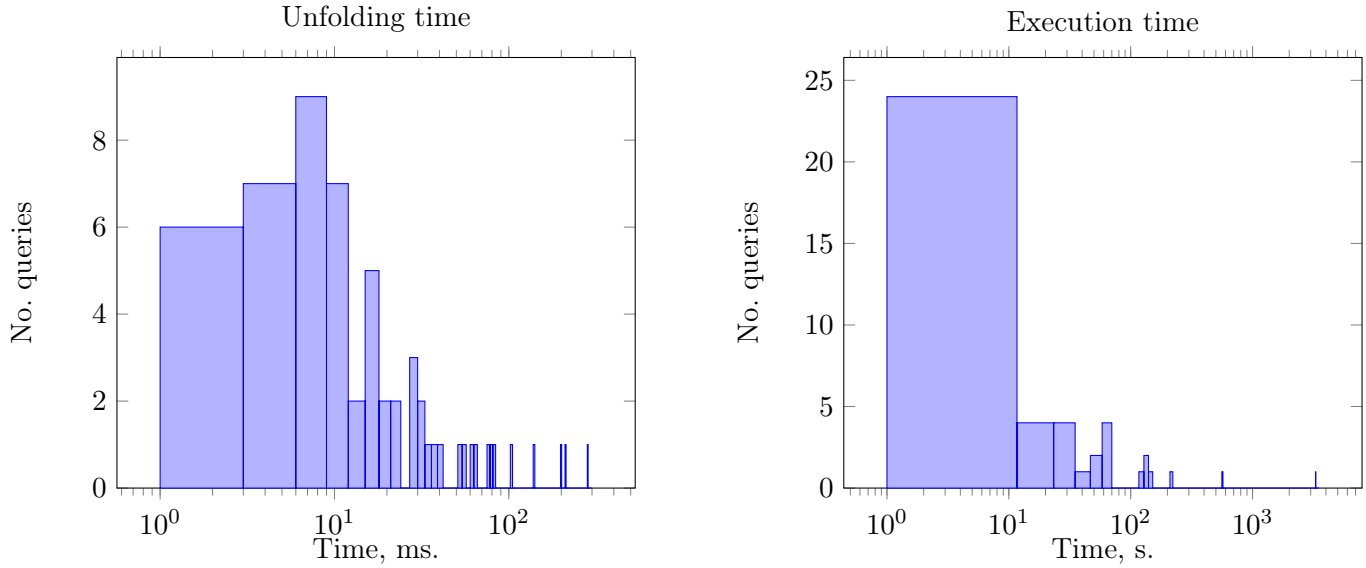

Figure 2.2: Query answering time. The diagrams show the number of queries according to unfolding time (in milliseconds) and execution time (in seconds), both on a logarithmic scale.

2.1.3 Evaluation of Query Answering on Siemens Historical Data

In this section we report on evaluation results related to the reactive diagnosis scenario within the Siemens power plant use case. In order to understand the reasons for some events such as start failures or unplanned shutdowns of turbines, the engineer has to analyze the timestamped data that are recorded by the sensors and that are stored in a central database. In most cases, the analysis involves constructing and answering a couple of queries that refer to a finite closed time interval, a.k.a. a window, over the historical data. For this kind of analysis the engineer can be supported by the query language STARQL, a query language framework that uses this central concept of a window in order to slide over data—either in real time for continuous query processing over real-time streams or in simulated possibly hyper-real time over historical data. (See Deliverable D5.1 regarding the formal definition of STARQL and Deliverable D5.2 for aspects of rewriting and unfolding). There is no conceptual difference in doing the one or the other—except for the need to specify the starting date and the end date for the portion of the historical data to taken into account. In the following, we state the experiment setups and results for the implemented STARQL engine, in particular we report on the performance results of query answering over the Siemens historical data.

The system is evaluated along two example queries formulated in STARQL. Following the OBDA paradigm, the STARQL engine transforms the queries w.r.t. some predefined mappings into queries in a SQL-like target language. In fact, we evaluated the STARQL w.r.t. two target languages. The first language, SQLite extended with additional python functions, is the one that is provided by the ADP (Athena Distributed Processing) system from the Optique partner UoA. The second target language is the SQL variant as provided by PostgreSQL databases. Both instances of the STARQL engine are running on a local computer. As the ADP system is designed for distributed computing and not for running on a local machine, we expect highly increased performance in the next evaluation step for distributed queries.

In the following we briefly introduce the used datasets, afterwards we show the specific STARQL queries and their transformations, and finally we discuss our evaluation results on both systems.

The Datasets

The datasets we use are part of the Siemens use case in Optique. We give a short overview on the data contained in the datasets. For more information, please consider Deliverable D1.1 and Deliverable D8.1.

The original data processed/produced by Siemens appliances are sensor measurements, event data, opera-

tion logs, and some other data stored in further tables. These data are confidential and are used internally at Siemens. Siemens provided a small public dataset and two larger anonymized datasets for use inside Optique. The public dataset has a simplified structure and has a size of approximately 100 MB. For the evaluation presented here we used two datasets, denoted Dataset1 and Dataset2. Dataset1 contains public data and has a size of approximately 546 KB. Dataset2 contains anonymous data and has a size of approximately 14,6 MB.

In order to explain the requirements on the expressivity of the query language later on, we first take a look at the table definitions. The schema we discuss is normalized such that redundancies are avoided as far as possible, but we note the denormalization is not supposed to give significant speed-ups. Measurements are represented with a table `measurement` and consist of a timestamp, a reference to the sensor involved, and a measurement value.

```
CREATE TABLE measurement (
  timestamp timestamp without time zone NOT NULL,
  sensor integer NOT NULL,
  value numeric(12,3) NOT NULL,
  CONSTRAINT measurement_pkey PRIMARY KEY ("timestamp", sensor),
  CONSTRAINT measurement_fk_sensor FOREIGN KEY (sensor)
      REFERENCES sensor (id) MATCH SIMPLE );
```

A measurement datum has timestamp, sensor, and value. For our evaluation we are using one dataset of about 80,000 entries with a timestamp ranging over 3 days (Dataset1) and another dataset with about 400,000 entries with a timestamp ranging over 5 years (Dataset2). Both datasets contain data referring to a number of sensors.

```
CREATE TABLE sensor (
  id serial NOT NULL,
  assemblypart integer,
  name character varying(20),
  CONSTRAINT sensor_pk PRIMARY KEY (id)
  CONSTRAINT fk_assemblypart_sensor FOREIGN KEY (assemblypart)
             REFERENCES assemblypart (id) MATCH SIMPLE );
```

The values of the attribute `name` hint at the purpose/function of the sensor, e.g., the *"Inlet Pressure"* says that the sensor measures a particular pressure. A few sensors with the same functions are distinguished by a numerical index, e.g., *"Burner Temperature 1"*, *"Burner Temperature 2"*. Within the OBDA paradigm, the string values for the attribute are mapped to concepts, querying for specific sensors reduces to asking for instances of corresponding concepts.

The STARQL Queries and their Transformations

The two example STARQL queries for the evaluation are given in the following.

The first STARQL query (Figure 2.3) builds, within each generated window, a sequence of all sensor values in the last 24 hours and checks whether one of the sensors shows a monotonic increase during this period of time. This query is expected to run quadratically slower as window size increases, due to the comparisons of all pairs of values ?x and ?y for all pairs of states (i,j).

The second query (Figure 2.4) outputs, every minute, sensors that show a value higher than 90. This query is expected to be faster because of its simple window content with at most one timestamp. So the sequence has a simple structure as it may contain at most one ABox, and no values are compared over this sequence as there is only one quantifier in the `HAVING` clause.

Both STARQL queries can be transformed into queries in an SQL variant, using mappings which are not shown here. As discussed above, we consider the transformation into queries into ADP-SQL (SQLite with

```
CREATE STREAM S_out1 AS
SELECT { ?sensor rdf:type :RecentMonInc }< NOW>
FROM burner_regulator 0 seconds <- [ NOW - 24 hours, NOW ]-> 24 hours
SEQUENCE BY StdSeq AS stateSequence
HAVING FORALL i, j IN stateSequence
FORALL ?x,?y
IF  { ?sensor :hasVal ?x }<i> AND  { :Regulator :hasVal ?y } <j> AND i < j
THEN ?x <= ?y) ELSE TRUE
```

Figure 2.3: STARQL query Query1

```
CREATE STREAM S_out2 AS
SELECT { ?sens rdf:type :tooHigh }<NOW>
FROM burner_3 0 seconds <- [ NOW , NOW ]-> 1 minute
SEQUENCE BY StdSeq AS stateSequence
HAVING FORALL i IN stateSequence
FORALL ?x
IF  { ?sens :hasVal ?x }<i> THEN ?x > 90) ELSE TRUE
```

Figure 2.4: STARQL query Query2

additional python functions) and into PostgreSQL. So, for each query we have two corresponding transformations. The transformations into ADP-SQL for the first and the second query are shown in Figure 2.5 and Figure 2.6, resp.

