ADVANCED DATABASE DESIGN SYSTEM (STAGE 2)

SYSTEM REQUIREMENTS SPECIFICATION

Supervisor: Janusz R. Getta
Members: Changjun, Eric, Michael

TABLE OF CONTENTS

Table of Contents

Introduction 1

Purpose 1

Definitions 1

System overview 3

System usage 3

Project deliverables 4

Overall description 5

Product perspective 5

User Interfaces 5

Software interfaces 5

Memory Constraints 5

Operations 5

Site Adaptation Requirements 9

Product functions 9

User characteristics 9

Constraints 9

Assumptions 10

Dependencies 10

Requirements 11

Priority definitions 11

Category definitions 11

Group A: Schema design quality control 12

Group B: Schema visualization 15

Non-Functional Requirements 18

Envisaged Requirements 19

Requirements drilldown 20

Functional requirements drilldown 20

Improved compiler quality 20

TABLE OF CONTENTS

High-quality schema visualization ____ 23
Schema design quality evaluation ____ 23
Printable schema ____ 24
Improve schema display ____ 24
Parameterized schema ____ 24
Improved technical doc ____ 25
Improved user doc ____ 25
Non-functional requirements drilldown ____ 25
Cross-browser accessibility for schema visualization component ____ 25
User-friendly usage ____ 25
Time limit on compilation and artifact production ____ 25
Envisaged requirements drilldown ____ 26
Schema visualization component: extended functionality ____ 26
Specific requirements ____ 27
External interface requirements ____ 27
Performance requirements ____ 27
Software System attributes ____ 27
Reliability ____ 27
Availability ____ 27
Security ____ 27
Maintainability ____ 27

Introduction

PURPOSE

To improve conceptual schema design quality, through a compiler which gives feedback to the end-user. Furthermore, several artefacts are produced which can be used as a reference by end-users during schema design refactoring.

DEFINITIONS

AJAX	Asynchronous JavaScript and XML. A technique to transfer data to and from the client browser and web server. To be more specific, The AJAX call from the JQuery interface will be utilized (opposed to other methods)
Bison	Lexical parser. Uses input from Flex to execute an associated action
C	A programming language that can be compiled and linked with other libraries to generate programs
C++	A programming language based on C, and provides additional functionality
C library	A collection of functions, procedures and/or classes which can be utilized to extend ones software functionality
CSS	Cascading Style Sheet. Controls visual appearance and layout of HTML elements
DTD	Data Definition Document
Flex	Fast Lexical analyzer. Analyzes lines passed from Bison
HTML	A markup language that is interpreted by a Web browser, to visually represent the content.
HTML element	A component within a HTML document (e.g. head, body, h1)
JavaScript	Client-side scripting language
JQuery	A JavaScript library, which provides additional functionality
JQuery plugin	A plugin written in JavaScript, that utilizes the JQuery interface
PHP	Server-side scripting language
Plugin	A software component that adds functionality to existing software
SQL	Structured Query Language. Used to retrieve data from a database

TCML	Textual Conceptual Modeling Language. A text-based language which describes a conceptual schema
XML	eXtensible Markup Language. Represents elements in nested structures such that parent-child relationships between elements are clear
XML Schema	An XML document that defines and validates the structure of a XML document

SYSTEM OVERVIEW

Note: The development in this project is based on work from the previous stage of this project (stage 1). Development details for this stage are listed under "Project deliverables" on this page, and under "Requirements" on page 9.

The system was developed to solve a common problem when designing conceptual schemas: there's no information which can be used by a conceptual schema designer to determine whether their design is right or wrong. With the end user of the system being a student instead of an experienced professional, designing conceptual schema appear to be even more confusing.

Hence, students "work in the dark" when designing conceptual schemas. Feedback on ones design helps a student get an idea of "the right schema" - however when other students, tutors and lecturers provide different opinions on how the "correct" design of a schema should be, the feedback proves ineffective as the opinions conflict.

UOW students are used to building their code, which involves a compilation step (more specifically, compiling C++ with GNU C++ compiler and Java with javac). The compiler plays a role in introducing opportunities for the user to improve their product, through informative feedback. This feedback can be distinguished into 2 categories - warnings and errors.

This system revolves around quality evaluation of a schema - a compiler that produces artifacts which can be referenced by the students to provide improve their schema design.

As stated previously, the target audience is university students undertaking a database course, where a method of getting feedback on their schema designs would prove to be highly beneficial.

SYSTEM USAGE

1. The user creates a text file adhering to the TCML specification.
2. This file is then compiled through a compiler (created from Flex and Bison technologies). The compiler is run via Windows command line.
3. The compilation result is an XML file representing the conceptual schema defined in the TCML text file.
4. This XML file is then used by several modules; most importantly, the module to generate an interactive conceptual schema within a web page.

