University of Southern Queensland

Faculty of Science

Department of Mathematics and Computing

Enhancing the IMS Q&TI Specification by adding support for dynamically generated parameterised quizzes

A Dissertation submitted by

Damien Clark, B Comp

In partial fulfilment of the requirements of the degree of Master of Computing

2004

Abstract

In recent years, the use of electronic online quiz systems has become quite popular due to pressures of increasing class sizes, and the need for more efficient methods of assessing distance students.

One of the main benefits of using multiple choice questions (MCQ) with online quiz systems is automated marking. However, to help address the problems of cheating and to allow students the opportunity to resit quiz tests, it is necessary to develop a large number of MCQ items from which a random selection is made. This ensures that each quiz taken is sufficiently different. To develop good quality MCQs takes considerable time; therefore, time saved in marking is lost in development of quality MCQs.

To offset this overhead, a method can be used whereby certain elements of the question are parameterised. Randomly selected option(s) can be inserted to alter the details of the question, while still testing the same content area. Creating one parameterised question can result in a large or infinite number of different quiz questions at low cost.

Independent or collaborating researchers have developed many custom built online quiz systems. Each uses a different data format to represent quiz questions, and results in poor or no interoperability. Questions developed for one system cannot be re-used in another. To stem the flow of incompatible custom built systems, the IMS Global Learning Consortium have developed an IMS specification for describing questions and tests called The Question and Test Interoperability Specification. Institutions and vendors are now implementing their systems to support this

specification such that interoperability can be achieved of tests between different quiz testing products. The IMS Question & Test Interoperability Specification is quite complete and is extensible to allow for functionality that does not fit within the specification guidelines. However, this is at the cost of breaking interoperability.

An enhanced version of the IMS Q&TI Specification has been developed to support the interoperable storage of parameterised quiz questions. Using this enhanced specification, ten multiple choice and fill in the blank questions from a range of subject areas have been defined which each implement parameterisation. To demonstrate these ten parameterised questions, a prototype implementation of a quiz system has been developed, which supports the enhanced specification.

Certification of Dissertation

I certify that the ideas, experimental work, results, analyses, software and conclusions reported in this dissertation are entirely my own effort, except where otherwise acknowledged. I also certify that the work is original and has not been previously submitted for any other award, except where otherwise acknowledged.

Signature of Candidate	Date

ENDORSEMENT:

Signature of Supervisor/s	Date

Acknowledgements

I would like to take the opportunity to express my sincere appreciation for the efforts of those who have helped guide me through this new journey of academic research.

I have often heard that to have an excellent supervisor can mean the difference between bumping into things in the dark and following a guiding light. In my case, I was blessed with not one, but two excellent supervisors. I would firstly like to thank Dr. Penny Baillie-de Byl who was my principal supervisor for the most part of my research. Penny's comprehensive feedback and timely advice was most appreciated. I'm a much more confident author as a result of Penny's efforts. Congratulations to Penny on the start of her new family with baby "Tabytha". Secondly, I would like to express my thanks to David Jones who agreed to be my associate supervisor in Rockhampton. David who has been a mentor for several years has always been a great source of honest feedback and advice, and this research project was no exception. David's words of encouragement during the low periods of this project were most appreciated.

I'd also like to acknowledge the efforts of others who have provided invaluable advice at some time or another during the past twelve months, including Associate Professor Russel Stonier, Dr. Robert Snoke, Dr. Daniel Stonier and Dr. Richard Watson who assumed the position of principal supervisor in Penny's late absence.

Finally, I would like to give special thanks to my wife Petrea for her patience, support and encouragement not only for the past year, but for the last 10 years. *I couldn't do it without you babe!* In addition, special thanks to my family for their encouragement and understanding.

Table of Contents

List of Figures

List of Tables

1 Introduction

This chapter explains the background associated with this research. It introduces Computer Aided Assessment, the IMS Global Learning Consortium's Question and Test Interoperability Specification, the concept of parameterised questions, and the objective of this research. It will also describe the layout of this document, including a brief explanation of each chapter.

1.1 Research goals

In recent years, Computer Aided Assessment (CAA) has become a common tool used by educational institutions as part of the delivery of their course materials. This is as a result of the benefits provided by this technology due to the possibility of automated assessment marking, such as reductions in staff workloads (Dalziel 2000; Jacobsen and Kremer 2000; Jefferies, Constable et al. 2000; Peat, Franklin et al. 2001; Pain and Heron 2003); more timely feedback to students (Merat and Chung 1997; Jefferies, Constable et al. 2000; Sheard and Carbone 2000; Woit and Mason 2000; Dalziel 2001); and reductions in educational material development and delivery costs (Muldner and Currie 1999; Jefferies, Constable et al. 2000). One form of CAA is Online Quizzes, which have become popular with educational institutions for providing an electronic version of the traditional quiz assessment. These Online Quiz Systems permits electronic collection of student responses. The electronic responses can then be automatically marked and results and feedback provided to the student almost immediately. However, development of quality questions for online quizzes can be time consuming (Crofts 1999; Dalziel 2000; Jacobsen and Kremer 2000; Lister 2000; Woodbury, Ratcliffe et al. 2001; Angseesing 2002; Davies 2002; Pathak and Brusilovsky 2002; Pain and Heron 2003). Furthermore, if the quiz is to be used for

summative assessment, it is also necessary to create a sufficiently large pool of questions, such that each student will not be asked all the same questions. This helps to prevent cheating through collusion but further exacerbates the problem of the cost of question construction. Researchers are working on solutions to help with this issue.

One area of active research is the parameterisation of quiz questions. This is defined as quiz questions which have components of the question definition (stem) represented by variables, which can be instantiated with random values (defined with boundaries) when the question is presented (Pathak and Brusilovsky 2002). This results in multiple or infinite instances of the same question with different details (Brusilovsky and Miller 1999; Pathak and Brusilovsky 2002). Refer to the subsection titled “Parameterised Quiz Questions” under Section 2.1.3 for a more detailed description. The benefits afforded by the use of this technique helps to reduce cheating where students share solutions to quizzes presented online, because each student will have different solutions for the same questions (Crofts 1999; Lundquist 2001; Ashton and Beevers 2002; Pathak and Brusilovsky 2002). Another major benefit from their use is to provide more effective formative assessment, allowing students to repeatedly attempt the same parameterised questions as each attempt will have different details and a different solution (Crofts 1999; Pathak and Brusilovsky 2002). Therefore, the student is continually challenged until mastery of the question topic is achieved (Lundquist 2001).