The common structure of both ADP-SQL queries is the following: In the first part of the queries, a stream of windows is generated from the data using the `timeslidingwindow` operator. This operator has three window parameters: `timewindow` for the width, `frequency` for the slide, and `granularity` for the step width between each timestamp. The window stream is saved in a table with a specific index for better performance. In the second part, the generated window data are used within the `having` view, which basically is an implementation of the STARQL `HAVING` clause. More information about the transformation can be found in Deliverable D5.2, Deliverable D7.2 reports on the usage of the ADP system.

```
ATTACH DATABASE 'database' as 'db';
DROP TABLE IF EXISTS burner_day;

CREATE TEMP TABLE burner_day AS
SELECT * FROM
   (timeslidingwindow timecolumn:0 timewindow:3600 frequency:60 granularity:60
    equivalence:floor select distinct * from db.publicdata)
WHERE wid > 0;

CREATE INDEX burner_day_index ON burner_day(wid, abox, value, sensor);
CREATE TEMP VIEW burner AS  SELECT * FROM burner_day;

CREATE TEMP VIEW S_out_2_having AS
SELECT burner.wid as wid0, burner."sensor" AS _sens, max(burner.timestamp) AS now
FROM burner
WHERE NOT EXISTS(
SELECT * FROM
  (SELECT * FROM
    (SELECT wid AS wid1, burner."sensor" AS _sens1, burner."value" AS _x, abox as i
        FROM burner
        WHERE burner."value" IS NOT NULL) as triple1,
     (SELECT wid AS wid2, burner."sensor" as _sens2, burner."value" AS _y, abox as j
         FROM burner WHERE burner."value" IS NOT NULL) AS triple2
         WHERE wid1 = wid2 )
    WHERE wid0 = wid1 AND wid1 = wid2 AND _x > _y AND i < j AND _sens1 = _sens2
          AND _sens = _sens1)
AND burner."sensor" IS NOT NULL
AND burner."value" IS NOT NULL
GROUP BY wid0, _sens;

CREATE TEMP VIEW S_out_2_starqlout AS
SELECT DISTINCT now AS timestamp, _sens AS Subject, 'rdf:type' AS Predicate,
               ':RecentMonInc' AS Object
FROM S_out_2_having;

SELECT * FROM S_out_2_starqlout;
```

Figure 2.5: Query1 in ADP-SQL

```
ATTACH DATABASE 'database' AS 'db';

CREATE TEMP TABLE burner_3 AS
SELECT * FROM
  (timeslidingwindow timecolumn:0 timewindow:60 frequency:60 granularity:1
   equivalence:floor select * from db.publicdata);

CREATE INDEX burner_index ON burner_3(wid, abox, value, sensor);
CREATE TEMP VIEW burner AS  SELECT * FROM burner_3 WHERE sensor IS NOT NULL;

CREATE TEMP VIEW S_out_having AS
SELECT burner.wid AS wid0, burner.sensor AS s0, burner.value, burner.timestamp AS now
FROM burner WHERE NOT EXISTS(
  SELECT * FROM
   (SELECT burner."value" as _x, abox AS i, wid AS wid1, burner.sensor AS s1 FROM burner
    WHERE burner."value" IS NOT NULL AND _x <= 90) AS triple1
 WHERE wid0 = wid1 AND s0 = s1 AND burner."value" IS NOT NULL;

CREATE TEMPW VIEW S_out_starqlout AS
SELECT now AS timestamp, s0 AS Subject, 'rdf:type' AS Predicate, ':tooHigh' AS Object
FROM S_out_having;

SELECT * FROM S_out_starqlout;
```

Figure 2.6: Query2 in ADP-SQL

The transformations of the STARQL queries into PostgreSQL are show in Figure 2.7 and in Figure 2.8.

The basic structure of the PostgreSQL queries is similar to that of the ADP-SQL queries. Windows are not generated here by an additional operator, but instead by three different views which build a window structure and which are joined afterwards with the data table in the `val` view. The two next views `q1` and `q3` are adding filtering information of the STARQL `HAVING` clause, and finally the results are summed up. For more information on the query transformation, especially for the window generation in the PostgreSQL

```
CREATE OR REPLACE VIEW window_range AS
SELECT row_number() OVER (ORDER BY x.timestamp) - 1 AS wid,
       x.timestamp as left, x.timestamp + '1 hour'::interval as right
FROM  (SELECT generate_series(MIN(mp.timestamp),
             MAX(mp.timestamp), '1 hour'::interval)  AS timestamp FROM measurement mp) x;

CREATE OR REPLACE VIEW wid AS
SELECT distinct wid, timestamp
FROM measurement mp, window_range w
WHERE mp.timestamp >= w.left and mp.timestamp < w.right;

CREATE VIEW win AS
SELECT wid, rank() OVER (PARTITION BY wid ORDER BY timestamp ASC) as ind,
       timestamp FROM wid;

CREATE VIEW val AS
SELECT DISTINCT rel1.WID , rel2.SID , rel2.VALUE , rel1.ind
FROM win rel1 , measurement rel2
WHERE rel2.timestamp = rel1.timestamp
ORDER BY wid, ind;

CREATE VIEW sensors AS
SELECT rel1.WID , rel1.SID
FROM val rel1;

CREAT VIEW q3 AS
SELECT rel1.WID , rel1.SID AS S
FROM val rel1 , val rel2
WHERE rel2.WID = rel1.WID AND rel2.SID = rel1.SID AND
    rel1.ind < rel2.ind AND
    rel1.VALUE > rel2.VALUE;

CREATE VIEW q1 AS
SELECT rel1.WID , rel1.SID AS S
FROM sensors rel1
WHERE NOT EXISTS(SELECT *
                 FROM q3 rel2
                 WHERE rel2.WID = rel1.WID AND rel2.S = rel1.SID);

CREATE VIEW s_out AS
SELECT rel2.right, rel1.S AS SID
FROM q1 rel1, window_range rel2
WHERE rel1.WID = rel2.WID;

SELECT DISTINCT s.right as timestamp, s.SID as Subject, 'rdf:type' as Predicate,
               ':recentMonInc' from s_out;
```

Figure 2.7: Query1 in PostgreSQL

```
CREATE VIEW window_range AS
SELECT row_number() OVER (ORDER BY x.timestamp) - 1 AS wid,
       x.timestamp as left, x.timestamp as right
FROM (SELECT generate_series(MIN(mp.timestamp), MAX(mp.timestamp), '1 minute'::interval)
        AS timestamp FROM measurement_sample mp) x;

 CREATE VIEW wid AS
 SELECT DISTINCT w.wid,
    mp.timestamp
   FROM measurement_sample mp, window_range w
   WHERE mp.timestamp between w.left and w.right;

CREATE VIEW win AS
SELECT wid, rank() OVER (PARTITION BY wid ORDER BY timestamp ASC) as ind, timestamp
FROM wid;

CREATE VIEW val AS
SELECT DISTINCT rel1.WID , rel2.SID , rel2.VALUE , rel1.ind
FROM win rel1 , measurement_sample rel2
WHERE rel2.timestamp = rel1.timestamp;

CREATE VIEW sensors AS
SELECT rel1.WID , rel1.SID
FROM val rel1;

CREATE VIEW q3 AS
SELECT rel1.WID , rel1.SID AS S
FROM val rel1
WHERE rel1.VALUE <= 90;

CREATE VIEW q1 AS
SELECT rel1.WID , rel1.SID AS S
FROM sensors rel1
WHERE NOT EXISTS ( SELECT *
                   FROM q3 rel2
                   WHERE rel2.WID = rel1.WID AND rel2.S = rel1.SID ) ;

CREATE VIEW s_out AS
SELECT rel2.right, rel1.S AS SID
FROM q1 rel1, window_range rel2
WHERE rel1.WID = rel2.WID;

SELECT DISTINCT s.right as timestamp, s.SID as Subject,
                'rdf:type' as Predicate, ':tooHigh'
FROM s_out;
```

Figure 2.8: Query2 in PostgreSQL

case, see Deliverable D5.2.

Test Results and Evaluation

For the evaluation part we first look at the transformation time from STARQL to ADP-SQL. Both queries can be transformed in approximately 1 second (1086 ms and 1160 ms)We believe that a transformation time with less than a second can be achieved in Year 3.

Additionally, we tested all four query transformations on the specific systems and data sets.The tests have been made in a virtual machine on a system with an i7 2.8 GHz CPU and 16 GB of ram. Mean values of several test runs with cold cache are shown in Table 2.3.

The tests reported in Deliverable 5.2 were conducted with a non-optimized version of the ADP system, as this was the last stable version at hand at that time. For the tests reported here, we used the last stable and optimized version of the ADP system from 2014. The 2014 version uses additional indexes on the window table. This important optimization increased the speed from minutes to seconds for some queries (see below).

Dataset	Test query	Test system	Time
Dataset1	1	PostgreSQL	45 sec
Dataset1	1	ADP-SQL	4.7 sec
Dataset1	2	PostgreSQL	1.2 sec
Dataset1	2	ADP-SQL	4.6 sec
Dataset2	1	PostgreSQL	50 sec
Dataset2	1	ADP-SQL	44.9 sec
Dataset2	2	PostgreSQL	30.6 sec
Dataset2	2	ADP-SQL	31.7 sec

Table 2.3: Query response times for ADP and PostgreSQL transformations

As shown in Table 2.3, all results are in the range of seconds. Query1 runs on the ADP system faster than on to the PostgreSQL system. For Query2 it is the other way around on both datasets. A crucial point is the gap for query1 on the public dataset. We think that this indicates that the ADP system is more capable of handling larger sequences of values. Dataset1 is the only dataset with full sequences per window, where the one-day windows of Dataset2 are not filled up.