Other artifacts produced from compilation include a DTD, XML Schema and a SQL script. However it's crucial to note that this development stage does **not** involve development on the production processes of these artifacts.

PROJECT DELIVERABLES

- A compiler (.EXE format), which generates several artefacts:
    - A XML file congruent with the schema specified in the user-supplied TCML file.
    - A DTD and XML Schema which validates the XML file.
    - A SQL script which contains “create table” statements to create the tables specified in the user-supplied file.
    - A web page, which supports interactivity with a visual representation of the schema.
- Technical documentation (revised from the previous stage.)
- End-user documentation (revised from the previous stage.)

Overall description

The product satisfies the need for quality assurance during conceptual modelling. The end-user specifies a text file adhering to the TCML (Textual Conceptual Modelling Language) specification, which is parsed by a compiler to generate several artefacts.

These artefacts can then be used as references by end-users to evaluate the quality of their conceptual schema design.

PRODUCT PERSPECTIVE

User Interfaces

- Whilst the client has explicitly stated that the final product WILL be run from command line, the Qt framework will be used to develop a GUI for demonstration purposes of the final product.

Software interfaces

- The Flex and Bison interfaces are used to develop the compiler.
- Flex interface (more specifically, the *yyparse( )* and *yywrap( ) methods, in conjunction with the yylval* and *yytext struct).*
- Bison interface to define the TCML grammar.
- C and C++ interfaces (e.g. methods from stdio.h to print warnings and errors, iostream methods to read the TCML contents).
- JQuery interface – particularly the AJAX call.

Memory Constraints

- None, since the end users machines have been recently upgraded – machines which are more than adequate to store the products files, run a small batch file, run a text compiler, open 1 text editor and open 1 web page.

Operations

- The compiler compiles the users TCML file to generate a XML file.
- The compiler implements quality evaluation techniques, by providing feedback to the user's terminal on the detection of design flaws. More specifically, a warning message on the detection of:
    - Cyclic association paradigms (see Figure 1).
    - 2 multi-value attributes existing in a class, where the second multi-value attribute is **semantically connected** to the first attribute. In this case, the second multi-value attribute should be existing as a separate class(s)

to preserve the relationship between both attributes upon transformation into a relational database (see Figure 2).