Another problem that is encountered with Online Quiz Systems is interoperability of quiz content between disparate quiz systems (Ferrandez 1998; Crofts 1999; Dalziel

2000; Anido-Rifon, Fernandez-Iglesias et al. 2001; Burguillo, Benlloch et al. 2001). For example, quiz questions created in one quiz system may not necessary be in a compatible format to be re-used in another quiz system. To overcome this issue, a group known as the IMS Global Learning Consortium, who produce open specifications for interoperable learning technology; have developed a specification for describing quiz questions. It has been designed such that all quiz system vendors who support the specification can import, export quiz question content, and allow sharing of quiz material. The IMS Question and Test Interoperability Specification (QTI) is a complete and extensible specification, which supports many interoperable features as used by Online Quiz Systems. However, one such feature that is not included is support for parameterised questions.

This dissertation investigates the development of an enhanced version of the IMS QTI Specification to support the interoperable recording of parameterised quiz questions.

1.2 Dissertation Structure

The next chapter (Chapter 2) discusses the current body of knowledge with regard to CAA and in particular, Online Quiz Systems, their perceived benefits, issues relating to their implementation, and industry specifications supporting interoperability of quiz content. It will also introduce the topic of parameterised quiz questions including their benefits and shortcomings.

Chapter 3 will take a closer look at the suite of IMS Specifications, and in particular, the QTI Specification. It will discuss the different sub-specifications that make up the QTI and how they relate to one another. It also presents a more detailed breakdown

of the QTI Item as part of the ASI sub-specification (Assessments, Sections, and Items), which is responsible for describing all information relevant to a particular test item (question). XML (Extensible Mark-up Language) is the technology used by the QTI Specification to facilitate the encoding of the stored information (refer to Appendix B for further explanation of XML). Diagrams are provided throughout to illustrate the Document Type Definition (DTD), which describes the XML structures used to encode the data. The chapter concludes with some examples of questions encoded in XML as per the IMS QTI Specification.

A conceptualisation of parameterised questions is provided in Chapter 4. Topics discussed in this chapter include a more detailed definition of parameterised questions, content that is well suited to parameterisation, media types, and examples of potential parameterised questions. Discussion continues with methods to perform candidate response processing, such as calculation of distracters and determining the correct answer to the question. The chapter finishes with an examination of question types including multiple choice and fill in the blank questions.

In Chapter 5, the improved version of the IMS QTI Specification supporting parameterised questions is introduced. This includes an initial overview of the new QTI Parameterisation Addendum (QTIPA) Specification XML structure, followed by a more detailed discussion of each component of the specification. Different approaches to implementation of the new specification are compared and contrasted including justifications for the selected options.

A prototype system, The QTIPA Quiz System, is introduced in Chapter 6. An overview of the implementation details are covered such as the level of compliance with the specification, along with an overview of the software design. The chapter discusses the use of the Perl programming language for the system, and describes the classes implemented to support the specification. The chapter concludes with an example multiple choice and an example fill in the blank question, illustrating the rendered output from the initial question presentation, the candidate response feedback, and the QTIPA XML, which drives the questions.

Chapter 7 discusses the outcomes from this research. It also reflects on the work completed and introduces ideas for future research to be undertaken on this topic. Appendix A lists the new QTIPA Specification DTD, which is the major outcome from this research. Appendix B provides a brief introduction to XML to assist those not familiar with the technology. Appendix C describes the contents of the attached Supplementary CDROM, which includes the complete source code for the QTIPA Quiz prototype system, and the QTIPA XML which implements each of the ten example parameterised questions used throughout this document. A glossary is provided in Appendix D.

2 Computer Aided Assessment

This chapter will discuss the current body of knowledge of Computer Aided Assessment (CAA), including fully-automated and semi-automated marking systems, online quiz systems and industry specifications supporting the interoperability of assessment data between such systems.

2.1 Introduction to CAA

Computer Aided Assessment (CAA) is a method of assessing students' performance, assisted through the means of a computer system. These systems may be implemented via mechanical means such as Optical Mark Reader (OMR) Systems, or via the use of Internet based technologies. Internet based technologies in CAA can be broadly categorised into the following system types: fully automated marking, semi-automated/computer assisted marking, and online quiz systems. In this section, each of these Internet based CAA types will be discussed.

2.1.1 Fully-Automated Marking Systems

A fully-automated marking system is a technique that can mark electronically submitted assignments via Online Assignment Submission Management (OASM) (Benford, Burke et al. 1994; Jones and Jamieson 1997; Roantree and Keyes 1998; Mason and Woit 1999; Darbyshire 2000; Thomas 2000; Huizinga 2001; Gayo, Gil et al. 2003; Jones and Behrens 2003; Trivedi, Kar et al. 2003), automatically generating a final grade for the assignment with no (or very little) interaction with a human.

The Ceilidh Courseware System (CCS) was an early work in the area of fully automated assessment marking developed by Benford, Burke et al. (1994). CCS

automatically marks student programming assessment and multiple choice questionnaires through an X Windows menu based interface on a UNIX platform (Benford, Burke et al. 1994), as it was implemented before the explosion of web technology on the Internet. The core of the system's programming grading algorithm is the use of UNIX regular expression pattern matching techniques. These regular expressions are used to analyse the output derived from a student's submitted programming assessment, to verify that it matches what is expected based on the input provided to the student's program. This analysis determines a grade.

English and Siviter (2000) from the University of Brighton developed a system to assess student web pages as part of the summative results for a first year introductory course. The student submitted web pages are assessed by validating their HTML (Hypertext Mark-up Language), checking for errors or broken links etc, and calculating statistics on use of tags ensuring they meet required metrics as per the assessment rubric (English and Siviter 2000).

MEAGER, introduced in 2003, is a program developed to automatically assess and grade Microsoft Excel spreadsheet assessments (Hill 2003). It utilises Microsoft Access Visual Basic Macros (VBA) to automate the process of checking the students' submissions against a model solution in a Microsoft Excel spreadsheet developed by the assessment creator. The fields are extracted from the student and model solution spreadsheets, storing them in Access database tables before performing a comparison using a free third-party application (Hill 2003). MEAGER automatically tallies the results for each student and records the feedback at the bottom of the student's Excel workbook, and also in an Access database backend (Hill 2003).

Other fully automated systems include the UK Examination board's assessment for the accreditation of word processing skills (Long 2000), an automated essay grading system (Palmer, Williams et al. 2002), the OASIS-fuzzy system using fuzzy logic techniques to automatically assess students' work (Bigdeli, Boys et al. 2002), and another approach using Microsoft Excel to automate marking of spreadsheet concepts assessments (Summons, Coldwell et al. 1997).

Fully automated marking facilitates the marking of electronically submitted assignments via OASM, such that little or no interaction is required by a human marker. However, some assessment types require human interpretation during the marking process. These can often be dealt with electronically using semi-automated marking systems.

2.1.2 Semi-Automated Marking/Computer Assisted Marking

Semi-automated marking describes systems that have some components of the automated marking process, but still requires human interpretation and analysis to assign a final grade.