All in all we can say that PostgreSQL and the ADP do not show significant differences regarding performance for the queries and datasets we evaluated. The ADP system used in Optique is even ahead several times. We believe that the speed can be increased significantly in the context of distributed computations. An even more optimized system will be shown in the next years' evaluation phase.

An additional set of anonymous data (internally named anonymous 1) provided by Siemens, having a size of 2,25GB, was also tested w.r.t. the ADP and PostgreSQL transformations, but the engines were not able to handle this test set in a reasonable amount of time. We expect to report on query results on this data set in Year 3; a concrete requirement to this regard was added in Chapter 3.

We are going to extend the performance evaluations in the near future along the following lines: Experiments both on historical data and streaming data can be run on the anonymized dataset Dataset2. Experiments for historical data can be run on a local ADP system; the plan is to compare various queries in SQL against their transformed STARQL counterparts.

Experiments on streaming data can be run on the simulation server installed by FluidOps. Performance parameters to be taken into account include the number of windows that can be evaluated, varying the number of parallel streams and the window size. Tests will have to be adapted w.r.t. code and installation changes of the simulation server.

2.2 Usability Evaluation

Usability evaluation (cf. [2]) is an integral part of research on visual query formulation. A set of *evaluation criteria* is required to assess if a visual query system (VQS) is competent of meeting its identified aim. VQSs are meant to enable users to formulate queries effectively. The *effectiveness* (cf. [4, 2]) is measured in terms of *accuracy* and *completeness* that users can achieve (i.e., "doing the right things"). However, on the one hand, the cost associated with the level of effectiveness achieved is a significant factor, called *efficiency* (cf. [4, 2]), and is mostly measured in terms of the time spent to complete a query (i.e., "doing the things right"). On the other hand, the perceived quality of dialog and user interface, called *user satisfaction* [10], which determines the attitude of users, such as *trust*, *engagement*, *acceptance*, and *comfort*, against a VQS or VQL, has considerable potential to cause failure, if not taken seriously. User satisfaction is usually measured through surveys, interviews etc. [10, 6] investigating the attitude of users after experiencing with the subject system or language.

User studies are typically realized by means of *query writing* and *query reading* tasks [5, 15] to evaluate the tool itself alone or to compare it with others with a *summative* or *formative* perspective; various other studies and measures (e.g., *learnability* [12]) can be defined for this purpose. However, consulting users only at the end (i.e., summative evaluation) does not help much for success [8, 4]. The active participation of users at every cycle of development, grounded on a *user-centered design* [13] approach with intermediary reflective assessments with small user groups (i.e., formative evaluation), is the true contributor.

In this respect, OptiqueVQS has been evaluated in three intermediary formative experiments. The first one has been conducted with 15 casual users on an example movie ontology, the second one at Statoil with 3 users on the NPD ontology, and the third at Siemens with 4 users on a Siemens diagnostic ontology. The first experiment tested the usability of OptiqueVQS on a general domain with non-domain experts so as to avoid any domain-specific effect issues. The second and third experiments were conducted over our use-case-specific domains with smaller user groups. Remark that the smaller sample sizes in the second and third experiments are not problematic, since in intermediary formative evaluations the number of users is typically not large;as Nielsen [11] suggests, 3-6 users will discover about 70-90 percent of the usability problems and at least 15 users will be needed to discover all the usability problems. In the following subsections, we report on all three experiments in detail.

2.2.1 Evaluation of the Visual Query Interface

The experiment was designed as a *think-aloud* study, since the goal of the experiment is not purely *summative*, but to a large extent *formative*. The experiment is built on a "movie ontology." An excerpt of the vocabulary for the music domain used is given in Figure 2.9; note that inverse properties are omitted in the figure for the sake of brevity. In total, the ontology includes 6 concepts, 16 relationships (including inverse properties), and 17 attributes. An ontology of this size already allows us to design complex queries. We avoided having a larger ontology in order to omit the effect of ontology size on the query formulation for this experiment.

A total of 15 participants took part in the experiment; the profiles of participants are summarized in Table 2.4. We selected our participants particularly among non-technical people, since they are the primary target of OptiqueVQS. A five-minute introduction of the topic and tool was delivered to the participants along with an example, then they were asked to fill in a profile survey. The survey asks users about their *age*, *occupation* and *level of education*, and asks them to rate their *technical skills*, such as on programming and query languages, and their *familiarity with similar tools* on a *Likert scale* (i.e., 1 for "not familiar at all," 5 for "very familiar"). Participants were then asked to formulate a series of information needs as queries with OptiqueVQS, given at most three attempts for each query. Each participant performed the experiment in a dedicated session, while being watched by an observer. Participants were instructed to think aloud, including any difficulties they encountered (e.g., frustration and confusion), while performing the given tasks.

Table 2.5 shows the 6 tasks representing the information needs used in the experiment. Each information need maps to a query at different level of complexity with respect to its topology and length, in an increasing order of complexity (all conjunctive): *short linear (T1)*, *long linear (T2)*, *short with branching (T3)*, *long*

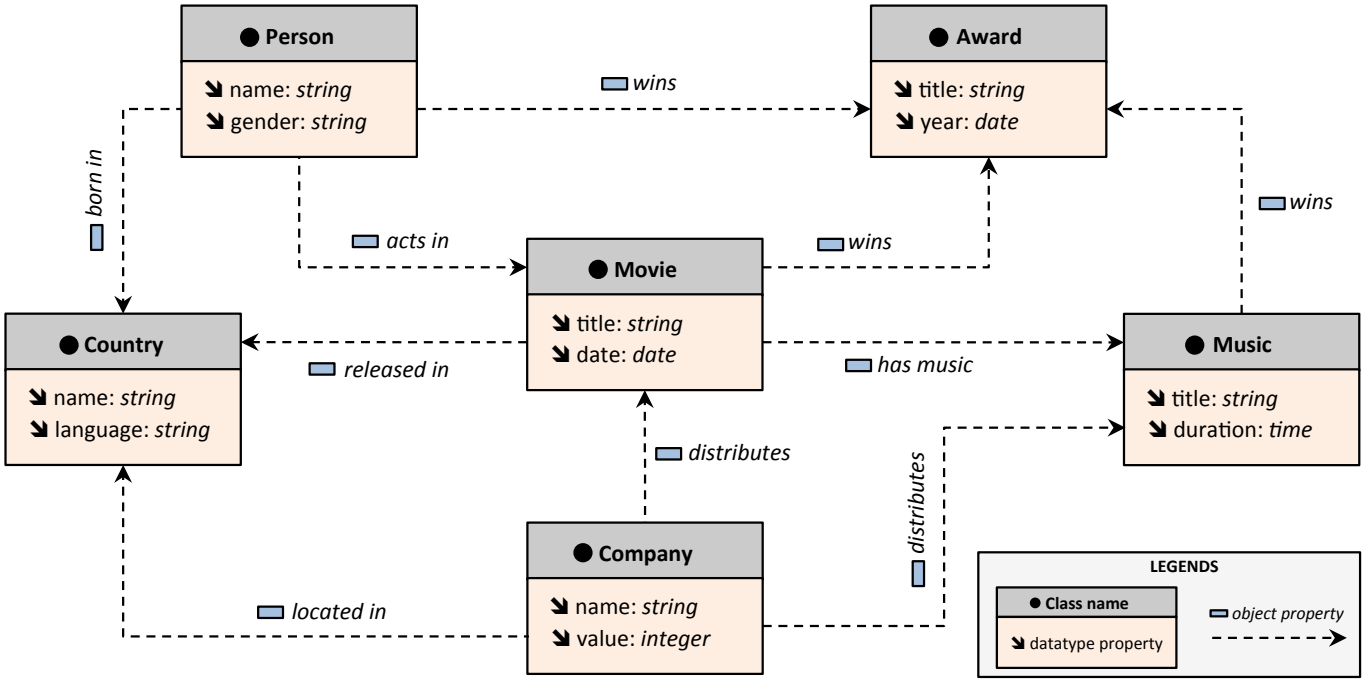

Figure 2.9: An excerpt of the vocabulary for the movie domain used..

Table 2.4: Profile information of the participants.

#	Age	Occupation	Education	Technical skills	Similar tools
P1	32	Chemist	PhD	2	3
P2	26	Math teacher	Bachelor	1	1
P3	43	Law student	Master	1	1
P4	21	Political science student	Bachelor	1	2
P5	22	Criminology student	Bachelor	1	3
P6	31	Hydrology student	Master	2	4
P7	26	Complex systems student	Master	2	3
P8	23	Psychology student	Bachelor	1	3
P9	24	Finance student	Bachelor	2	3
P10	21	Law student	Bachelor	2	2
P11	21	Law student	Bachelor	1	1
P12	21	Biology student	Bachelor	1	1
P13	23	Natural sciences student	Bachelor	1	1
P14	24	History student	Bachelor	1	3
P15	22	Biology student	Bachelor	1	1
Avg.	25	-	-	1.3	2.1

Table 2.5: Information needs used in the experiment.