Figure 1: Example of a design containing cyclic associations

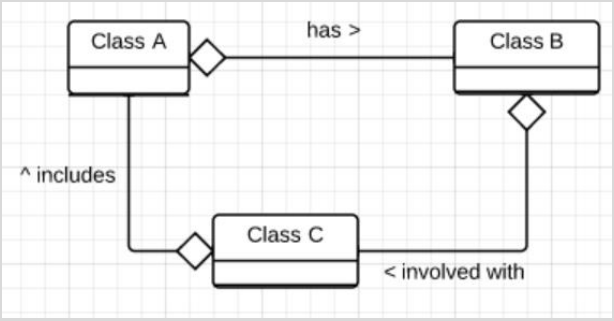
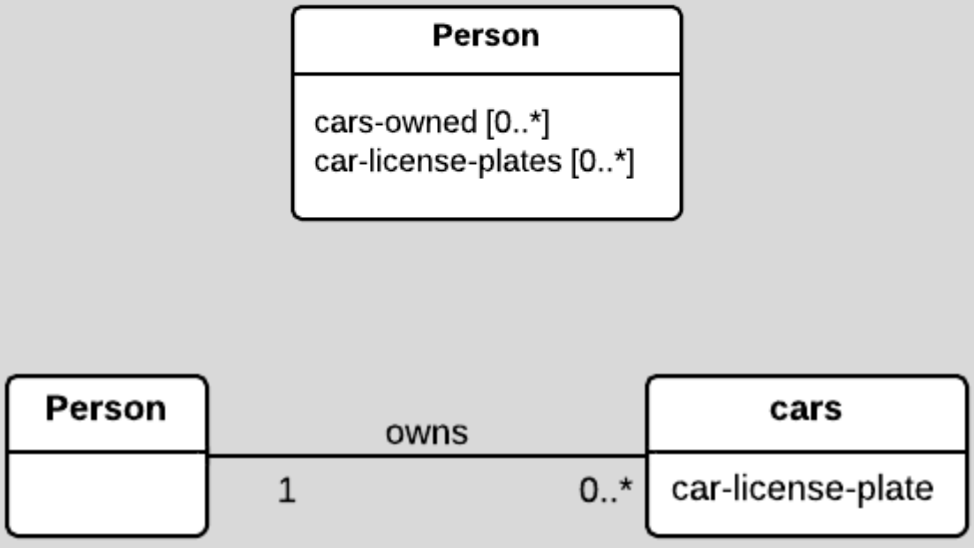
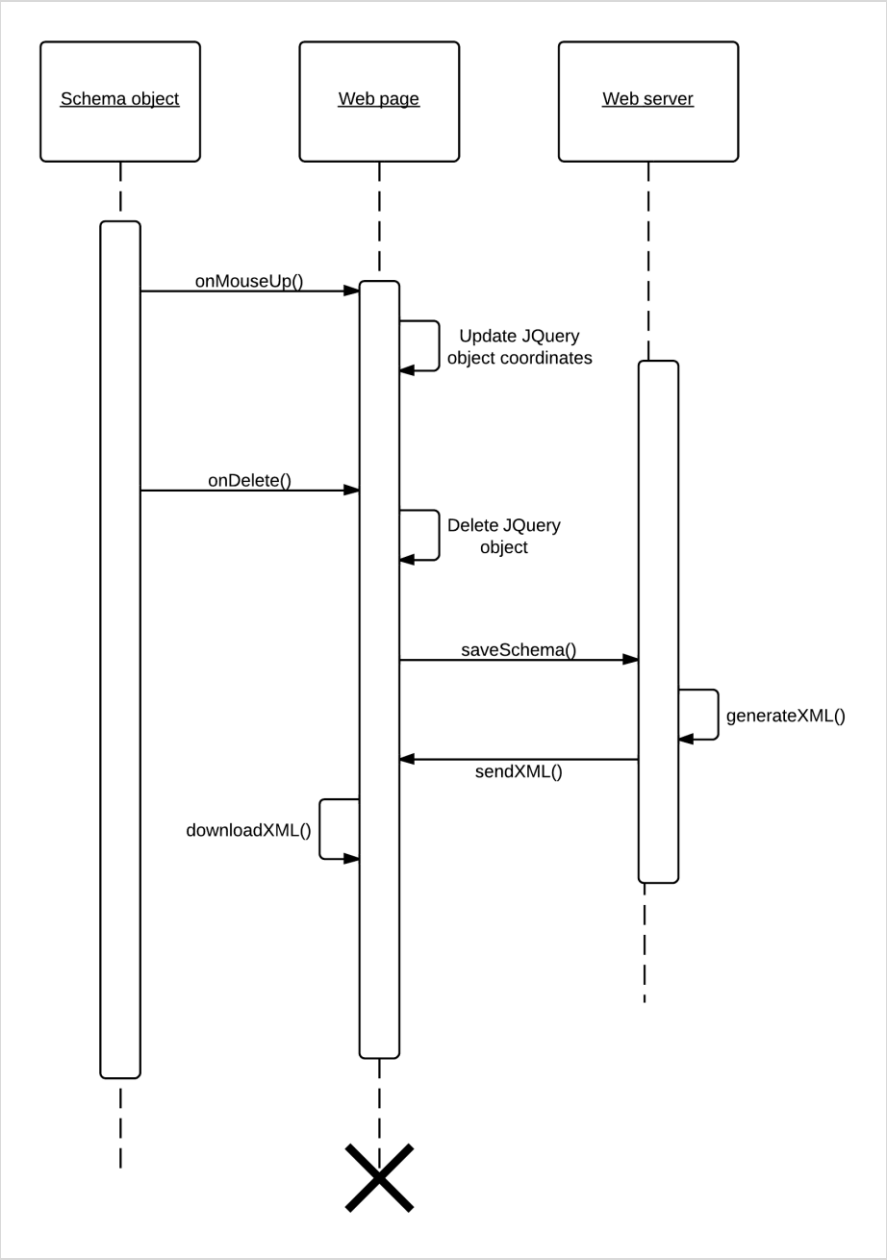


Figure 2: Example of a design involving misplaced multi-valued attributes, followed by its correct design.



- The compiler also provides feedback through the user's terminal, when a syntax error is detected in the TCML file. As a result, the build process is terminated and hence, no artifacts will be produced.
- If no errors are found during compilation, a XML file is written to the file system.
- The product then uses the XML file to produce several additional artifacts:
    - A DTD and XML Schema.
    - A SQL "create table" script, which is congruent to the specification defined in the TCML file.
    - A HTML file containing elements which represent the elements in the XML file. The web page supports interactivity through JavaScript and JQuery.
- Upon schema interactivity (dragging or deleting a class), JavaScript and JQuery code will be executed to update the schema on the web page accordingly.
- When the user chooses to save the altered schema, the web page responds by calling a PHP script hosted on a web server via AJAX. The response from the server is a XML file and a download request. See figure 3 on the following page.

Figure 3: Interactive schema workflow



Site Adaptation Requirements

- None. The package resources are all inside a ZIP file, which provides a high level of portability for different site environments.