Systems have been developed to support the routine tasks associated with marking programming assignments, like compilation and testing of student submitted programs (Joy and Luck 1998; Jackson 2000). The BOSS System, for example allows students to submit their programming code, and will then automatically test the code, comparing the output with that of a known correct solution (Joy and Luck 1998). Although assignment of a final grade is the sole responsibility of the marker,

this determination can be achieved faster, more accurately and more consistently by relying on the results of the automated tests reported by BOSS (Joy and Luck 1998). Jackson (2000) has developed a similar system which guides the marker through the different criterion for the assessment, such as: comparison of test output, review of documentation, programming style and comments. This reduces much of the mechanical work involved in marking programming assessment. Classmate, a prototype system developed by Baillie-de Byl (2004) provides an online computer assisted marking environment, in which a team of markers can provide timely, valuable and consistent feedback to students. The system assists the marker by automating many of the laborious tasks associated with marking student assessment such as retrieving and opening students' submissions, electronic storage of feedback and marks, automatic application of penalties where necessary for late student submissions, and the return of feedback to the student (Baillie-de Byl 2004).

Another content area addressed by semi-automated systems includes work by Price and Petre (1997). They developed a marking tool for conversion and collation of student submitted word processor documents, into a standard marking document template (Price and Petre 1997). The marking template provides a number of built-in tools to assist the marker in providing feedback, storing and calculating results for each student's assessment resulting in shorter marking time (Price and Petre 1997).

Markin by Creative Technology (http://www.cict.co.uk/software/markin/index.htm) is a commercially developed software package. It supports semi-automated marking of student assessment providing similar features to that of the system developed by Price

and Petre (1997), but is not limited to Microsoft Word submitted assignments (Creative-Technology 2003)

By automating some of the steps involved in marking students' assessment, in particular the repetitive mechanical tasks, efficiency gains can still be realised through a reduction in the time taken by the marker to assess the students' submissions (Joy and Luck 1998). However, CAA is not limited to assignment based assessment, but also extends to the more traditional area of online examinations, such as electronic or online quizzes.

2.1.3 Online Test/Quiz Systems

Online quiz systems facilitate the testing process via computer, thus allowing electronic collection of student responses. Innovations in the area of online quiz systems include:

- use of fuzzy logic, associative memory or physical number theory to analyse student answers, assigning partial credit when appropriate (Hubler and Assad 1995),
- guided discovery exercises (Woolf, Day et al. 1999),
- marking mathematical expressions through the CUE system's Judged Mathematical Expression (JME) (Ashton and Beevers 2002),
- hidden multiple choice questions[1] (Ashton and Beevers 2002),
- programming assessment via completion of a template program (Lister 2000),

[1] Hidden Multiple Choice Questions are designed to present each possible selectable answer one at a time such that the student must immediately recognise the correct answer once presented, thus reducing the scope for guessing (Ashton and Beevers 2002).

- general online programming examination systems with automated marking (Woit and Mason 1998; Arnow and Barshay 1999; Daly and Waldron 2002a),
- the insertion and updating of a quiz test bank online via a web browser (Ratna, Raymonth et al. 2003),
- scaling of students' scores in quiz tests, based on students' confidence of knowing the answer (Davies 2002; Richmon, O'Shea et al. 2002),
- a client/server protocol for an online quiz system (Farah 2002),
- a modular quiz system, allowing ability to plug in new modules to support new question types (Matsuno, Tsutsumi et al. 1999),
- use of rich multimedia via Flash MX to deliver quizzes (Li 2003),
- use of client-side javascript (Hay and Nance 2002) or java applets (Crisp 2001) in online quiz systems,
- self-assessment (Peat, Franklin et al. 2001),
- adaptive testing and intelligent tutor systems (Woolf, Day et al. 1999; Sjoer and Dopper 2002),
- question meta-data (Crofts 1999),
- parameterised programming quiz questions (Pathak and Brusilovsky 2002).

Of all the innovations taking place in the recording and storage of quiz questions, the most active areas of research are adaptive quizzes, quiz meta-data, and parameterised quiz items (Brusilovsky and Miller 1999). These will now be examined in detail.

Adaptive Quizzes

Adaptive quizzes are designed in such a way as to alter the sequence and selection of subsequent test items, based on the student's performance on previous test items (Sjoer and Dopper 2002), adapting to the student's level of knowledge (Brusilovsky and Miller 1999). The main advantage of adaptive testing methods is that students receive question items commensurate with their abilities in the course, thus resulting in a test that continues to challenge each individual student (Davies 2001; Sjoer and Dopper 2002).

A system that supports two adaptive testing methods is that of Etude (Sjoer and Dopper 2002). It allows for: 1) students to be presented with a question and based on whether they are correct or not, they will next receive a more difficult or less difficult question; and 2) the instructor to devise specific paths of questions for students to take based on the correctness of previous responses (Sjoer and Dopper 2002). A similar system that supports adaptive testing is the AEGIS system (Mine, Shoudai et al. 2000). It allows for adaptive testing by adjusting the difficulty level of questions based on the performance of the student (Mine, Shoudai et al. 2000). However, in what way can an adaptive quiz system know which questions to select to adapt to a student's needs or performance? This can be achieved using quiz meta-data.

Quiz Meta-data

To support advanced features in quiz systems such as adaptive quizzes, Brusilovsky and Miller (1999) suggest that appropriate attributes pertaining to the questions need to be recorded. This would include the:

- content area the question assesses,
- difficulty level of the question,
- pre-requisite knowledge or competency,
- type of question (multiple choice or fill in the blank for example),
- content keywords.

This type of information when stored with the question is known as quiz meta-data. The main advantage of including meta-data is that questions can be grouped and catalogued according to their attributes. This allows intelligent selection and sequencing of material such that systems are able to generate customised and personalised quizzes for students (Brusilovsky and Miller 1999).

Crofts (1999) also found that with only a limited set of attributes recorded for question items, there is often more meta-data associated with the question than the question material itself, and suggests there needs to be major incentives to invest the time in developing and maintaining this meta-data. Daly (2002b) presents the use of MS-Office for recording and storing quiz question banks, including the ability to record quiz item meta-data.

Parameterised Quiz Questions

Another area of active research is the parameterisation of quiz questions. This is defined as quiz questions that have components of the question definition (stem) represented by variables, which can be instantiated with random values (defined with boundaries) when the question is presented (Pathak and Brusilovsky 2002) thus resulting in multiple or infinite instances of the same question with different details (Brusilovsky and Miller 1999; Pathak and Brusilovsky 2002). For example, a typical question might be that shown in Figure 2.1.

What is 12 multiplied by 6?

Figure 2.1 Example of a typical parameterised question.

To parameterise the question in Figure 2.1, simply replace 12 and 6 with variables and define a range of possible values for each. If one variable had a range of 1 to 20, and the other 1 to 10 each in increments of one, then there would be 20 x 10 = 200 possible instances of the question, each testing the same skill: multiplication.