#	**Query type**	**Information need**
T1	Short linear	Find the *names* of all the *companies* that *distribute* a *movie titled* "Titanic".
T2	Long linear	Find the *names* of all the *people* who *acted in* a *movie released between* 1970 and 1980 and *distributed by* a *company located in* Germany.
T3	Short with branching	Find the *titles* of all the *musics* that *won* an *award titled* "Best of Movie Musics" and are *played in* a *movie titled* "The Red Warrior".
T4	Long with branching	Find the *titles* of all the *movies* that are *distributed by* a *company owned by* a *person born in* USA and *have* a *music* that *won award between* 1980 and 1990.
T5	Short with branching and type III cycle	Find the *names* of all the *companies* that *distribute* a *movie titled* "Titanic" and *distribute* a *music played in* a *movie released in* 1980.
T6	Long with branching and type III cycle	Find the *titles* of all the *musics distributed by* a *company located in* the UK and *played in* a *movie* that *has* an actor named "George" who was *born in a country* in the African continent and *won* an *award* in 1990.

with branching (T4), *short with branching and type III cycle (T5)*, and *long with branching and type III cycle (T6)*. Here type III cycle refers to repetition of concepts, i.e., the query includes at least one instance where a concept appears twice. Long queries are queries with a maximum tree depth of at least 3.

Once users were done with their tasks, they were asked to fill in an exit survey asking about their experiences with the tool. The survey asks users to rate whether the questions were easy to do with the tool (*S1*), the tool was easy to learn (*S2*), was easy to use (*S3*), gave a good feeling of control and awareness (*S4*), was aesthetically pleasing (*S5*), was overall satisfactory (*S6*), and was enjoyable to use (*S7*) on a Likert scale (again, 1 for "strongly disagree" and 5 for "strongly agree"). Users were also asked to comment on what they did like and dislike about the tool and to provide any feedback which they deemed important.

Results

The results of the experiment are presented in Table 2.6. A total of 90 tasks was completed by the participants with a 80 percent first-attempt correct completion rate (i.e., percentage of correctly formulated queries in the first attempt). On average a task took 74 seconds to complete on 1.2 attempts; the first task and fourth task took the shortest and the longest times to complete, on average 34 seconds and 93 seconds respectively. The third task had the highest average in the number of attempts with 1.5, while the first and the sixth tasks had the lowest average in the number of attempts with 1 and 1.1 respectively.

According to the results and our observations, participants solved the first task (i.e., short linear) quite easily. However, when it came to the third task (short with branching), half of the participants failed in their first attempt. This is particularly due to fact that they were mostly not expecting a branching after two linear queries and did not pay attention to the text of the information need. Yet, as soon as they realized the case, they did quickly recover and manipulated their queries accordingly. The average number of attempts then decreased for the subsequent tasks (i.e., all with branching) as users became more aware. The fourth task (i.e., long with branching) took the longest time on average, since after the third task participants paid more attention to clearly understanding the information need. Participants solved the fifth task (i.e., short with branching and type III cycle) comparatively quickly; this was due to the short length of the query and due to the fact that participants did not have any confusion, when a concept appeared twice in the query (only one participant had this confusion and raised it). Finally, participants solved the last task (i.e., long

Table 2.6: The results of the experiment (c for complete, t for time in seconds, and a for attempt count).

	T1			**T2**			**T3**			**T4**			**T5**			**T6**			**Avg.**		
#	c	t	a	c	t	a	c	t	a	c	t	a	c	t	a	c	t	a	c	t	a
P1	1	50	1	1	70	1	1	94	2	1	55	1	1	53	1	1	68	1	1	65	1.2
P2	1	48	1	1	83	1	1	80	1	1	113	2	1	60	1	1	70	1	1	76	1.2
P3	1	81	1	1	87	1	1	80	2	1	180	2	1	141	2	1	145	1	1	119	1.5
P4	1	18	1	1	44	1	1	41	1	1	124	2	1	73	1	1	90	1	1	65	1.2
P5	1	32	1	1	85	1	1	62	1	1	74	1	1	82	1	1	85	1	1	70	1.0
P6	1	16	1	1	136	2	1	125	2	1	86	1	1	108	1	1	100	1	1	95	1.3
P7	1	27	1	1	105	2	1	102	2	1	126	2	1	122	2	1	135	1	1	103	1.7
P8	1	75	1	1	47	1	1	78	2	1	54	1	1	48	1	1	71	1	1	62	1.2
P9	1	23	1	1	59	1	1	54	1	1	82	1	1	45	1	1	81	1	1	57	1.0
P10	1	14	1	1	54	1	1	41	1	1	73	1	1	47	1	1	80	1	1	52	1.0
P11	1	17	1	1	42	1	1	65	1	1	53	1	1	105	2	1	60	1	1	57	1.2
P12	1	29	1	1	72	1	1	84	2	1	103	1	1	56	1	1	83	1	1	71	1.2
P13	1	38	1	1	54	1	1	44	1	1	75	1	1	46	1	1	80	1	1	56	1.0
P14	1	28	1	1	96	1	1	65	1	1	58	1	1	54	1	1	60	1	1	60	1.0
P15	1	19	1	1	125	2	1	112	2	1	144	1	1	50	1	1	168	2	1	103	1.5
Avg.	1	34	1	1	77	1.2	1	75	1.5	1	93	1.3	1	72	1.2	1	91	1.1	1	74	1.2

with branching and type III cycle) quite smoothly and with confidence, although it was the longest and the most complex one (i.e., with two branches and one type III cycle). A snapshot from the final query is given in Figure 2.10.

The feedback provided by the participants through the exit survey is presented in Table 2.7 and Table 2.8. Participants overall rated the tool good with 4 out of 5 on average. The first statement (cf. S1 – the questions were easy to do with the tool) had the lowest rank with 3.7; according to our observations, this was mostly due to the texts of the information needs, rather than the tool. The texts describing the information needs (cf. Table 2.5) include a number of relative pronouns along with a passive sentence structure, which make them hard to understand at a first glance and to keep in short-term memory. Although this structure was intentionally selected in order to avoid a step-by-step question form, for subsequent evaluations a different form could be considered. As listed in Table 2.8, participants mainly found the tool orderly. Participants liked the way that queries were visualized, i.e., a diagrammatic overview that users can interact with. They also appreciated the fact that the tool allows them to formulate detailed information needs easily and in an organized way. The introduction given to the users were only around five minutes with an example query, therefore participants were mostly expected to learn on the way, since one of our goals was to have a tool with a low learning curve and effort. This case is reflected and confirmed by the comments of participants.

Observing the participants in action allowed us to acquire some specific insights about the tool. One major issue was that while formulating the fourth task, participants initially looked for a "birth place" field in W2, since the information need was specifying a person born in USA. It took only a while for participants to realize that this information is only accessible through a relationship rather than an attribute. A participant first considered the branches as "OR" rather than "AND" and asked whether it was possible to construct "OR" branches. Two participants realized that indeed they do not have to follow the logical order given in the descriptions of information needs (i.e., to join the concepts in the given order), but the alternatives exist. One of these participants solved one of the tasks successfully with an alternative order. Finally, from a general perspective, users did not have any major difficulties in using and learning the tool and were quick in realizing the given tasks. Participants largely stated that their experience with the tool was comparable to the games in terms of the joy they had, raised their interest on the tool, and asked further questions after the experiment, mostly concerning the context that the tool is going to be used.