PRODUCT FUNCTIONS

- Compiles a text file specified in TCML.
- The compiler notifies the end-user via terminal of any warnings (when the designs quality can be improved) and errors (when the TCML is syntactically incorrect).
- The compiler generates a well-formed XML file upon successful compilation, along with several additional artifacts:
    - A DTD and XML Schema to validate the XML file.
    - A SQL script (*.sql file type), containing the necessary "create table" statements to logically create the tables defined in the TCML file.
    - A web page (*.htm file type), representing the elements in the XML file along with JavaScript and JQuery code to support end-user interactivity.

USER CHARACTERISTICS

- Skilled enough to learn how to operate new programs fast (this comes from constantly learning new programming languages fast in Computer Science subjects).
- May have insufficient experience to operate the Windows command line. Even if they are familiar with operating an Ubuntu terminal at this stage in their curriculum, there are some differences in operating the Windows command line (e.g. *dir* instead of *ls*). They are however, expected to respond to this by “googling” the command line commands and **briefly read** (not thoroughly read) the user manual.
- Experienced enough to handle the “mental load” when introduced with a complex series of steps, instead of becoming frustrated. Respectively, this means installing and running this product.
- Expected to be able to execute an appropriate response to reverse an unexpected operation (e.g. being able to respond using an “undo” button instead of the “redo” button, because they understand the different operations of the 2).

CONSTRAINTS

- Technology: Must be freeware (free to distribute, edit and use in commercial domains). Must also be free of purchase. Any non-proprietary software integrated into this product must require permission from the author(s). The HTTP protocol utilized to communicate between the client web page and web server must adhere to HTTP 1.1 standards (outlined in RFC 2616); JavaScript, JQuery and PHP languages reinforce this standard through their interfaces.

- Time: runtime is expected to take less than 10 seconds (to ensure client satisfaction with the product.)

ASSUMPTIONS

- End-users are fluent in reading English, so they should be able to read the end user documentation. This highlights a potential issue with international students, so the end-user manual may be provided in additional languages.
- Machines used by Database students are assumed to be running Windows. The student's accounts should also have sufficient privileges to uncompress and unpack ZIP files, and run batch files. (This is a critical assumption because it was an issue before the Windows 7 transition, as machines in 1 lab did not have a ZIP program – this has been fixed with the new Windows 7 rollout on UOW machines.)

DEPENDENCIES

- Windows native interface (e.g. running batch files, editing in notepad), Flex, Bison, GNU C++ compiler, Git (including the bash shell.)
- "Dependency hell" is not an issue, as dependencies are used in loosely coupled components.

Requirements

PRIORITY DEFINITIONS

High - Requirements which are necessary to fulfill the core functionality of the system.

Low - Requirements that are the stretch goals for the project. These will only be implemented if time presents itself.

CATEGORY DEFINITIONS

Functional requirement	Requirements which signify the end products functionality.
Non-functional requirement	Requirements which identify performance characteristics of the product.
Envisaged requirement	Optional requirements that may not be implemented by the end of development due to time constraints, but are beneficial enough to the project that they should be considered to be developed past this projects completion date, or if there is sufficient time before the end of the project to implement them.

FUNCTIONAL REQUIREMENTS

GROUP A: SCHEMA DESIGN QUALITY CONTROL

ID: F1	**Requirement:** improved complier quality	**Priority: high**

Description: Improve the existing compiler by amending the following bugs (as specified in the supervisors PDF report):

- Ensure that all classes have an ID, except in cases where the class is a subclass from generalization (e.g. if a class "car" inherits "vehicle", and "vehicle" has an ID attribute, and "car" inherits all attributes from the superclass "vehicle", then its ok for "car" to not have an ID attribute).
- Overlapping composite ID's (i.e. a common attribute(s) between 2 composite keys) aren't processed correctly. An instance of this problem would be a KEY1 = (attribute1, attribute2) and a KEY2 = (attribute2, attribute3)
- Example #1, page 1 of the clients specification (titled "Example 1 Specification of a class") outlines incorrect processing by the compiler, where a derived attribute isn't written to the XML file
- Example #8, page 8 of the clients specification (titled "Example 8 Qualification") outlines the problem of an incorrect primary key being placed in a class "Room"
- Example #10-2, page 11 of the client's specification (titled "Example 10-2 Qualification") identifies a compiler bug, where the processing for an association "Building Includes Room" results in an unqualified association.
- Overlapping composite identifiers are processed incorrectly.(e.g. ID1=(a, b, c) and ID2 = (b, c, d).)
- Incorrect primary key occurs in Qualification.
- An association is not qualified, when it should be. An instance of this problem is an association between "university" and "lecturer", where lecturer teaches at a university. The association should be qualified so that the university can identify the exact lecturer its connected to, using the lecturers primary key/ID.

Detection of advanced design errors and warnings

Design errors

(i) Missing identifiers
A class has neither a generic identifier, nor derived identifier, nor inherited identifier.

A generic identifier is a real world attribute or a set of real world attributes that describes a class and its values uniquely identify each object instance in a class. For example, student number is a generic identifier of a class STUDENT.

