26 November, 2014

CS3202 Project: Iterative development of SPA

Project overview

In CS3201, you have completed analysis and architectural design, as prerequisites to iterative development. In CS3202, you will develop SPA in four iterations, described in four assignments.

Each development iteration delivers a *mini-SPA* that extends the scope of SPA developed in the previous iteration. Mini-SPAs are fully operational and can be tested against SPA requirements specifications provided in the Project Handbook.

At the end of the third iteration, you will have implemented basic requirements for SPA, as described in the Project Handbook. The fourth iteration defines extensions that you may implement for bonus points, provided all the basic SPA functionalities are operational and well tested. Aiming for bonus points based on shaky foundation won't bring you much benefit from the project experience or grading point of view.

Assignment 1-3 suggest the scope of development iterations. Treat our suggested scope as a guideline, an example, and scope your iterations as you wish. But please use the opportunity of weekly consultations to discuss your plans with your team supervisor. This can't hurt, but may save your precious time in case your plans were to create unforeseen difficulties.

Team work

CS3202 is a team project and is evaluated as such. Different students may contribute in different way, according to their skills, abilities and project plans set by the whole team. But all the students are expected to put similar effort to the project.

All the students in team share equal responsibility for creating team spirit, and making the team work as a whole. Should a problem arise, each student must be willing to work towards resolving the problem. Do not hesitate to ask your supervisor for mediation - in the past almost all problems could be helped if addressed early. Each student should deliver work products according to the project plans set by the whole team.

Grading Policy

Please refer to the Grading Policy and Project Evaluation criteria on IVLE.

Readings

For suggestions on how to make detailed design decisions, please refer to Chapter 10 in the Project Handbook. Check Section 10.5 Coding Practices.

Revision of the Prototype

***Due date*: Monday, 26 January, by 12 noon.**

Grace period: 2 hours. We accept your report without penalty until 2 pm.
If you submit report:

Between 2 pm on due date and 2 pm one day later, the penalty is 1/2 of the assignment marks.
After that you receive 0 marks for the assignment.

Deliverables:

Project Report: Place Report into the box outside COM2 #03-21. One Report to be submitted by each team (six students). Integrate descriptions produced by team members and submit one coherent document per team, organized in the required format. You will also present your solution during the consultation during the week when the assignment due date.

Softcopy report and Doxygen docu: Submit to: IVLE->Workbin->Student Submission-> Revision Prototype

SPA will be growing bigger and more complex as you go through CS3202. What counts is that you succeed at the end, so set your priorities accordingly and adopt a long-term perspective.

In the first two weeks of the course, your task is to consolidate the team and revise the prototype you implemented in CS3201 to set up a firm foundation for SPA that you will develop in iterations throughout CS3202. You do not have to implement any new functionality in this initial revision stage. Just clean, refine your design and implementation, improve design decisions.

Do not rush extending your prototype you developed in CS3201. You may need a different set up for a full-blown SPA. Assess your prototype in the view of long-term development in CS3202. **You may even decide to discard it and start anew. Then, it is ok if you extend the re-work of the prototype to Iteration 1.**

Reflect on major decisions regarding how you organize the SPA program. Adopt simple and practical solution of how SPA components (Parser, Design Extractor or Query Evaluator) access data stored in the PKB.

1) Revise your abstract APIs (if appropriate) and concrete APIs.
2) Revise your coding standards (Section 10.5 in Project Handbook).
3) Propose and compare at least two alternative design decisions for each of the following:
    a) Choice of the data representation for abstract syntax structure of SIMPLE program (AST)
    b) Choice of the representation for Modifies
    c) Strategies for Basic Query Evaluator (BQE) to process queries with multiple query clauses
        i) The data structure for storing intermediate results as BQE evaluates query clauses/conditions one by one
        ii) An algorithm to update the data structure storing intermediate results, as BQE evaluates query clauses/conditions.
        iii) Remark: In Basic Query Evaluator you do not consider optimizations by means changing the evaluation order of query clauses/conditions.