There have been many systems developed that support parameterised quiz questions. Pathak and Brusilovsky (2002) have developed a web based system called QuizPACK, which delivers dynamic parameterised quizzes for programming related courses. Traditionally, parameterised questions are used in mathematics based courses in which objective answers can be automatically calculated based on the parameter values (Pathak and Brusilovsky 2002). The QuizPACK system has been designed to support parameterised code evaluation questions for the C and C++ programming languages (Pathak and Brusilovsky 2002). A code evaluation question

provides a segment of programming code to the student, and the student must be able to interpret the algorithm and determine a resulting value or output from the code segment. By introducing parameters into the segment of code, a different outcome will result for each attempt of the question.

Kashy, Thoennessen et al. (1997) reveal through the CAPA System the ability to parameterise both the value and the placement of labels within a diagram (image) as part of a question. The students must answer a series of questions based on the information provided within this diagram. This demonstrates that parameterisation is not limited to textual media. Due to the number of labels and the variation of values that can be assigned in the example provided, a large number of possible instances can be generated (Kashy, Thoennessen et al. 1997).

Through the use of the commercial product WebCT, Lundquist (2001) of Lulea University of Technology has implemented parameterised questions to assess students in an engineering statistics course. By using a combination of randomly selected questions, some of which are also parameterised, circumvention of assessment becomes more difficult, as students cannot simply share solutions with one another (Lundquist 2001). This is because the solutions for each student's quiz are different.

Although not the focus of their work, other research into online quiz systems that provide support for parameterisation include:

- CUE as used by Heriot-Watt University in Edinburgh (Ashton and Beevers 2002),
- CyberProf developed at the University of Illinois (Hubler and Assad 1995),

- OWL (Online Web-based Learning) implemented to support electronic homework at the University of Massachusetts (Woolf, Day et al. 1999),
- a web based system developed by Merat and Chung (1997) at Case Western Reserve University.

2.1.4 Question Types

A question type is commonly defined as a method of categorising the different rendering or interface formats for presenting a question to a student, and the method by which the student provides their solution. The question types can be loosely categorised into two separate groups: selection and fill in the blank.

Essentially selection based question types require that the student select one or more answers from a list, which can be presented in many ways. This provides a very objective method of soliciting a response from the student, as they are constrained by the options provided to them. Thus, selection question types provide a simple method of automating the marking of the assessment.

In contrast, fill in the blank based question types are more free-form, allowing the student to be more expressive with their response, which in turn allows for a more subjective method of soliciting a response. This subjective method can be problematic when working towards an automated marking solution. Still, this is also dependent on the content of the question and whether it is objective (eg. mathematics, computer programming) or subjective (eg. art, journalism).

The IMS Global Learning Consortium, committed to developing open specifications for interoperable learning technology, provide support for the following question types within their Question & Test Interoperability Specification (IMS 2002a).

- Selection rendering
    - Multiple Choice variations
        - True/False
        - Multiple Choice
        - Multiple Response
    - Image hot spot
    - Slider
    - Drag object
    - Select text
- Fill in the blank rendering
    - Fill in the blank
    - Short response
    - Essay response

An example of a typical selection based question is illustrated in Figure 2.2, representing a multiple choice question (MCQ). The student must make a selection from the list of options, labelled (a) to (e) to answer the question stem (question definition). One of the options is the correct answer, known as the key. While the remaining four options are known as distracters, as they are designed to distract the student from selecting the key. Thus ensuring the student is challenged in answering the question.

For the TCP/IP protocol, in which OSI Relational Model Layer does the tcp protocol belong?

a) Layer 1
b) Layer 2
c) Layer 3
d) Layer 4
e) Layer 5

Figure 2.2 Example multiple choice question.

The other selection methods work on a similar theme, being that the student must make a selection from a finite data set in answering the question. Fill in the blank (FIB) type questions require that the student type in a response to the question, which can be expressed as the student wishes. Figure 2.3 provides an example of a typical fill in the blank question.

What is:

5 multiplied by 6? <student input>.

Figure 2.3 Example fill in the blank question.

The student is required to place the answer of the question inside the box labelled "<student input>" to complete the question. Similarly to the selection based question types, other variants of the FIB question type are based on a similar theme, where the student must type in a response to the question.

In summary, a question type is merely a method of categorising the different rendering or interface formats for presenting a question to a student, and the method in which the student provides their solution. These question types can be loosely categorised into two separate groups: selection and fill in the blank, where the student

must either select the correct answer from a list, or express their answer by typing in the content.

2.2 CAA Implementation

This section will discuss the benefits of implementing CAA, the issues that can arise, and what industry specifications are available to support interoperable quiz question banks.

2.2.1 Benefits of CAA

The push towards Computer Aided Assessment (CAA) systems has been driven by many factors. These include:

- the need to reduce educational staff workloads (Dalziel 2000; Jacobsen and Kremer 2000; Jefferies, Constable et al. 2000; Peat, Franklin et al. 2001; Pain and Heron 2003),
- a push for more timely feedback to students (Merat and Chung 1997; Jefferies, Constable et al. 2000; Sheard and Carbone 2000; Woit and Mason 2000; Dalziel 2001),
- reduction in educational material development and delivery costs (Muldner and Currie 1999; Jefferies, Constable et al. 2000),
- the proliferation of online education (White 2000).

The introduction of CAA can significantly reduce the workload associated with assessing large student classes through automated marking. Automated marking is the process by which CAA technology is used to automate the process of evaluating and assessing students' performance (see Section 2.1).

Another advantage in using automated marking is the reduction in turnaround time of assessment feedback. Race (1995) as referenced by Davies (2001), suggests that the most important thing lecturers do for their students is to assess their work. Feedback on student assessment is an important element of the learning process (Dalziel 2001). Due to the ability to automate the marking process afforded by Online Quiz Systems, one of the benefits provided by CAA is being able to give more timely feedback to students on their assessable work (Merat and Chung 1997; Jefferies, Constable et al. 2000; Sheard and Carbone 2000; Woit and Mason 2000; Dalziel 2001). For example, the CADAL Quiz system developed by Sheard and Carbone (2000) allows students to monitor their knowledge throughout the term by self-assessment and as a revision tool for their examination without relying on the human tutor or marker.

Given that CAA provides for automation in the marking process, there is the potential for reduction in operating costs through the use of this technology particularly with larger classes, as human tutors and markers need not be employed to undertake the laborious marking process (Muldner and Currie 1999; Jefferies, Constable et al. 2000). Furthermore, by re-using a randomly selected subset of questions from a larger question databank, the CAA system can further increase cost savings as it is not necessary for all new assessment material to be developed anew for every offering of the course (Jefferies, Constable et al. 2000).