A derived identifier is a real world attribute or a set of real world attributes that uniquely identifies each

object instance in a class and such that all or some of the attributes are "borrowed" from another class. For example a class ROOM "borrows" an identifier building number from a class BUILDING to create an own derived identifier (building number, room number).

An inherited identifier is an attribute or a set of attributes that uniquely identifies each object instance and all its attributes are inherited from another class. For example a class POSTGRADUATE-STUDENT inherits identifier student number form a class STUDENT.

(ii) Two or more associations with the same name between a class an many other classes, like for example BUILDING Consists-of OFFICE, BUILDING Consists-of LECTURE-HALL, BUILDING Consists-of TOILET, etc. Typically generalization is missing on the classes OFFICE, LECTURE-HALL, TOILET, etc.

(iii) Missing inheritance due to generalization. For example, generalization OFFICE, LECTURE-HALL, TOILET etc ISA ROOM requires an attribute area to be used in a description of a class ROOM and then it can be inherited by the subclasses.

(iv) Attribute name indicating plural an used without multivalued attribute tag, e.g. [0..*], or [*], etc.

(v) Using artificial ID like identifiers, e.g. roomed, room-id, room_id, etc.

(vi) Identification, that involves optional, multivalued, derived attributes.

(vii) Wrong multiplicity on the other side of qualification, for example cannot be [1..*].

(viii) Attributes used in qualification must be removed from a class on the other side of qualification.

(ix) Link attribute or association class used for one-to-one or one-to-many association.

(x) Empty link attribute

Design warnings

(i) Two or more multivalued attributes in a class.

(ii) Incompleteness, comparison of text based specification with the all names of classes associations, attributes.

(iii) Cycles over associations

(iv) Hidden association, a name of attribute or a set of attributes used as an identifier is also used to describe another class.

Detection of advanced design errors and warnings

(i) Comparison of a design with a sample solution.

(ii) Comparison of a number of designs in order detect similarities.

(iii) Listing of a design in a form of narrative description.

(iv) Drawing shadow diagrams
(v) Identification of design patterns

(4) Standard transformations

(i) Promotion of a multivalued attribute to a class

(ii) Transformation into design with temporal information, e.g.transformation of time dependent attribute, time dependent class, time dependent association.

(iii) Denormalization

(iv) Transformation of non (t-e) generalization into (t-e) generalization.

ID: F2	**Requirement**: schema design quality evaluation	**Priority: high**

Description: A program to analyze the quality of the schema design specified in the generated XML file. Since the compiler already provides functionality for error reporting when TCML syntax is incorrect, there is no need to re-implement the functionality.

Warnings will be shown at command-line (in the same shell used for running the product). There will be 2 events when a warning is produced:

1) When a cyclic association is present in the schema and

2) When a class has 2 multi-value attributes. This is because the second multi-value attribute may have a semantic relationship with the first, and thus when the transformation from a conceptual schema to a relational database is done, the relationship connection is lost.

An instance of an "incorrect" second multi-value attribute is if a "person" class has 2 multi-value attributes, "cars-owned" and "car-license-plates". On transformation to a relational database, the relationship between both attributes is lost. So "car-license-plates" should exist as a separate class with a connection to the class which has "cars-owned" attribute, to preserve the relational information between both attributes upon transformation.

In this case, the second multi-valued attribute may need to exist as a separate class, and have an association with the class it originally resided. The component will be implemented as a separate program, which parses the XML file and detects flaws using algorithms. NB: This method is subject to change

GROUP B: SCHEMA VISUALIZATION

ID: F3	Requirement: high-quality schema visualization	Priority: high
Description: Compiler must produce a HTML file (web page) to display an interactive and parameterized schema and be able to download the newly edited schema as a XML file. The new XML file will include additional properties not previously produced by the compiler, including class locations and schema parameters (parameter specifics in functional requirement F2.3). The new XML file will then be able to be used from the existing compiler to re-generate the visualization. Furthermore, association lines must NOT overlap on the web page.		

ID: F4	Requirement: printable schema	Priority: high
Description: Allows printing of the webpage (and hence, the schema). The trigger will be a HTML button, with the callback function providing schema scaling capabilities (such that if the schema is too big to fit on 1 A4 page, the algorithm will re-scale the entire schema so that it does fit). The printing request is conducted using the browsers native 'print page' functionality (similar to selecting 'File > Print'.)		