Iteration and Assignment 1

***Due date*: Monday, 9 February, by 12 noon.**

Grace period: 2 hours. We accept your report without penalty until 2 pm.
If you submit report:
Between 2 pm on due date and 2 pm one day later, the penalty is 1/2 of the assignment marks.
After that you receive 0 marks for the assignment.

Deliverables:

Project Report: Place Report into the box outside COM2 #03-21. One Report to be submitted by each team (six students). Integrate descriptions produced by team members and submit one coherent document per team, organized in the required format. You will also present your solution during the consultation during the week when the assignment due date.

Softcopy report and Doxygen docu: Submit to: IVLE->Workbin->Student Submission->Iteration 1

1. Suggested scope of implementation in this iteration

Your SPA should allow you to enter a source program and queries. It should parse source program, build PKB, and process queries. For each query, SPA should validate and answer the query, format query result, and display query result.

1.1 Parser of *SIMPLE*

Implement parser for the full *SIMPLE* as described in the Handbook.

1.2 PKB contents

Calls

AST (Follows, Parent)

Modifies, Uses

1.3 Query Processor

In addition to the subset of PQL you implemented in the CS3201 prototype, in Iteration 1 you will implement Calls relationship and **with**-clause.

Query sub-system to be implemented in this iteration

a) Queries can involve the following relationships:
Follows, Follows*, Parent, Parent*, Calls, Calls*

Modifies, Uses – taking into account procedure calls.

b) Implement Query Preprocessor to validate queries and build a query tree.
c) Implement **Basic Query Evaluator** (see hint below)

Grammar rules for subset of PQL in Iteration 1:

Auxuliary defintions: Refer to Handbook, Appendix A.

select-cl : declaration* '**Select**' result-cl suchthat-cl with-cl pattern-cl
declaration : design-entity synonym (',' synonym)* ';'
result-cl : synonym | 'BOOLEAN'

with-cl : '**with**' attrCond
attrCond : attrCompare ('**and**' attrCompare)*

attrCompare : ref ‘=’ ref
// left-hand-side and right-hand-side 'ref' must be of the same type (INTRGER or character string)
ref : attrRef | synonym | ‘”’ IDENT ‘”’ | INTEGER
// in the above, 'synonym' must be a synonym of prog_line
attrRef : synonym ‘.’ attrName

suchthat-cl : ‘**such that**’ relCond
relCond : relRef
relRef : ModifiesS | UsesS | Parent | ParentT | Follows | FollowsT | Calls | Calls*
ModifiesS : ‘Modifies’ ‘(’ stmtRef ‘,’ entRef ‘)’
UsesS : ‘Uses’ ‘(’ stmtRef ‘,’ entRef ‘)’
Calls : “Calls” “(” entRef “,” entRef “)”
CallsT : “Calls*” “(” entRef “,” entRef “)”
Parent : ‘Parent’ ‘(’ stmtRef ‘,’ stmtRef ‘)’
ParentT : ‘Parent*’ ‘(’ stmtRef ‘,’ stmtRef ‘)’
Follows : ‘Follows’ ‘(’ stmtRef ‘,’ stmtRef ‘)’
FollowsT : ‘Follows*’ ‘(’ stmtRef ‘,’ stmtRef ‘)’

pattern-cl : **‘pattern’** syn-assign (entRef, expression-spec)
// syn-assign must be declared as synonym of assignment (design entity ‘assign’).
entRef : synonym | ‘_’ | ‘“’ IDENT ‘”’
expression-spec : ‘_’ ‘“’ factor ‘+’ factor ‘”’ ‘_’ | ‘_’ ‘“’ factor ‘”’ ‘_’ | ‘_’
factor : var_name | const_value
var_name : NAME
const_value : INTEGER

1.4 Hints for Query Evaluator:

In Iteration 1, design the best possible Basic Query Evaluator (BQE) that can correctly evaluate any query without optimizations. Query involving many relationships must be evaluated incrementally, in steps. In the design of BQE, pay attention to computing and representing intermediate query results.