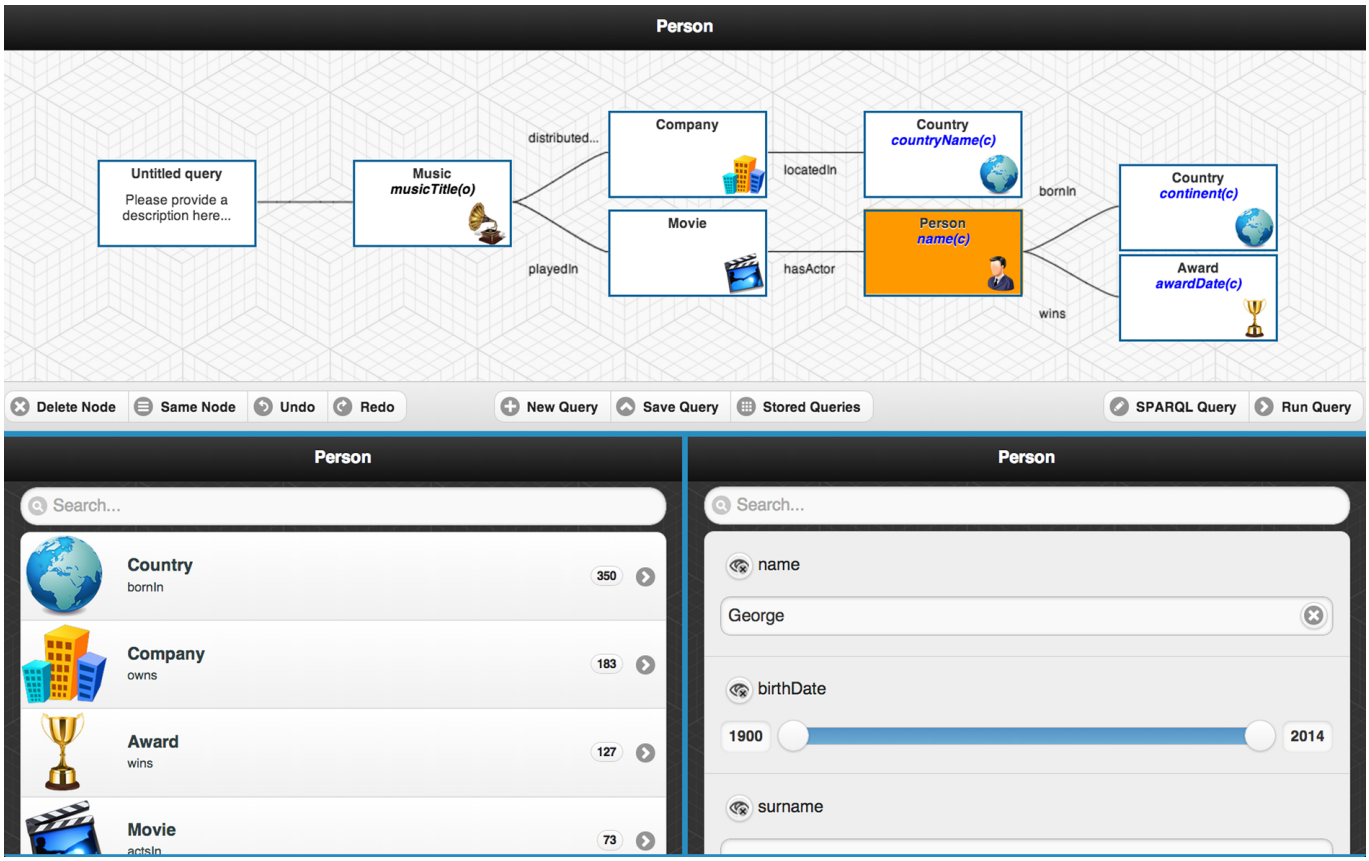

Figure 2.10: An excerpt from a query formulated by the participants during the experiment.

Table 2.7: The results of the exit survey.

#	S1	S2	S3	S4	S5	S6	S7	Avg.
P1	5	5	5	5	4	4	5	4.7
P2	4	4	5	5	5	4	5	4.6
P3	4	4	4	4	4	4	4	4.0
P4	3	4	2	3	4	4	4	3.4
P5	4	4	4	4	4	4	5	4.1
P6	4	5	4	5	5	4	4	4.4
P7	3	3	4	4	4	4	4	3.7
P8	5	5	5	5	4	5	5	4.9
P9	3	4	4	4	4	4	4	3.9
P10	3	3	4	4	3	4	4	3.6
P11	3	4	4	4	4	4	4	3.9
P12	4	4	4	4	4	4	4	4.0
P13	4	3	4	4	4	4	4	3.9
P14	3	3	4	3	4	4	4	3.6
P15	4	4	3	3	4	4	4	3.7
Avg.	3.7	3.9	4.0	4.1	4.1	4.1	4.3	4.0

Table 2.8: The feedback given by the participants.

#	**Like**	**Dislike**
P1	“Visual, easy to use, fast and easy to correct mistakes."	“...can be visually improved."
P2	“Easy to jump on [diagram] and suggestions [of the W1] were relevant"	-
P3	“Easy and organized. Good for an organized and focused search."	“Nothing"
P4	“I like that it can make search process go faster and make it more specific"	“It could get complicated as you have to link and sometimes go back to previous boxes."
P5	“It was OK to find what the tasks asked for without having to look too long for the right variables."	-
P6	“The schematic diagram"	“It has fixed options."
P7	“Good overview"	-
P8	“The way you connect the nodes, the way it was easy to incorporate a lot of information in the right way, and it was easy to be organized."	“Maybe seems a bit simple at the first glance, but then it was good!"
P9	“Nice visualization [for diagram]"	“Many steps"
P10	“Easy to use"	-
P11	“It was quite simple."	“It felt I did not have much time [to learn]."
P12	“The organization in images and scheme"	-
P13	“The scheme on top is pretty helpful to see where you are actually getting to what you are looking for"	“It took me some time to get used to it, but then I think it works!"
P14	“You could really go in to details and ask many things about same person/company etc."	“It was a bit tricky to learn, but I think that it is possible to get a hang on it if you use it for a while"
P15	“Easy access for specific information regarding the search options: movies, music etc."	“Some difficulties [for] managing the correct search option"

Table 2.9: Profile information of the participants.

Question	**P1**	**P2**	**P3**
"Age"	39	40	49
"What is your occupation?"	Geologist	Biostrat	Senior IT Advisor
"What is your level of education?"	Master	Master	Master
"I have technical skills (i.e., computer) such as programming and query languages (e.g., SQL, Java, PHP, SPARQL etc.)"	Neutral (3)	Disagree (2)	Strongly Agree (5)
"I am familiar with tools similar to OptiqueVQS"	Neutral (3)	Strongly Disagree (1)	Agree (4)

2.2.2 Results of the Statoil End-User Workshop

The experiment for the Statoil end-user workshop was designed as a *think-aloud* study similar to the first experiment. The experiment is built on a bootstrapped NPD ontology. In total, the ontology includes 253 concepts, 208 relationships (including inverse properties), and 233 attributes.

A total of 3 participants took part in the experiment; the profiles of participants are summarized in Table 2.9. A five-minute introduction of the topic and tool had been delivered to the participants along with an example before they were asked to fill in a profile survey. The survey asks users about their *age*, *occupation* and *level of education*, and asks them to rate their *technical skills*, such as on programming and query languages, and their *familiarity with similar tools* on a *Likert scale* (i.e., 1 for "not familiar at all," 5 for "very familiar"). Participants were then asked to formulate a set of information needs into queries with OptiqueVQS, with at most three attempts for each query. Each participant performed the experiment in a single session, while being watched by an observer. Participants were instructed to think aloud, including any difficulties they encounter (e.g., frustration and confusion), while performing the given tasks.

There were 9 tasks, representing the information needs used in the experiment – see Table 2.10. The task characteristics and shape of queries followed the first experiment.

Once users were done with their tasks, they were asked to fill an exit survey asking about their experiences with the tool. The survey was built on the SUS and users were also asked to comment on what they did like and dislike about the tool and to provide any feedback which they deemed important.

Results

The results of the experiment are presented in Table 2.11. A total of 27 tasks was completed by the participants, with 84 percent correct completion rate and 65 percent first-attempt correct completion rate. The first user had only one incorrect, and the second user has no incorrect result. The third user has one missing record (task 2), therefore this has been omitted from the results. The third question was asking for fields operated by Statoil. Instead of formulating a Field - Company pair, the third user formulated a Field - FieldOperator pair. This confusion between FieldOperator and Company led him to incorrectly solve the task 5 as well.

In increasing order, the task completion times and number of attempts on average are: 53 s and 1 attempt for task 1, 90 s 1 attempt for task 2, 113 s and 1.3 attempt for task 5, 142 s and 1.6 attempts for task 3, 276 s and 1.3 attempts for task 4, 280 s and 1.6 for task 8, 391 s and 1 attempt for task 9, and 495 s and 2.3 attempts for task 7. The results suggest that queries with branching and type III cycles take longer time compared to others. Task 7 not only takes the longest time but also the highest average attempts. This is particularly due to conceptual mismatch between the users' understanding of domain and the ontology, which forced users to iterate several times.

Table 2.10: Information needs used in the experiment.

#	**Query type**	**Information need**
T1	Single concept	List all fields.
T2	Single concept and property	What is the water depth of the Snorre A platform (facility)?
T3	Short linear	List all fields operated by Statoil Petroleum AS company.
T4	Short with branching	List all exploration wellbores with the field they belong to and the geochronologic era(s) with which they are recorded.
T5	Short linear with type III cycle	List the fields that are currently operated by the company that operates the Alta field.
T6	Long linear	List the companies that are licensees in production licenses that own fields with a re- coverable oil equivalent over more than 300 in the field reserve.
T7	Short with branching	List all production licenses that have a field with a wellbore completed between 1970 and 1980 and recoverable oil equivalent greater than 100 in the company reserve.
T8	Long linear	List the blocks that contain wellbores that are drilled by a company that is a field operator.
T9	Short with branching and type III cycle	List all producing fields operated by Statoil Petroleum AS that has a wellbore containing gas and a wellbore containing oil.

Table 2.11: The results of the experiment.

Participant	**Task**	**Correct**	**Attempt**	**Time**
1	1	1	1	87.9
1	2	1	1	91.6
1	3	1	1	121
1	4	1	1	286.6
1	5	1	1	143
1	6	1	1	281.1
1	7	0	3	507
1	8	1	1	162
1	9	1	1	525
2	1	1	1	45.1
2	2	1	1	89.2
2	3	1	1	109.2
2	4	1	2	437
2	5	1	2	102.6
2	6	1	1	490
2	7	1	2	521.6
2	8	1	3	454
2	9	1	1	311.5
3	1	1	1	26.7
3	2	*	*	*
3	3	0	3	197
3	4	1	1	105.4
3	5	0	1	94.9
3	6	0	2	266
3	7	1	2	456.5
3	8	1	1	224.7
3	9	1	1	339.4

Table 2.12: The results of the exit survey.