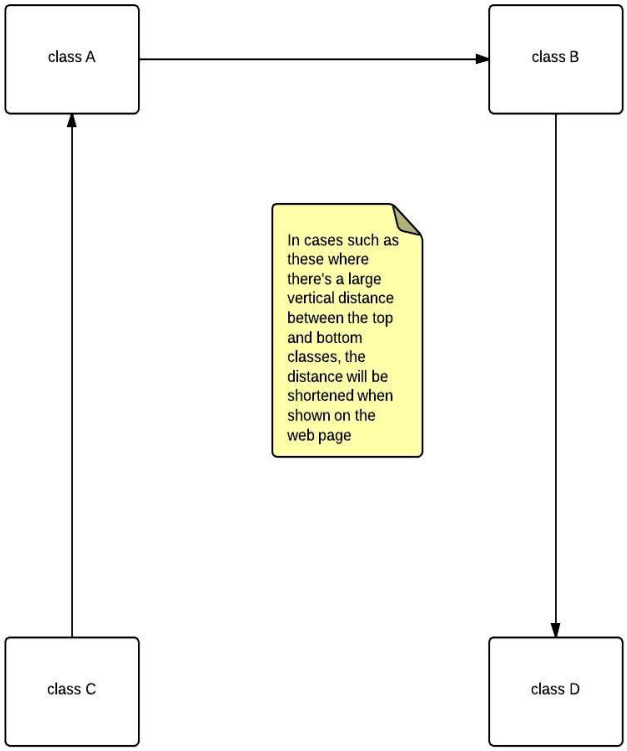
ID: F5	Requirement: improve schema display	Priority: high
Description: The web page (showing the schema) will display schema elements in a space-effective way. To be specific, if class placement on the web page results in unnecessarily large distances between them, then the web page will shorten these distances for viewing and printing convenience (*see figure 2.3 on the following page for an illustration.*)		

ID: F6	Requirement: parameterized schema	Priority: high
Description: Upon loading the web page, the user can choose parameters to configure the schemas visual representation. Configuration of these parameters will be done through a popup box. The parameters include: • The corner radius of class boxes		

- Class box background color
- Font family and size
- Choose the schemas background from a pre-created set

ID: F7	Requirement: improved technical doc	Priority: high
Description: Improved technical documentation, in terms of detail and clarity. The exact specifics of what's to be changed is TBD upon more thorough analysis of the documentation upon a later date. However the vision is currently to mimic the manuals of retail products (e.g. clear logical structure through table of contents and index, figures which reinforce a technical explanation in the document which can be understood by technical users.)		
ID: F8	**Requirement:** improved user doc	**Priority: high**
Description: Improved end-user documentation, in terms of clarity. Specifics of what is to be changed is TBD upon more thorough analysis of the documentation at a later date.		

Figure 2.3: An instance of schema visualization producing an unsatisfactory layout


class A
class B
In cases such as these where there's a large vertical distance between the top and bottom classes, the distance will be shortened when shown on the web page
class C
class D

NON-FUNCTIONAL REQUIREMENTS

ID: N1	**Requirement**: cross-browser accessibility for schema visualization component	**Priority: high**
Description : The webpage (containing the schema representation) will be cross-browser compatible with Chrome, Firefox, Safari, and IE7+.		

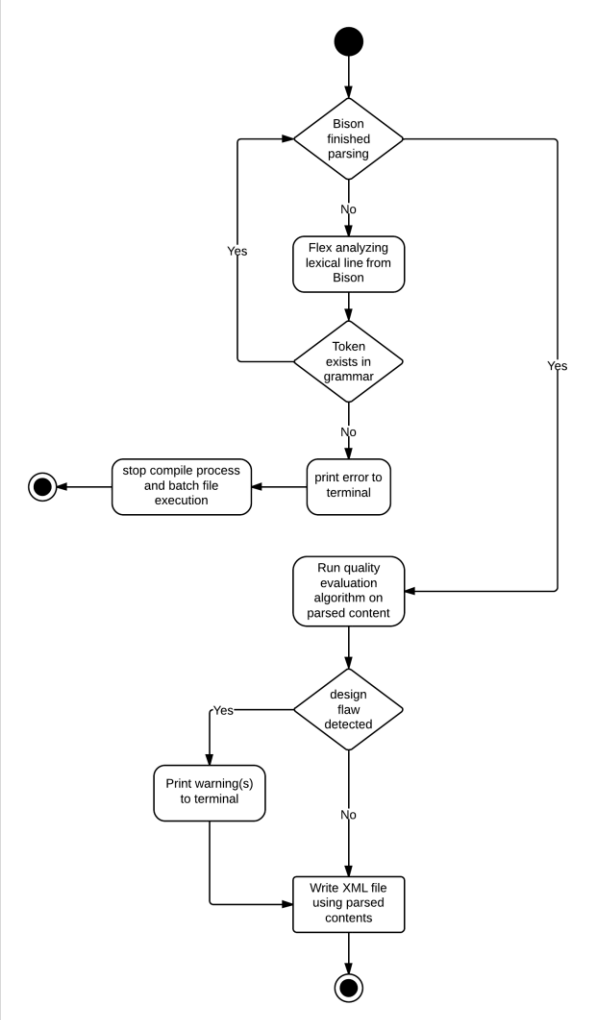
ID: N2	**Requirement**: user-friendly usage	**Priority: high**
Description: With the target audience being very specific (i.e. undergraduate students undertaking a database subject), the user-friendliness of the product will cater for their assumed skill level. More specifically, they will be able to understand the technical and user documentation and be technically capable enough to learn how to use the compiler.		