Work out a strategy for Basic Query Evaluator to evaluate queries with multiple conditions.

In subsequent iterations, consider optimization strategies. BQE should be prepared to employ various optimization strategies such as re-arranging the order in which to evaluate relationships in a query and many others. You will have to experiment with various optimization strategies that is why it is important that different optimization strategies can be plugged-into your BQE. BQE should not depend on any specific optimization strategy, but easily work with any optimization strategy you can think of.

1.5 Assignment report

Follow format provided at the end of this document.

Iteration and Assignment 2

***Due date*: Monday, 9 March, by 12 noon.**

Grace period: 2 hours. We accept your report without penalty until 2 pm.
If you submit report:
Between 2 pm on due date and 2 pm one day later, the penalty is 1/2 of the assignment marks.
After that you receive 0 marks for the assignment.

Deliverables:

Project Report: Place Report into the box outside COM2 #03-21. One Report to be submitted by each team (six students). Integrate descriptions produced by team members and submit one coherent document per team, organized in the required format. You will also present your solution during the consultation during the week when the assignment due date.

Softcopy report and Doxygen docu: Submit to: IVLE->Workbin->Student Submission->Iteration 2

2. Suggested scope of implementation in this iteration

2.1 PKB contents

AST (Follows, Parent), Modifies, Uses (for all statements including procedure calls)

Calls, Next

2.2 Query Processor

1) Write down query validation rules and implement a table-driven Query Validation.
2) Start planning optimization strategies for Query Evaluator.

The scope of PQL for this iteration:

Queries should involve multiple **such that, with and pattern** clauses, such as
Select s **such that** … **and** … **and** … **with … and … such that … and … with … and … and …**

Queries should involve the following relationships:

Follows, Follows*, Parent, Parent*

Modifies, Uses

Calls, Calls*

Next, Next*

Grammar rules for subset of PQL in Iteration 2:

Auxuliary defintions: Refer to Handbook, Appendix A.

select-cl : declaration* '**Select**' result-cl (suchthat-cl | with-cl | pattern-cl)*
declaration : design-entity synonym (',' synonym)* ';'
result-cl : synonym | 'BOOLEAN'

with-cl : '**with**' attrCond
attrCond : attrCompare ('**and**' attrCompare)*

attrCompare : ref '=' ref
// left-hand-side and right-hand-side 'ref' must be of the same type (INTRGER or character string)
ref : attrRef | synonym | '"' IDENT '"' | INTEGER
// in the above, 'synonym' must be a synonym of prog_line
attrRef : synonym '.' attrName

suchthat-cl : '**such that**' relCond
relCond : relRef
relRef : ModifiesP | ModifiesS | UsesP | UsesS | Calls | CallsT |
Parent | ParentT | Follows | FollowsT | Next | NextT

ModifiesP : 'Modifies' '(' entRef ',' entRef ')'
ModifiesS : 'Modifies' '(' stmtRef ',' entRef ')'
UsesP : 'Uses' '(' entRef ',' entRef ')'
UsesS : 'Uses' '(' stmtRef ',' entRef ')'
Calls : 'Calls' '(' entRef ',' entRef ')'
CallsT : 'Calls*' '(' entRef ',' entRef ')'
Parent : 'Parent' '(' stmtRef ',' stmtRef ')'
ParentT : 'Parent*' '(' stmtRef ',' stmtRef ')'
Follows : 'Follows' '(' stmtRef ',' stmtRef ')'
FollowsT : 'Follows*' '(' stmtRef ',' stmtRef ')'
Next : 'Next' '(' lineRef ',' lineRef ')'
NextT : 'Next*' '(' lineRef ',' lineRef ')'

pattern-cl : '**pattern**' patternCond
patternCond : pattern ('**and**' pattern)*
pattern : assign | while | if
//the remaining grammar rules related to pattern is the same as in Appendix A.

3. Hints for query optimization:

You can cache intermediate query results as you evaluate query. But you cannot cache across queries, i.e., you cannot use cached results computed in one query to evaluate another query.