Question	**P1**	**P2**	**P3**
"I think that I would like to use this system frequently."	Neutral (3)	Agree (4)	Agree (4)
"I found the system unnecessarily complex."	Agree (4)	Agree (4)	Neutral (3)
"I thought the system was easy to use.	Disagree (2)	Disagree (2)	Neutral (3)
"I think that I would need the support of a technical person to be able to use this system."	Agree (4)	Neutral (3)	Agree (4)
"I found the various functions in this system were well integrated."	Neutral (3)	Neutral (3)	Agree (4)
"I thought there was too much inconsistency in this system."	Neutral (3)	Agree (4)	Disagree (2)
"I would imagine that most people would learn to use this system very quickly."	Disagree (2)	Disagree (2)	Agree (4)
"I found the system very cumbersome to use."	Neutral (3)	Agree (4)	Neutral (3)
"I felt very confident using the system."	Disagree (2)	Neutral (3)	Disagree (2)
"I needed to learn a lot of things before I could get going with this system."	Agree (4)	Neutral (3)	Neutral (3)

Table 2.13: The feedback given by the participants.

"What did you like about the tool?"	**Person**
"Ability to explore new relationships"	P1
"Flexibility"	P1
"Graphic front-end"	P2
"Idea of searching different content"	P2
"Easy graphical interface"	P3
"Responsive"	P3
"understandable functionality between windows"	P3
"What didn't you like about the tool?"	**Person**
"Similar concept names"	P1
"Mappings are not so easy to understand"	P1
"The meaning of branches"	P1
"Delete node"	P2
"Lack of drag & drop"	P2
"I probably missed some understanding of how to express some of the relations and constraints"	P3
"I needed some training."	P3

The feedback provided by the participants through the exit survey is presented in Table 2.12 and Table 2.13. The usability scores given by participants are considerably low despite high completion rates. According to our observations this is particularly due to: the quality of ontology (i.e., ontology was bootstrapped with little manual fine tuning), the size of the ontology, incomplete mappings (i.e., for some cases users found alternative means to formulate a given task for which there was no mapping support), insensible information needs (i.e., some of the query tasks did not make sense for the users), and finally lack of training (which has been intentional to observe learnability of the tool).

2.2.3 Results of the Siemens End-User Workshop

The experiment design follows the first and second experiments – i.e., think-aloud. The experiment is built on a Siemens diagnostic ontology. In total, the ontology includes 5 concepts, 5 relationships (excluding inverse properties), and 9 attributes.

A total of 4 participants took part in the experiment; the profiles of participants are summarized in Table 2.14. A five-minute introduction of the topic and tool was delivered to the participants along with an example before they were asked to fill in a profile survey. The survey asks users about their *age*, *occupation* and *level of education*, and asks them to rate their *technical skills*, such as on programming and query languages, and their *familiarity with similar tools* on a *Likert scale* (i.e., 1 for "not familiar at all," 5 for "very familiar"). Participants were then asked to formulate a set of information needs into queries by using OptiqueVQS and the Optique platform, given at most three attempts for each OptiqueVQS task. Each participant performed the experiment in a single session, while being watched by an observer. Participants were instructed to think aloud, including any difficulties they encounter (e.g., frustration and confusion), while performing the given tasks.

There were 8 tasks, representing the information needs used in the experiment – see Table 2.15. Five of

Table 2.14: Profile information of the participants.

Question	**P1**	**P2**	**P3**	**P4**
"Age"	33	27	60	45
"What is your occupation?"	Software Engineer	Diagnostic Engineer	Mechanical Engineer	Engineer
"What is your level of education?"	Bachelor	Bachelor	Master	Bachelor
"I have technical skills (i.e., computer) such as programming and query languages (e.g., SQL, Java, PHP, SPARQL etc.)"	Strongly Agree (5)	Strongly Agree (5)	Neutral (3)	Strongly Disagree (1)
"I am familiar with tools similar to OptiqueVQS"	Disagree (2)	Strongly Agree (5)	Strongly Disagree (1)	Disagree (2)

these tasks require OptiqueVQS, the remaining three were meant to be solved using the dashboard.

Once users were done with their tasks, they were asked to fill an exit survey asking about their experiences with the tool. The survey was built on the SUS and users were also asked to comment on what they did like and dislike about the tool and to provide any feedback, which they deemed important.

Results

The results of the experiment are presented in Table 2.16. A total of 28 tasks was completed by the participants, with 92 percent correct completion rate and 78 percent first-attempt correct completion rate. The third and fourth users have one incorrect task. The third user exceeded the allocated time for the session and could not attempt the last three tasks, therefore these tasks have been omitted from the results for the third user.

In increasing order, the task completion times and number of attempts on average are: 33 s and 1 attempt for task 1, 36 s 1 attempt for task 6, 162 s and 1.25 attempt for task 5, 168 s and 2 attempts for task 7, and 258 s and 2 attempts for task 2. No time data was recorded for the tasks involving the dashboard as they were rather exploratory.

The feedback provided by the participants through the exit survey is presented in Table 2.17 and Table 2.18. The usability scores given by participants are considerably higher compared to Statoil study. The Siemens diagnostic ontology used in the experiment is small in size and is manually created (i.e., of higher quality). This resulted in a coherent picture, where both task completion rates and usability scores given by the users were in line. User's comment suggest that they did like the design of interface, while they had minor issues such as date format consistency and ambiguity in result presentation. In the informal discussion session, users also highlighted that they would prefer a longer training session.

Table 2.15: Information needs used in the experiment.

#	**Query type**	**Information need**
T1	VQS	Find all assemblies that exist in system.
T2	VQS	Show all messages that tribune NA0101/01 generated from 01.12.2009 to 02.12.2009.
T3	Dashboard	Show all messages that turbine NA0101/01 generated from 01.12.2009 to 02.12.2009.
T4	Dashboard	Find all messages generated by turbine NA0101/01 between 01.12.2009 and 02.12.2009 that contain the text "Trip".
T5	VQS	Show all turbines that sent a message containing the text "Trip" between 01.12.2009 and 02.12.2009.
T6	VQS	Show all event categories known to the system.
T7	VQS	Show all turbines that sent a message category "Shutdown" between 01.12.2009 and 02.12.2009.
T8	Dashboard	Find out whether turbine NA0101/01 is currently running.

Table 2.16: The results of the experiment.

Participant	**Task**	**Correct**	**Attempt**	**Time**
1	1	1	1	68.7
1	2	1	1	89.8
1	3	1	1	*
1	4	1	1	*
1	5	1	2	189.8
1	6	1	1	20.1
1	7	1	3	167
1	8	1	1	*
2	1	1	1	20.7
2	2	1	1	136.7
2	3	1	1	*
2	4	1	2	*
2	5	1	1	279
2	6	1	1	60.3
2	7	1	1	67.6
2	8	1	1	*
3	1	1	1	20.1
3	2	0	3	557
3	3	1	1	*
3	4	1	1	*
3	5	1	1	59.2
3	6	*	*	*
3	7	*	*	*
3	8	*	*	*
4	1	1	1	25
4	2	0	3	250
4	3	1	1	*
4	4	1	1	*
4	5	1	1	120
4	6	1	1	30
4	7	1	2	270
4	8	*	*	*

Table 2.17: The results of the exit survey.

Question	**P1**	**P2**	**P3**	**P4**
"I think that I would like to use this system frequently."	Disagree (2)	Strongly Agree (5)	Strongly Disagree (1)	Agree (4)
"I found the system unnecessarily complex."	Neutral (3)	Disagree (2)	Disagree (2)	Neutral (3)
"I thought the system was easy to use.	Neutral (3)	Agree (4)	Agree (4)	Neutral (3)
"I think that I would need the support of a technical person to be able to use this system."	Agree (4)	Neutral (3)	Disagree (2)	Disagree (2)
"I found the various functions in this system were well integrated."	Neutral (3)	Agree (4)	Agree (4)	Neutral (3)
"I thought there was too much inconsistency in this system."	Agree (4)	Disagree (2)	Strongly Disagree (1)	Agree (4)
"I would imagine that most people would learn to use this system very quickly."	Neutral (3)	Agree (4)	Strongly Agree (5)	Agree (4)
"I found the system very cumbersome to use."	Neutral (3)	Neutral (3)	Disagree (2)	Neutral (3)
"I felt very confident using the system."	Neutral (3)	Agree (4)	Strongly Agree(5)	Agree (4)
"I needed to learn a lot of things before I could get going with this system."	Agree (4)	Disagree (2)	Neutral (3)	Disagree (2)

Table 2.18: The feedback given by the participants.

"What did you like about the tool?"	**Person**
"Very easy to build queries if not SQL expert"	P1
"Good visual confirmation"	P2
"Nice UI design"	P2
"MediaWiki integration is good'	P2
"Nodes make the query understandable"	P2
"Nice interface"	P3
"Visual interface"	P4
"Quick response searching the database'	P4
"What didn't you like about the tool?"	**Person**
"Case sensitive'	P1
"How do you query and/or?"	P1
"Deleting is tiddy'	P2
"Input field need validation'	P2
"No SELECT DISTINCT or GROUP BY"	P2
"Keep going this is the future..."	P2
"Took a couple of minutes to have a feeling about the structure"	P3
"Dashboard should filter the assemblies in different ways"	P4
"Dates shall have same format in all views"	P4
"Explain in the headers what the results are"	P4
"Explain what is expected to be filled in the cells"	P4

Chapter 3

Requirements

This chapter lists the requirements that were identified for the prototype for the third project year. Two end user workshops were held to gather end-user requirements for WP8 and WP9. The results are presented in Section 3.1. Furthermore, input from the scientific work packages and the Description of Work was used to summarize project-internal requirements for Project Year 3 in Section 3.2.