The Internet has brought about the advent of online courses through distance learning (Ratna, Raymonth et al. 2003). In recent years this form of education has experienced a great deal of growth in enrolments (White 2000). Having students enrolled in courses when they can be based anywhere in the world presents some serious

challenges, including how to assess the students. The use of CAA through web technologies has supported this mode of study by allowing students to sit online assessment pieces at their chosen location, without having to travel to an examination centre at a scheduled time. Distance education provides for students who do not live within proximity of a campus, and so they must rely on electronic communications and print materials to complete their study. The use of CAA can provide benefits to distance education students such as:

- support for independent and self-managed learning (Sheard and Carbone 2000),
- the ability to undertake online quizzes from their own home (Thomas, Price et al. 2001),
- provide the ability to gauge the student's progress throughout the term (Hay and Nance 2002), due to limited interaction between the teacher and students.

In brief, the majority of benefits incurred from the use of CAA are derived from the automated marking process (Dalziel and Gazzard 1999). The evolution of CAA systems has been driven by the need to assess students in a more efficient manner, and to improve the benefits to both students and lecturers, such as cost efficiency, time efficiency, and delivery flexibility.

2.2.2 Issues Relating to CAA Implementation

There are many issues surrounding the implementation of CAA such as cheating and plagiarism, development time for quality questions and lack of interoperability between CAA software products. Each of these issues will be investigated further.

Cheating and Plagiarism

Cheating and plagiarism in summative assessment has been a constant problem, and summative assessment through the use of CAA is no exception (Winslow 2002). The students simply find new and more creative ways to cheat in an online environment (White 2000; Winslow 2002). However, Shaughnessy (1988) as referenced by Winslow (2002) suggests very few students premeditate cheating, yet will take advantage of a situation if given the opportunity. Students themselves also have reported their concerns regarding the less honest students cheating with CAA systems (Jacobsen and Kremer 2000). Honest students find it very discouraging to see other students cheating without detection (Pain and Heron 2003). Many of the issues surrounding the integrity of CAA through online quiz systems are discussed further on. This includes methods employed by students to cheat, countermeasures to try to reduce cheating, and the role parameterised quiz question play in this quest.

Newstead et al. as referenced by Winslow (2002) reports from a survey of 943 college students that cheating is quite widespread. Winslow further references research by Hollinger (1996) who states it was found that over two-thirds of undergraduate students had engaged in some form of "academic dishonesty" during a semester.

Students can cheat in CAA environments by exploiting the following shortcomings.

- Non-invigilated quizzes resulting in an inability to ensure that the work is the students' (Lundquist 2001; Thomas, Price et al. 2001).
- Invigilated electronic quizzes suffer from traditional cheating methods, such as students bringing in hidden crib sheets, or accessing other non-authorised material electronically (Pain and Heron 2003).

- Students citing fake technical difficulties in an effort to circumvent the assessment process, like pretending they accidentally dropped an item on the keyboard submitting their quiz prematurely (White 2000).
- Use of insecure client-side processing algorithms for answers, such as javascript technologies can allow students to deviously determine the answer (Hay and Nance 2002).
- Not specifically limited to CAA environment, however online plagiarism can be achieved by copying text directly from web pages without reference and has received much attention in research literature (Winslow 2002).
- Inability to prevent students from discussing answers with other students who are sitting their assessment at a later time also allows students to circumvent the assessment process (White 2000; Pathak and Brusilovsky 2002; Pain and Heron 2003).

Measures can be implemented to help reduce cheating in CAA environments such as:

- variation on the time a quiz is available to be taken (Lundquist 2001),
- delivering the online quiz in an invigilated environment (White 2000; Pain and Heron 2003),
- performing a style analysis of students' other assessment pieces in comparison with a similar analysis of their quiz answers (Thomas, Price et al. 2001),
- use of Internet conferencing software like Microsoft Netmeeting to verify the student's identity (Thomas, Price et al. 2001),
- advocating student compliance with value systems[1] (Winslow 2002),

[1] Schneider (1999) and Paldy (1996) as referenced by Winslow (2002) suggest instituting honour codes with students is an effective countermeasure to cheating.

- randomly selected questions, making each quiz unique so students cannot pass around all answers to the quiz (Brusilovsky and Miller 1999; Crofts 1999; White 2000; Lundquist 2001; Pain and Heron 2003),
- randomisation of values within questions to make each question subtly unique (parameterised questions) such that quiz questions shared by multiple students will not have the same answers (Brusilovsky and Miller 1999; Crofts 1999; Lundquist 2001; Pathak and Brusilovsky 2002).

Cheating and plagiarism are not just limited to traditional assessment methods (Winslow 2002), and therefore lecturers administering CAA assessments must still be ever vigilant in ensuring integrity. One of the shortcomings of traditional online quiz systems, is the inability to prevent students from discussing answers with other students who are sitting their assessment at a later time (White 2000; Pathak and Brusilovsky 2002; Pain and Heron 2003). Therefore, by implementing parameterised quiz questions, at least one simple opportunity for cheating is eliminated, as the solutions for each student will be different, yet the students are still assessed in the same content area. There are other benefits to parameterised questions, such as a reduction in development time for course assessments.

Development time for quality question

This section discusses the time required to create quality Questions for CAA and some methods that can help reduce this time, including the use of parameterised questions.

Multiple Choice Questions (MCQ) (see Section 2.1.4) are the most common question type used in CAA (IMS 2000) due to their objective nature and the ease of automated marking. However development of good quality MCQs is a very time intensive activity (Crofts 1999; Dalziel 2000; Jacobsen and Kremer 2000; Lister 2000; Woodbury, Ratcliffe et al. 2001; Angseesing 2002; Davies 2002; Pathak and Brusilovsky 2002; Pain and Heron 2003). Davies suggests that it is not so much the writing of MCQs, but rather writing good quality distracters that makes it a time consuming process (Davies 2002). Rust (1973) estimated the full cost for development of an objective test item at £10 in 1973, therefore after factoring in inflation, this cost would be around £40 in 2000 (Woodbury, Ratcliffe et al. 2001). While some believe that although there is great time saved in the automation of the marking, this time saved is lost through development/maintenance of the question banks (Dalziel 2000; Lister 2000). Yet others suggest that after the initial overhead of the development of MCQ banks, it will show a return of the invested time after a few years when the material is re-used or when used for large class sizes (Dalziel 2000; Angseesing 2002). The use of FIB type questions too has its own overheads, which can also make their development time consuming. These overheads are incurred through the sometimes difficult task of evaluating a student's response. This is because FIB type questions have a more free-form response method allowing the

student to be more expressive with their response, and are therefore often not as objective in nature as MCQs (see Section 2.1.4).