The two main factors that have to do with effective query evaluation is the size of the intermediate result and the evaluation time. Optimizations should reduce the size of the intermediate result and/or the evaluation time. Changing the order in which you evaluate relationships in a query, as well as extra information about the source program may help you evaluate queries in more effective way. Assess various optimizations analytically before you implement them.

4. Constraints

Your solution must be scalable, therefore pre-computing Next*, Affects and Affects* for all program statements and storing it is not acceptable. Design efficient algorithms to compute Next*, Affects and Affects* on demand, during query evaluation.

5. Assignment report

Follow format provided at the end of this document.

Iteration and Assignment 3

***Due date*: Monday, 30 March, by 12 noon.**

Grace period: 2 hours. We accept your report without penalty until 2 pm.
If you submit report:

Between 2 pm on due date and 2 pm one day later, the penalty is 1/2 of the assignment marks.
After that you receive 0 marks for the assignment.

Deliverables:

Project Report: Place Report into the box outside COM2 #03-21. One Report to be submitted by each team (six students). Integrate descriptions produced by team members and submit one coherent document per team, organized in the required format. You will also present your solution during the consultation during the week when the assignment due date.

Softcopy report and Doxygen docu: Submit to: IVLE->Workbin->Student Submission->Iteration 3

1. Suggested scope of implementation in this iteration

Full SPA as described in the Project Handbook.

Challenges in Iteration 3:

- Affects and Affects*
- Tuple results, and
- Query evaluation optimization strategies.

2. Constraints

Your solution must be scalable. Therefore Next*, Affects and Affects* must not be pre-computed but computed on-demand as you evaluate a query. Other design abstractions you can pre-compute and store in PKB. Design efficient algorithms to compute Next*, Affects and Affects* on demand, during query evaluation.

You can cache intermediate query results as you evaluate a query. But you cannot cache across queries, i.e., you cannot use cached results computed in one query to evaluate another query. This particularly refers to the situation when you evaluate queries one by one using Autotester.

3. Assignment report

Follow format provided at the end of this document.

Format for report “Revision of the Prototype”

Cover page:

CS3202, date

Team number and name Consultation Day/Hour:

Team members:

Group-PKB:

name	matric number	e-mail

Group-PQL:

name	matric number	e-mail

Free format. Describe how you revised the prototype. Which design decisions have you decided to change and which ones you decided to keep?

Assess of your prototype from CS3201 in the view of iterative development in CS3202. Ask these questions: Is each design abstraction implemented by modules that hide data representation exposing APIs clients? Are concrete C++ APIs in sync with your abstract APIs? If your prototype does not comply to SPA architecture and APIs – I suggest to start development from scratch. This will pay off in longer term.

Format for assignment reports 1-3

Organize your assignment report into the following sections:

Cover page:

CS3202, date

Team number and name Consultation Day/Hour:

Team members:

Group-PKB:

name matric number e-mail

Group-PQL:

name matric number e-mail

1. Achievements and problems in this iteration

1) The scope of SPA implemented in this iteration (SIMPLE and PQL grammars).
2) Any special achievements and/or problems.

2. Describe project plans

Describe your plan for the whole project and for this development iteration.

2.1 Plan for the whole project

Plan who will be doing what during the project. *Tasks* correspond to considerably large work units such as: design parser for subset of SIMPLE, design PKB API, etc. Define tasks of such size that you feel comfortable with planning.

	iteration 1			iteration 2			iteration 3			iteration 4		
team member	task 1	task2	task 3	task 1	task2	task 3	task 1	task2	task 3	task 1	task2	task 3
Mary	*		*									
John		*						*				
Tom												
Suzan						*						
Jack												
Jill												

2.2 Plan for this development iteration

Plan who will be doing what during this development iteration. *Activity* is a smaller work unit than task, for example, you may have an activity such as test interface to AST. Tasks consist of activities. Define activities of such size that you feel comfortable with planning.

	iteration 1				
team member	activity 1	activity 2	activity 3	activity 4	
Mary	*		*		
John		*			
Suzan					
Jack					
Jill					

3. UML diagrams

Draw UML diagrams that you found useful. For each diagram that you draw, explain how you used it (e.g., in project planning, communication, test planning or in other situations), and comment on the value a diagram added to your project.