All requirements in this section stem from a need expressed by some stakeholder (end user, project member, or the Description of Work of the Optique project). Thus, requirements in this deliverable are an essential input towards formulating the Year 3 goals and beyond, rather than a list of concrete steps that should be implemented.

The end user requirements for Year 2, as reported in D1.1, contained four "must-have" items essential for end user acceptance. In general, end user acceptance was higher in the Year 2 end user workshops, and the requested features align with the project work plan for Year 3; no "must-have" items deviating from the work plan were identified.

Hence, following both the work plan and end user input, the project intends to focus on the areas of Streaming and Federation in Year 3. In general, work package leaders and scientific leadership will coordinate in prioritizing the requirements and coordinating implementation efforts; requirements touching high-focus areas will be given higher priority.

3.1 End-User Requirements

This section summarizes the requirements that were identified following the usability evaluations held during the end user workshops at Siemens and Statoil. The identified requirements fall into four groups: modifications of the user interface (Section 3.1.1), requirements concerning the query language (Section 3.1.2), requirements concerning interaction with existing end user tools (Section 3.1.3), and requirements for managing the Optique system (Section 3.1.4). The requirements are presented in terms of the *user need*, without prescribing a solution (except when there exists one obvious solution to the end user need). For some requirements, editorial notes typeset in *italics* give additional commentary.

3.1.1 GUI Requirements

▶ Requirement R2.1: Fix Minor GUI Issues — WP3

- Do not display a default icon when there is no icon available. The "Cogwheel" icon used by default looks so distinctive that the user thinks it does something special.
- Make it clear what the little icon with the ">" (right arrow shape) in the concept browser does.

- When going from concept A to concept B, then focussing back on A, the search field content is restored but the list of relations is not filtered according to the search field's content. Either filter the list, or clear the search field.
- The displayed text "(inv)" is unclear and confusing
- The delete button should work on the selected node instead of switching to "delete" mode where the user needs to click on the node again.
- The search input field should stay visible at the top, currently it scrolls out of sight when scrolling through the concept or attribute list.
- Make it possible to enter constraints both as drop-down list and freetext for one attribute - currently it's one or the other. *(Ed. note: part of this requirement is fulfilled by the standard behavior of drop-down lists, which can be searched by typing a prefix of the item, but this behavior is not discoverable and was observed to have issues with focus (user types prefix, selecting desired item → user moves mouse to click on item → list selection changes before user can click on desired item).)*

▶ Requirement R2.2: Make it possible to browse ontology, concepts, relationships WP3

Sometimes the user wants to browse a concept, not add it to the query being built. It should be possible to display a concept's attributes and related concepts without adding it to the query immediately. (The user says "We want to drag & drop instead of adding the concept to the query when clicking.")

▶ Requirement R2.3: Enhance the query catalog WP3, WP2

- It should be possible to share queries with a colleague ("you're the expert, please look at this output")
- The query catalog should offer to save the current query before loading another one.
- The query catalog should show "related queries," by similarity of query or affinity in workflow. *(Ed. note: the user conceptualized workflow affinity as a "Query path": To diagnose this behavior, use query 1, then query 2 or query 3 depending on what you see, etc. The query cataloger could offer "Breadcrumbs" through a diagnostic decision tree.)*

▶ Requirement R2.4: Implement easier query editing WP3

- Make it easy to always / automatically / easily select relevant attributes to display for a concept
- drag-and-drop editing of query structure

  The user wants to "drag and drop" to rearrange the query instead of deleting a concept in one place and recreating it in the other. *(Ed. note: this is difficult in general since two concepts can be connected in more than one way, i.e., via one of many attributes.)*

▶ Requirement R2.5: Better Presentation of Query Under Construction WP3

- After refining the query by restricting a concept by attribute value, show the filter value in the graph, not only the name of the filtered attribute. *(Ed. note: the end user formulated this as "Constraints in the graph, like 'name(c)', could be more concrete, and give that value of the constraint too, e.g. 'name=ATLA'. Maybe: more info about constraints when hovering over a node?)*
- It should be possible to see the name of the relation that connects two concepts in the query graph.

▶ Requirement R2.6: Better Presentation of Query Results WP3, WP2

- The header labels in the results table should be descriptive, not always A, B, C, D.
- When no attribute is selected to display for a given concept, pick one that is meaningful to display in the results table. *(Ed. note: Currently the "label" or else the "title" attribute is always displayed; if this attribute is selected for display in the query, it is displayed a second time. One way to solve this would be to have this attribute always selected for display.*

▶ Requirement R2.7: Better unit handling in query formulation and result presentation WP3, WP4

Units (e.g., feet vs meters) should be handled properly both in query formulation and result presentation.

- It should be possible to set attribute constraints in meters when the table contains attributes in feet.
- Presentation of numbers in the result table should carry the unit, either in the header or in each row.

▶ Requirement R2.8: Better Presentation of the Ontology WP3

- Handle big list of concepts better: group concepts, rank concepts, ... – The current presentation as alphabetical list of concept names does not scale, especially when starting with a new query (where all concepts are available).
- When searching for a concept, e.g. "Field," make sure the exact match comes first
- The treatment of n-ary relations as concepts that "reify" the relationships is too confusing. In particular it is too easy to confuse specialisations like "currentFieldOperator" with reified relations like "Field-Owner" (which may also describe past operators). Working with n-ary relations must become simpler. *(Ed. note: this is partly an issue of ontology engineering, partly of user interface presentation.)*
- When several relations go between the same two concepts, it is currently too easy to confuse them (in part since the goal concept name is more prominent than the relation). It should be easier to pick the

right relation. *(Ed. note: in some cases, subtypes of the goal type might be a better solution than many relations)*

▶ Requirement R2.9: Clear Up Confusion Between Concepts and Relationships WP3

- Make it easier to find out whether a subclass is needed or setting an attribute. E.g. "Exploration Wellbore" is that using a "type" attribute on Wellbore, or is it a separate target class of a role. Or: is there a class "ProducingField"?
- make it easier to find a concept or relationship, even when focusing on the wrong variable. Can be hard to find out where some piece of information should be linked into the query.
- Confusing to the users that the concepts and the attributes look the same, struggle with when to use the left hand side window, and when to use the right hand side window.

One possible solution for these problems is to unify the search fields for concepts and relationships (the current prototype has two search fields, one for each, which means end users have to know whether they search for a concept or attribute). We observed that end users search the ontology by name, and if the GUI displays all matching concepts, attributes and relationships, they would be presented two relevant lists of choices.

▶ Requirement R2.10: Make VQI scale to big data sizes WP3, WP6, WP7

- Make it possible to cancel the execution of a query.
- Make it possible to "test-run" a query. It should be possible to validate a query, e.g., by seeing "typical" result data, before starting a query that will run for hours.
- Give feedback on the approximate size of the result set while constructing the query. Specifically, tell when there will be no results.
- When there are no results, the user should get feedback on whether the reason is that no data is available, or that the query is not satisfiable (and which parts of the query are not satisfiable).

▶ Requirement R2.11: Implement visual query interface for streaming queries WP3, WP5, WP8

It should be possible for end users to construct streaming queries in a visual way, using a suitable subset of STARQL. The design should be guided by the requirements of the WP8 query cataloger.

3.1.2 Query Language Expressivity

▶ Requirement R2.12: Select Multiple Filter Values for an Attribute WP3, WP6

- It should be possible to select different fields from the map, not only one
- Can I show in the results two oil fields at the same time? For example if they are near each other? (Not yet / only in some cases (sometimes we support selecting multiple values for an attribute)).
- Can we choose not only Statoil but also the other Statoil names? (For example in the EPDS database there's "company" and "company names")
- multiple selection of items also on the map widget.

▶ Requirement R2.13: Wildcards in Attribute Queries WP3, WP6

Can we use wildcards (e.g., STATOIL*) for restricting attributes in queries? (For numbers, we can select a range.)

▶ Requirement R2.14: Negations in Queries WP3, WP6

Can we ask for wellbores *not* containing gas? (But how to choose between fields not having a wellbore having gas and fields containing a wellbore that does not contain gas?) *(Ed. note: This came up as a question during the initial presentation, not during discussion of actual end user queries. As such, it is a low-priority requirement.)*

▶ Requirement R2.15: Optional Attributes in Results WP3, WP6

Can we have optional attributes? Optional relationships? I.e., can we display all results, including the ones where the given attribute is empty?

▶ Requirement R2.16: Geographical Constraints and Filters WP3, WP6

- Select a polygon or rectangle on a map and query all wellbores, etc., inside that polygon
  - This is relevant for specifying a project area, a "basin," a country, etc.
  - It might be enough to have a database table of relevant polygons and pick from those, instead of specifying a free-form area.

▶ Requirement R2.17: (Optionally) eliminate duplicates in results WP3, WP6

For a query like "how many machines shut down yesterday?", if one machine shut down 30 times, it should occur once in the results (even though there were 30 distinct shut down events for that machine).

3.1.3 Integration in Enterprise Systems

▶ Requirement R2.18: Export query results in useful formats WP2

- Based on Statoil end user interviews, CSV and KML are a useful basis for further experimentation with the system. Siemens gave similar feedback.
- Siemens expressed interest in exporting time-series data for use in simulation tools.