Methods of reducing the time involved in developing/maintaining test item banks include:

- use of parameterised questions (see Section 2.3) (Pathak and Brusilovsky 2002),
- adoption of textbook question banks (Dalziel 2000),
- sharing of question banks between departments, or even institutions (Crofts 1999),
- use of an Open Software Foundation's (OSF) General Public Licence (GPL) copyright model for sharing of questions in the public domain (Dalziel 2000).

The time required to develop quality MCQ can be quite considerable, however by enabling re-use of existing question banks through parameterised questions and sharing this material with others, can help reduce the time required to develop and maintain this material.

Lack of interoperability

A great deal of research has been done in the area of CAA and in particular, development of online quiz systems. However, much of this work has been undertaken by independent or small groups of researchers who are developing systems to service the particular needs of their courses or institutions, without regard for interoperability. Such research includes:

- the Cyberprof Quiz System (Hubler and Assad 1995)

- a web based quiz system for a microprocessor course (Merat and Chung 1997)
- the WebQP Quiz System (Anzai, Hirahara et al. 1999)
- the WebToTeach Learning Management System (Arnow and Barshay 1999)
- the Interactive Courseware Quiz Creator (Matsuno, Tsutsumi et al. 1999)
- ACME - the Automated Courseware Management Environment (Muldner and Currie 1999)
- OWL - Online Web-based Learning (Woolf, Day et al. 1999)
- a custom built web-mounted testing facility (MacKay 2000)
- the Tripartite Interactive Assessment Delivery System (TRIADS) through the proprietary storage format developed by Authorware™ (Mackenzie 2000)
- the Computer Aided Dynamic Assessment and Learning Quiz (CADAL Quiz) (Sheard and Carbone 2000)
- the Mapview System (Monitoring, Access, and Provision) (Woodbury, Ratcliffe et al. 2001)
- CMAPROG (Angseesing 2002)
- the BOSS Quiz System (Joy and Luck 1998)
- CALM - Computer Aided Learning in Mathematics (Ashton and Beevers 2002)
- the iQUIZ System (Farah 2002)
- a parameterised quiz system for delivering programming quizzes (Pathak and Brusilovsky 2002)

Of the quiz systems discussed above, very few mention any technical aspects of how their systems work, which is limited at that. Furthermore, none of this research describes what format is used to store question content, either use a proprietary format

such as Authorware™, or do not demonstrate any ability to share question content with other systems. Crofts (1999) indicates that the lifetime of quiz questions will be longer than that of the systems in which they were first implemented, leaving the difficulty of porting the question banks to a new system, or sharing the material with other organisations using incompatible software.

2.2.3 Industry Standards

Standards organisations around the world have been working hard in recent years to develop new learning technology standards (Burguillo, Benlloch et al. 2001; Dodds 2003; IEEE 2003; ISO 2003; IMS 2003a; IMS 2003b). These standards are necessary as educational materials are defined, structured, and presented in various formats, and application components embedded in learning systems cannot be re-used or interoperated with other systems in a straightforward manner (Ferrandez 1998; Crofts 1999; Dalziel 2000; Anido-Rifon, Fernandez-Iglesias et al. 2001; Burguillo, Benlloch et al. 2001).

Researchers have also contributed standards in the area of Learning Technologies such as:

- Tutorial Mark-up Language (TML) (Williams, Browning et al. 1997),
- SCAAN Project (Sclater and Howie 2000),
- SimulNet, a component model for standardised web-based education (Anido-Rifon, Fernandez-Iglesias et al. 2001),
- Webtest Mark-up Language (WTML) (Crofts 1999).

Although there are many organisations working on education technology standards, they are aware of each other's contributions and their work is beginning to converge. For example, the ISO SC36 group ensures that technical work is synchronised and resources are put to effective use, by liaising with other affiliated organisations and being mindful of related work (ISO 2003), and the Advanced Distributed Learning (ADL) Convergence/Catalyst Approach (Dodds 2003). The SCORM (Sharable Contect Object Reference Model) project by ADL has been designed to join together disparate works by other standards organisations in the area of learning objects, into a comprehensive suite enabling interoperability and reusability of web based learning content (Dodds 2003).

The IEEE Learning Technology Standards Committee (LTSC) has been working towards the development of technical standards and practices for learning technology (IEEE 2003). They have taken on the role of collating recommendations and proposals from other learning standardisation bodies and projects (Ferrandez 1998; Anido-Rifon, Fernandez-Iglesias et al. 2001) to develop these standards through working and study groups. These groups are in the areas of: Architecture and Reference Model, Digital Rights Expression Language, Computer Managed Instruction, Learning Objects Meta-data, Platform and Media Profiles, and Competency Definitions (IEEE 2003).

The IMS Global Learning Consortium is working towards building a set of open specifications for online distributed learning systems, and encourage their adoption to allow greater interoperability between the different distributed learning environments (IMS 2003a). Included in this suite is the IMS Question & Test Interoperability (QTI)

Specification, which provides a proposed standard using XML (refer to Appendix B for further information regarding XML) for describing questions and tests (IMS 2000).

It is suggested that the IMS QTI is the most advanced (Dalziel 2000) and has reportedly a large following of organisations implementing the specification (IMS 2003b). The IMS QTI Specification is designed to describe questions and tests to ensure interoperability of quiz test systems and development tools (IMS 2000). Figure 2.4 illustrates the use of the QTI Specification to express the rendering and basic response processing of the example MCQ in Figure 2.2 from Section 2.1.4, while Figure 2.5 represents the FIB question illustrated in Figure 2.3.

```
  <item ident="Static_MCQ_Example">
    <presentation>
      <flow>
        <material>
          <mattext>For the TCP/IP protocol, in which OSI relational model layer does
the tcp protocol belong?</mattext>
        </material>
        <response_lid ident="MC01">
          <render_choice shuffle="No">
            <response_label ident="A">
              <flow_mat>
                <material>
                  <mattext>Layer 1</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="B">
              <flow_mat>
                <material>
                  <mattext>Layer 2</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="C">
              <flow_mat>
                <material>
                  <mattext>Layer 3</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="D">
              <flow_mat>
                <material>
                  <mattext>Layer 4</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="E">
              <flow_mat>
                <material>
                  <mattext>Layer 5</mattext>
                </material>
              </flow_mat>
            </response_label>
          </render_choice>
        </response_lid>
      </flow>
    </presentation>
    <resprocessing>
      <outcomes>
        <decvar defaultval="0"/>
      </outcomes>
      <respcondition>
        <conditionvar>
          <varequal respident="MC01">D</varequal>
        </conditionvar>
        <setvar action="Add">1</setvar>
      </respcondition>
    </resprocessing>
  </item>
```

Figure 2.4 Example QTI xml for multiple choice question.

```
<item ident="FIB_Example">
  <presentation>
    <flow>
      <material>
        <mattext>What is 5 multiplied by 6? </mattext>
      </material>
      <response_num ident="FIB1">
        <render_fib fibtype="Integer">
          <response_label ident="A"/>
        </render_fib>
      </response_num>
    </flow>
  </presentation>
  <resprocessing>
    <outcomes>
      <decvar defaultval="0"/>
    </outcomes>
    <respcondition>
      <conditionvar>
        <varequal respident="FIB1">30</varequal>
      </conditionvar>
      <setvar action="Add">1</setvar>
    </respcondition>
  </resprocessing>
</item>
```

Figure 2.5 Example QTI xml for fill in the blank question.