ID: N3	**Requirement**: time limit on compilation and artifact production	**Priority: low**
Description: For user convenience, the time from compilation to artifact production (i.e. web page, sql create table statements, dtd, xml schema) should be performed in less than 10 seconds.		

ENVISAGED REQUIREMENTS

ID: E1	Requirement: schema visualization component: extended functionality	Priority: low
Description: Extending the functionality of the interactive schema on the web page, by providing: • The ability to add attributes to a class from the web page, and • Delete classes in interactive schema and hence, delete all connected associations.		

Figure 4: Warning and error detection during compilation



Requirements drilldown

This sections purpose is to clarify how the functional, non-functional and envisaged requirements will be implemented. This is done by "drilling down" each requirement and exposing the sub-tasks involved. Each sub-task is then given a priority of scalar value, rather than a category (e.g. "3", instead of "high priority"), indicating the importance of the task to the implementations success.

Scale legend:

1 = Quality of implementation affects the requirements implementation success **very heavily**.

2 = Quality of implementation affects the requirements implementation success **heavily**.

3 = Quality of implementation affects the requirements implementation success **moderately**.

FUNCTIONAL REQUIREMENTS DRILLDOWN

Improved compiler quality

Bug fix #1: No information about derived attribute is stored in XML file ("Example #1" from client's specification.)	3
Bug fix #2: Overlapping composite identifiers are processed incorrectly.(For example ID1=(a, b, c) and ID2 = (b, c, d).)	3
Bug fix #3: Incorrect primary key occurs in Qualification (Example #10-2 from client's specification.)	3
Bug fix #4: Compiler error. Association is not qualified in particular qualification	3
Bug fix #5: Missing and misplaced ID's in XML file from incorrect compiler processing of TCML file (e.g. Example #8 from clients specification.) Cases where a primary key occurs in a qualification when it should not must be amended.	3
Bug fix #6: Generate compiler warnings when a possible case of unqualified associations arise, where they may need to be qualified	3

Detection of advanced design errors and warnings **Design errors** (i) Missing identifiers A class has neither a generic identifier, nor derived identifier, nor inherited identifier. A generic identifier is a real world attribute or a set of real world attributes that describes a class and its values uniquely identify each object instance in a class. For example, student number is a generic identifier of a class STUDENT. A derived identifier is a real world attribute or a set of real world attributes that uniquely identifies each object instance in a class and such that all or some of the attributes are "borrowed" from another class. For example a class ROOM "borrows" an identifier building number from a class BUILDING to create an own derived identifier (building number, room number). An inherited identifier is an attribute or a set of attributes that uniquely identifies each object instance and all its attributes are inherited from another class. For example a class POSTGRADUATE-STUDENT inherits identifier student number form a class STUDENT. (ii) Two or more associations with the same name between a class an many other classes, like for example BUILDING Consists-of OFFICE, BUILDING Consists-of LECTURE-HALL, BUILDING Consists-of TOILET, etc. Typically generalization is missing on the classes OFFICE, LECTURE-HALL, TOILET, etc. (iii) Missing inheritance due to generalization. For example, generalization OFFICE, LECTURE-HALL, TOILET etc ISA ROOM requires an attribute area to be used in a description of a class ROOM and then it can be inherited by the subclasses. (iv) Attribute name indicating plural an used without multivalued attribute tag, e.g. [0..*], or [*], etc.	3

(v) Using artificial ID like identifiers, e.g. roomed, room-id, room_id, etc. (vi) Identification, that involves optional, multivalued, derived attributes. (vii) Wrong multiplicity on the other side of qualification, for example cannot be [1..*]. (viii) Attributes used in qualification must be removed from a class on the other side of qualification. (ix) Link attribute or association class used for one-to-one or one-to-many association. (x) Empty link attribute **Design warnings** (i) Two or more multivalued attributes in a class. (ii) Incompleteness, comparison of text based specification with the all names of classes associations, attributes. (iii) Cycles over associations (iv) Hidden association, a name of attribute or a set of attributes used as an identifier is also used to describe another class. ## Detection of advanced design errors and warnings (i) Comparison of a design with a sample solution. (ii) Comparison of a number of designs in order detect similarities. (iii) Listing of a design in a form of narrative description.	