Hint: Read through relevant parts of Handbook Section 10. Refine UML sequence diagrams given in examples into more detailed sequence diagrams showing communication between SPA functional components and specific data abstractions in PKB.

4. Documentation of important design decisions

Follow guidelines in Handbook Section 10.2 to analyze, justify and document detailed design decisions. Pay attention to clarity of the description (check hints in Section 10.2).

Address detailed design decisions related to data representations for design abstractions (ADTs) in PKB, any other solutions to speed up access to information stored in PKB, algorithms to fetch data from PKB during query evaluation, and any other issues that you consider important, except query evaluation (which is addressed in separate section).

Design patterns provide standardized solutions to design problems. You studied some typical design problems for which published design patterns exist in CS2103. By applying a design pattern, you usually win more flexibility, but an overall program solution may be more complex to work with. Always evaluate carefully the trade-offs involved in terms of expected benefits and the cost of applying a design pattern. Apply a design pattern only if the benefits outweigh the cost.

If you applied design patterns, document them in this section:

a) Explain the design problem and pattern you applied

b) Document expected benefits and costs of applying a design pattern

c) Document the actual benefits and costs of a design pattern that you experienced in the project after applying it.

5. Coding standards and experiences

1) State coding standards adopted by your team.

2) Comment on how you used abstract PKB API to guide design of relevant C++ classes and how you keep abstract and concrete PKB API in sync with other.

   a) Do you use same naming conventions in abstract PKB API and your C++ program?

   b) Do you use typdef to map type names used in abstract PKB API to C++ types?

3) Comment on any experiences - problems and benefits - that you observed working from abstract PKB API towards C++ program.

6. Query processing

6.1 Validating program queries

Describe query validation rules, only in case there is some difference as compared to what you described in your previous assignment. An example of query validation rule is: “checking if all relationships have correct number and types of arguments, as defined in PQL definition in Handbook”. DO NOT provide procedural description (pseudocode) of how Query Pre-processor checks the rules.

If you use table-driven approach to query validation – show the structure of your tables.

6.2 Design and implementation of query evaluator

1. Describe data representation for program queries
2. Describe your strategy for Basic Query Evaluation (BQE)
3. Describe optimizations
4. Discuss detailed design decisions regarding BQE and optimizations

Hint: Follow guidelines in Handbook Section 10.2 to properly analyze, justify and document detailed design decisions. Pay attention to clarity of the description (check hints in Section 10.2). Do not repeat what you already discussed in Section 4, but refer to relevant points.

7. Testing: Group-PKB and Group-PQL

Be sure that you understand the role of Autotester released to you, integrate it with your SPA, and use it to automate regression testing throughout the project.

7.1 Describe your test plan for this iteration

7.2 Provide examples of test cases of different categories

1. unit testing:
   a) provide TWO samples of specific unit test cases for PKB and TWO test cases for PQL
   b) if you used assertions, describe how and show examples
2. integration testing
   a) UML sequence diagrams show communication among SPA components. Use sequence diagrams to plan integration testing and to indicate which component integrations you have tested.
   b) provide TWO sample integration test cases
3. validation testing
   a) provide TWO sample test cases

Document each test case in standard way as follows:

Test Purpose: explain what you intend to test in this test case

Required Test Inputs: explain what program module (or the whole system) you test and what input must be fed to this test case

Expected Test Results: specify the results to be produced when you run this test case

Any Other Requirements: describe any other requirements for running this test case; for example, to run a test case for a program module in isolation from the rest of the system, you may need to implement a simulated environment to call the test case.

8. Discussion

Discuss problems encountered that affected project schedule.

In which areas do you plan to improve in the next iteration?

Discuss any other project experiences and issues.

Appendix A: Abstract PKB API

Include up to date documentation of your abstract APIs for design abstractions.

--- The End ---