▶ Requirement R2.19: Integrate import functionality of Optique platform query results in existing end-user tools WP2, WP9

In order to stimulate uptake and awareness of the Optique platform for Statoil's end-users, integration of query results import functionality in existing tools like Petrel and ArcGIS is necessary. We propose to solve this by developing simple plugins or scripts that can be executed from the end-users' tools and import query results using the SPARQL protocol for communicating SPARQL queries and results over HTTP.

▶ Requirement R2.20: Offer hyperlinks in result display that connect to other systems WP2, WP8

Some end users use browser-based tools that operate on entities that can be queried by Optique. When displaying such entities in a query result, the system should offer a hyperlink that opens the same entity in another tool. *(Ed. note: this can likely be handled by either customizing the presentation of the entity detail page, or via generating the link in the ontology / mapping for the concept.)*

3.1.4 Administrative Interface Requirements

The administrative interface of the Optique solution is the management of ontologies and mappings. This has until now only been done by project members, but will be increasingly important for IT administration department stakeholders as the project progresses.

▶ Requirement R2.21: Help with Avoiding Mapping Update Anomalies WP4

Updating a mapping, e.g., changing a URI pattern, or what columns are used to populate a URI pattern, is currently a manual and error-prone process. This should be supported better by the platform.

▶ Requirement R2.22: Improve Ontology Development Workflow WP2

- The workflow for ontology development should be improved, avoiding at least some of the many export–upload steps from Protege to the platform.

- The platform should handle better the situation when multiple ontologies with the same IRI are uploaded. Either deny these uploads, add versioning information, ignore one etc.

3.2 Scientific Requirements

This section presents focus areas for each scientific work package for Year 3. The material presented is based on the progress in Year 2, the Description of Work, and additional input from work package leaders.

▶ Requirement R2.23: Query Formulation WP3

- Temporal and streaming queries:
  - Translation of visually formulated queries into STARQL
  - Usability studies of the temporal and streaming component of OptiqueVQS
- Enhancing query coverage and usability of OptiqueVQS
  - Adaptivity of the interface to end-users' interaction history
  - Development and end-user study for ranking functions
  - Development and end-user study for ontology projection
- Query Driven Ontology Construction:
  - continue development of techniques for ontology construction from query catalogs
  - preliminary enduser study of the developed techniques

▶ Requirement R2.24: Ontology and Mapping Management WP4

- Approximation in OBDA:
  - definition of a technique to approximate both ontology and mapping specification at the same time
  - implementation and experimental evaluation of the above technique
- Mapping analysis:
  - optimization of the current algorithms for mapping analysis
  - definition of algorithms for the verification of further properties
  - release of the final version of the mapping analysis component
- Ontology and mapping evolution:
  - definition of techniques for supporting OBDA evolution
  - implementation and experimental evaluation of a first component for supporting OBDA evolution

- Provenance:
  - extension of the current technique for managing provenance of query results in OBDA
  - implementation and experimental evaluation of the above technique

▶ Requirement R2.25: Time and Streams WP5

- Multi-stream queries should be handled for correlating data by the STARQL query translation engine
- Aggregation operators must be supported as part of the STARQL query translation engine (e.g., trends, correlations, etc.)
- Performance: larger datasets for temporal queries (e.g., Dataset 3 as mentioned above should be handled in minutes using distributed processing techniques)
- Experiments with a large number of continuous queries registered to a server, possibly with a load of temporal (historical) queries in the background. Datasets should be larger, and we should be able to scale them.
- More expressive Tboxes and larger static Aboxes describing objects and events need to be included into the tests.

▶ Requirement R2.26: Query Transformation WP6

- Runtime Query Rewriting: Use Information given by ADP to improve the query rewriting algorithms (possibly making use of materialization). Study the pro and cons of SPARQL vs SQL Federation.
- Runtime Query Rewriting: Study query rewriting in the context of OBDA in the presence of aggregates in queries.
- Transformation System Tuning
  - Continue the activity on the query catalogues provided by WP8 and WP9, and develop techniques and optimizations that improve performance so that ideally all queries can be answered.
  - Continue developing the NPD Benchmark to properly measure the impact of each optimisation/change implemented in the system.

▶ Requirement R2.27: Distributed Query Execution WP7

- Explore adaptive query optimization and execution techniques. This way, ADP can start executing operators, collect statistics from the runtime engine regarding the materialized (intermediate) tables, and re-consider optimization decisions based on the actual numbers and not solely on predictions. This

can also minimize estimation errors that occur from modeling or from the use of inaccurate statistics (histograms).

- Explore the possibility to take as input feedback from Ontop (along with the queries or a-priori) about views that could be useful if they were materialized in ADP.
- Explore intelligent network caching techniques with varying grain of materialization. Replicate cached objects and dynamically direct queries to the appropriate cache container. Explore perfecting options, when appropriate.
- Explore techniques for distributed stream processing. This involves methods for stream replication and scheduling of query graphs (stream producer / consumer relationships) taking into account the rates that queries consume their input streams and the resources they need to be executed (memory, CPU, etc).
- Use UDFs to bundle processing primitives that are hard to express declaratively in e.g. SQL. In the case of stream queries, bundled computations may be pushed near the stream sources, reducing network load.
- Explore lossy compression and model-driven data acquisition techniques to enable real-time processing of massive streams with limited network and CPU resources.
- Fully integrate OPC functionality in the platform.

3.3 Project Internal Requirements

This section lists some requirements that are deemed desirable to ensure forward progress for the project.

3.3.1 Platform Development Requirements

▶ Requirement R2.28: Configuration management WP2

The Optique platform use case installations are becoming increasingly complex artifacts, with intricate dependencies between different software components, e.g., the visual query system, the rewriting component Ontop, and the Information Workbench; and the non-software components like ontologies, mappings, and queries. The project should set up a configuration management system that allows different versions of the components and their dependencies to be represented and monitored.

▶ Requirement R2.29: Release management WP2

- The Optique platform use case installations are becoming increasingly complex artifacts. The project should set up a release management system that allows independent components of the use case platform installations to be tested, deployed and released in regular and controlled fashion using proper infrastructure. The testing and release workflow must include testing on (a test version of) the public showcase, and should, to the extent possible, include testing on the use case installations. Bug reports and suggestive enhancements should be recorded in the internal Bugzilla installation.

3.3.2 Requirements for Impact-generating Activities

▶ Requirement R2.30: Create Training Material WP10

- White paper, training curricula and one-day courses targetting each of ICT, Engineering, Consulting
- Documentation and support material for the Laboratory (public showcase)
- Research compendium: a commented list of essential papers, from the project and elsewhere

▶ Requirement R2.31: Create Partner Program Supporting Material WP10, WP11

- A template business plan, which can be used to address specific enterprise needs by filling in the blanks
- An implementation plan (white paper) describing recommended steps for implementing Optique in an enterprise
- A commercialisation plan
- A risk and mitigation measure list
- "Alpha package" consisting of a test/demo installation of Optique, documentation, presentations, articles sufficient to become familiar with Optique
- Experience reports from pilot installations
- Organize partner program meetings

Bibliography

[1] Samantha Bail, Sandra Alkiviadous, Bijan Parsia, David Workman, Mark van Harmelen, Rafael S. Goncalves, and Cristina Garilao. FishMark: A linked data application benchmark. In *Proc. of the Joint Workshop on Scalable and High-Performance Semantic Web Systems (SSWS+HPCSW 2012)*, volume 943, pages 1–15. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2012.

[2] Nigel Bevan and Miles Macleod. Usability measurement in context. *Behaviour and Information Technology*, 13(1-2):132–145, 1994.

[3] Christian Bizer and Andreas Schultz. The Berlin SPARQL benchmark. *Int. J. on Semantic Web and Information Systems*, 5(2):1–24, 2009.

[4] Tiziana Catarci. What happened when database researchers met usability. *Information Systems*, 25(3):177–212, 2000.

[5] Tiziana Catarci, Maria F. Costabile, Stefano Levialdi, and Carlo Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.

[6] John P. Chin, Virginia A. Diehl, and Kent L. Norman. Development of an instrument measuring user satisfaction of the human-computer interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1988)*, pages 213–218. ACM, 1988.

[7] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. 3(2–3):158–182, 2005.

[8] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2007)*, pages 13–24. ACM, 2007.

[9] Davide Lanti, Martin Rezk, Mindaugas Slusnys, Guohui Xiao, and Diego Calvanese. The NPD benchmark for OBDA systems. 2014.

[10] G Lindgaard and C Dudek. What is this evasive beast we call user satisfaction? *Interacting with Computers*, 15(3):429–452, 2003.

[11] Jakob Nielsen. Why you only need to test with 5 users.

[12] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 1993.

[13] Donald A. Norman and Stephen W. Draper, editors. *User Centered System Design: New Perspectives on Human-computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, 1986.

[14] Martin G. Skjæveland, Espen H. Lian, and Ian Horrocks. Publishing the Norwegian Petroleum Directorate's FactPages as semantic web data. In H. Alani, L. Kagal, A. Fokue, P. Groth, C. Biemann, J.X. Parreira, L. Aroyo, N. Noy, C. Welty, and K. Janowicz, editors, *The Semantic Web – ISWC 2013*, volume 8219 of *LNCS*, 2013.

[15] Harald Storrle. VMQL: A visual language for ad-hoc model querying. *Journal of Visual Languages and Computing*, 22(1):3–29, 2011.