(iv) Drawing shadow diagrams (v) Identification of design patterns (4) Standard transformations (i) Promotion of a multivalued attribute to a class (ii) Transformation into design with temporal information, e.g.transformation of time dependent attribute, time dependent class, time dependent association. (iii) Denormalization (iv) Transformation of non (t-e) generalization into (t-e) generalization.	

High-quality schema visualization

create html5 web page	3
devise and implement algorithm to parse xml, and consequentially produce DOM elements representing the schema	3
integrate jsPlumb component for association lines	3
devise pseudo code to layout classes in an order where the association lines don't overlap + implement code	3
provide functionality to download schema as a XML file	2
implement functionality to read class locations and schema parameters from XML	3
cross-browser testing	2

Schema design quality evaluation

Devise psuedocode to detect cyclic associations + desk check with team	3

Implement code to detect cyclic associations, and output a warning message to terminal	2
Devise pseudo code to detect possible misplaced multi-value attributes(more specifically, where there are 2 relationally-connected multi-value attributes exist in a single class)	3
Implement code to detect misplaced multi-value attributes, and output a warning message to terminal	2
Implement "suggestion" functionality (i.e. The program will suggest how to correct the design flaw(s))	2

Printable schema

devise psuedocode for finding the ratio needed so that the schema can be printable on 1 A4 page + desk check with team	3
implement code to scale schema	3
create button graphics in Photoshop, and export as png	2
link button DOM element with JavaScript code to implement printing functionality	3

Improve schema display

devise psuedocode to layout classes in a grid-like space-efficient manner + desk check with team	3
implement code for space-efficient class layout on schema	3

Parameterized schema

elicit customizable parameters from supervisor	3
implement JavaScript code for popup box, and for textboxes to change parameters	3
implement JavaScript code such that the parameters determine: • The corner radius of class boxes • Class box background color • Font family and size • Choose the schemas background from a pre-created set	3

Improved technical doc

All team members to analyse aspects of documentation that can be improved, and jot them down on paper	2
Discussion of improvable aspects in team meeting	1
Cross-check with client/supervisor of change log	1
Edit documentation content	3
Change documentation design to a DOCX design template	3
Create UML diagrams for technical explanations	2
Insert UML diagrams into documentation	1
Export DOCX to PDF	1
Present PDF to client/supervisor, to ensure satisfaction and feedback	3

Improved user doc

<same as previous requirement drilldown>

NON-FUNCTIONAL REQUIREMENTS DRILLDOWN

Cross-browser accessibility for schema visualization component

Chrome test (latest version)	3
Firefox test (latest version)	3
Safari test (latest version)	3
IE7+ test	3

User-friendly usage

revise technical and user documentation together with team	3
amend any areas which could be improved	3

Time limit on compilation and artifact production

time the batch script on all machines in database laboratory	3

ENVISAGED REQUIREMENTS DRILLDOWN

Schema visualization component: extended functionality

Implement functionality to add attributes to a class from the web page	3
Implement functionality to delete classes in interactive schema and hence, delete all connected associations.	3

Specific requirements

EXTERNAL INTERFACE REQUIREMENTS

- Communication from Bison to Flex must be compatible with Flex's external interface (e.g. yylex() for parsing the TCML and sending each line to Flex).
- Communication from Flex to Bison must be compatible with Bison's external interface (e.g. grammar and tokens defined within Bison's header file, tokens returned from Flex must match one of the tokens defined in Bison's token section).
- Other interfaces (e.g. C++ iostream, JQuery interface, AJAX etc.) have already been implemented, and their role is simply a component in this system.

PERFORMANCE REQUIREMENTS

- Runtime should be less than 10 seconds (to ensure client satisfaction with the product.)

SOFTWARE SYSTEM ATTRIBUTES

Reliability

- Since different users may have different opinions on what is the "best" design for a problem, the product can't be relied on completely to determine the quality of a schemas design.

Availability

- Only available to Windows NT users. (TBA at a later date, however NT is the bare minimum required for decent web page rendering and DOS support.)

Security

- Users must have sufficient privileges to download a file within a web browser.

Maintainability

- Product documentation will be improved. This will allow future developers on the system to have a clearer understanding of system semantics.
- A consistent variable naming convention:
    - **1 word** variables must be in lower-case *(e.g. list, argc.)*
    - **2 words or longer** variables initially start with a lowercase letter, with subsequent words beginning with an uppercase. Whitespace, hyphens, dots etc. are NOT to be used in variable names. *(e.g. linkedList, lowerToUpper, compareDouble.)*
- A consistent commenting style. Other styles are strictly prohibited in the interest of code maintenance. Accepted styles are outlined on the following page.

Function comments

```
/**
 * Function description beginning with an uppercase
 * @param parameterName parametersRole
 * @return descriptionOfWhatIsReturned
 */
void myFunc() { … }
```

Single-line comments

```
// This comment has a space after the double splash, and begins with a capital
```