Although the specification is quite advanced, it still supports further extensibility for functionality that is not included in the specification, such as parameterised quiz questions, without compromising the rest of the specification (Smythe and Roberts 2000). Yet, this is at the cost of interoperability where those extensions are used.

In conclusion, there are many standards organisations working diligently to develop new learning technology standards. Work by these standards organisations include: SC36 of the ISO (ISO 2003), the SCORM project undertaken by ADL (Dodds 2003), the efforts by the LTSC of the IEEE group (IEEE 2003), and the QTI and related specifications as developed by IMS (IMS 2003a). Of this work, the QTI specification of IMS is considered the most advanced (Dalziel 2000). These standards are necessary due to the various formats in which educational materials are defined, structured, and presented resulting in problematic re-use of application components embedded in learning systems (Ferrandez 1998; Crofts 1999; Dalziel 2000; Anido-Rifon, Fernandez-Iglesias et al. 2001; Burguillo, Benlloch et al. 2001).

2.3 Parameterised Quiz Questions

Parameterised quiz questions have been an active area of research due to the benefits they provide to both the author and the student. These benefits will now be discussed.

2.3.1 Benefits of Parameterised Quiz Questions

One of the benefits observed from parameterised quiz questions is their usefulness as a learning tool through formative assessments (Crofts 1999; Pathak and Brusilovsky 2002). By allowing students to repeatedly practice solving problems through the same question, the repeated attempts of the question are more meaningful to students providing variation of material, rather than promoting rote learning (Lundquist 2001). This can be illustrated through the example FIB question in Figure 2.1 of Section 2.1.3, in which the student must perform multiplication on 2 values. By parameterising this question, the student can repeatedly attempt the question and still be challenged, as the values of the multiplication would continually change, yet the task being learned is still multiplication.

Another benefit afforded by the use of parameterised quiz questions is the prevention or reduction of cheating through students sharing quiz solutions (Crofts 1999; Lundquist 2001; Ashton and Beevers 2002; Pathak and Brusilovsky 2002). Lundquist asserts that since a particular answer set cannot be shared in any two quiz sessions, it has been found that students were less likely to answer questions for another (Lundquist 2001). As a result, parameterised quiz questions also allow greater re-use of the same questions within summative assessment (Crofts 1999; Pathak and Brusilovsky 2002), which results in more efficient use of material, but in terms of time and cost.

Despite the many benefits provided by parameterised quiz questions, they are not without problems.

2.3.2 Issues Relating to Parameterised Quiz Questions

Paterson (2002) suggests that careful consideration of the instantiated parameters needs to be taken, to ensure consistency in the questions generated, so as not to open the question to different interpretations. Furthermore, parameterisation is most effective in assessing objective tasks requiring some form of algorithmic processing that can be easily automated through computer software. Another common problem with online quiz systems is lack of interoperability of assessment material (see Section 2.2.3). This problem also extends to parameterised questions. Although work by the IMS for example, is leading the way in developing specifications to ensure interoperability of quiz question material, unfortunately the specification does not provide interoperable support for parameterised questions.

By introducing support for parameterised questions into the IMS QTI specification, all the benefits of parameterisation can be shared and re-used amongst heterogenous quiz systems, rather than being limited to proprietary or custom built systems. The next chapter will provide an overview of the existing IMS QTI Specification.

3 IMS Question & Test Interoperability Specification

The IMS Question and Test Interoperability (QTI) Specification is designed to describe questions and tests to ensure interoperability of quiz test systems and development tools (IMS 2003a). Version 1.2 of the QTI Specification was released February 2002 (IMS 2002b). An addendum to the specification, version 1.2.1 was released in March, 2003 (IMS 2003c). This chapter will give a brief introductory overview of the concepts contained in those documents. Furthermore, it will also identify components within the existing specification structure that will require alteration to support the work reported in this document. For more detailed discussion of the IMS QTI Specification, refer to publications from IMS including: "IMS Question & Test Interoperability: ASI XML Binding Specification" and "IMS Question & Test Interoperability: ASI Best Practice & Implementation Guide" which are available from the IMS website (www.imsglobal.org).

3.1 The purpose/scope of QTI.

The IMS Global Learning Consortium's QTI Specification provides a standard format for recording questions, tests, and the subsequent scoring results (IMS 2003a). Through adoption of this specification, different Learning Management Systems (LMS) such as WebCT (www.webct.com) and Blackboard (www.blackboard.com) can exchange questions, tests, and results, allowing greater ease in sharing assessment content, and reducing the problems associated with LMS migrations.

The design concept behind the QTI is to allow interoperability of assessment content between different LMS, without constraining innovation by LMS vendors. The specification supports extensions providing proprietary functionality for LMS vendors

to gain an edge in the market place. Although these extensions are not interoperable with other systems, at least the remaining core content of the system remains so. As innovation in this area continues to grow, so too can the functionality incorporated into the QTI. For example, IMS plan to investigate the inclusion of adaptive sequencing into later releases of the specification (IMS 2002c). Those who will benefit from this specification include: publishers, teachers, certification authorities, or anyone else responsible for creation of assessment (IMS 2002h).

In the next section, the structure of the QTI Specification will be examined.

3.2 QTI Structure

The QTI is one of many specifications developed by IMS. Other specifications include: Meta-data Specification, Content Packaging Specification, Learner Information Packaging Specification and Sequencing Specification (IMS 2002a). Storage of meta-data associated with learning materials is the responsibility of the IMS Meta-data Specification. The IMS Content Packaging Specification provides the ability to describe and package learning materials (IMS 2004b), while storage of information specifically related to the learner is standardised through the IMS Learning Information Package Specification (IMS 2004a). Finally, the IMS Sequencing Specification defines a standardised method to record the order in which material is presented (IMS 2004c). Great effort has gone into ensuring there is integration between each of the IMS specifications, where appropriate, to reduce redundancy of effort and improve coupling between the related data sets (IMS 2002a). Figure 3.1 represents the basic relationships between the QTI and other IMS specifications:

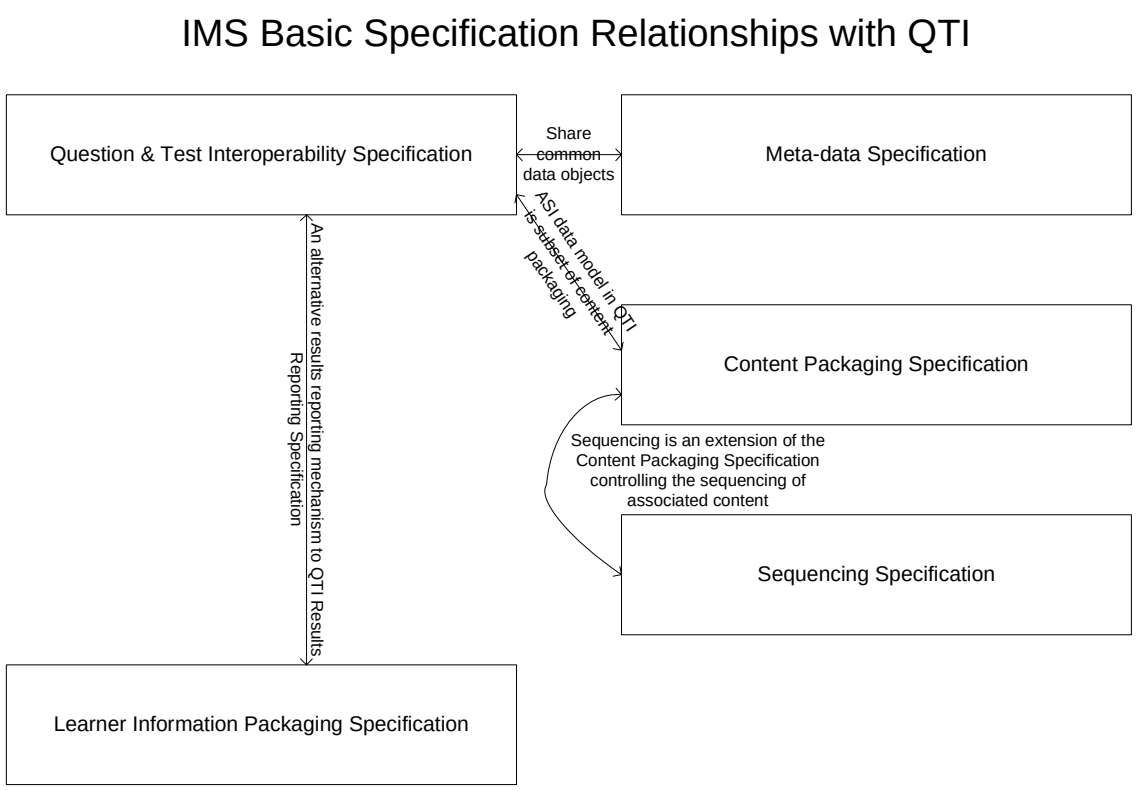


Figure 3.1 Basic relationships between QTI and other IMS Specifications.

Furthermore, the IMS QTI Specification is also made up of sub-specifications that provide certain modular functionality to the specification as a whole. This is represented in the Figure 3.2. The shaded sub-specifications would not be directly impacted by the implementation of parameterisation.

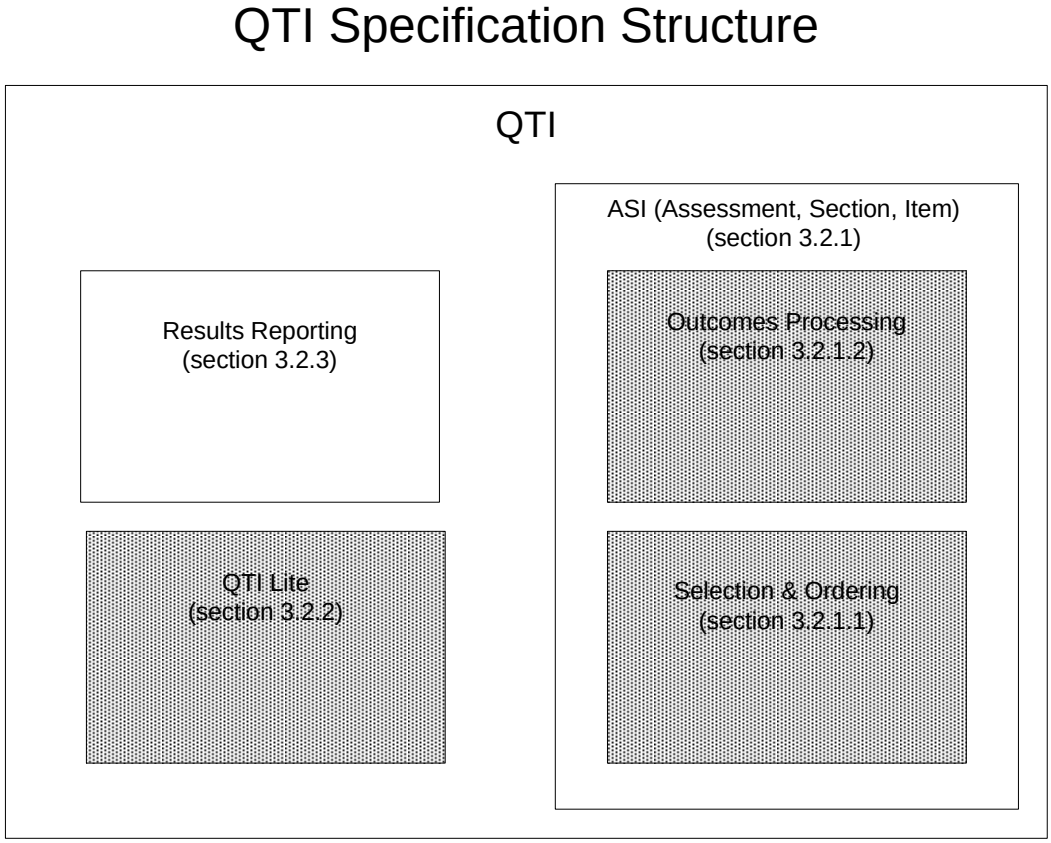


Figure 3.2 QTI Specification structure.

The ASI Specification stores quizzes in terms of assessments, sections and items. It also comprises of further sub-specifications. The Selection and Ordering Specification expresses how questions should be selected for inclusion in an assessment, and how they should be sequenced. The Outcomes Processing Specification describes the method in which marks are aggregated to determine a final grade for an assessment. As a cut-down version of the QTI specification, the main objective of the QTI Lite Specification is to provide a minimalist set of functionality to allow interoperability between other QTI Lite compliant systems. Finally, the storage of the final outcomes from assessments is specified through the Results Reporting Specification. A more detailed explanation of each of these sub-specifications follows.

3.2.1 Assessment, Section & Item

The Assessment, Section and Item (ASI) Specification is an integral component of the QTI Specification. As its name suggests, it is responsible for recording data relating to assessments, sections, items, and additionally object-banks. Figure 3.3 illustrates all the different possible arrangements for these data object containers.

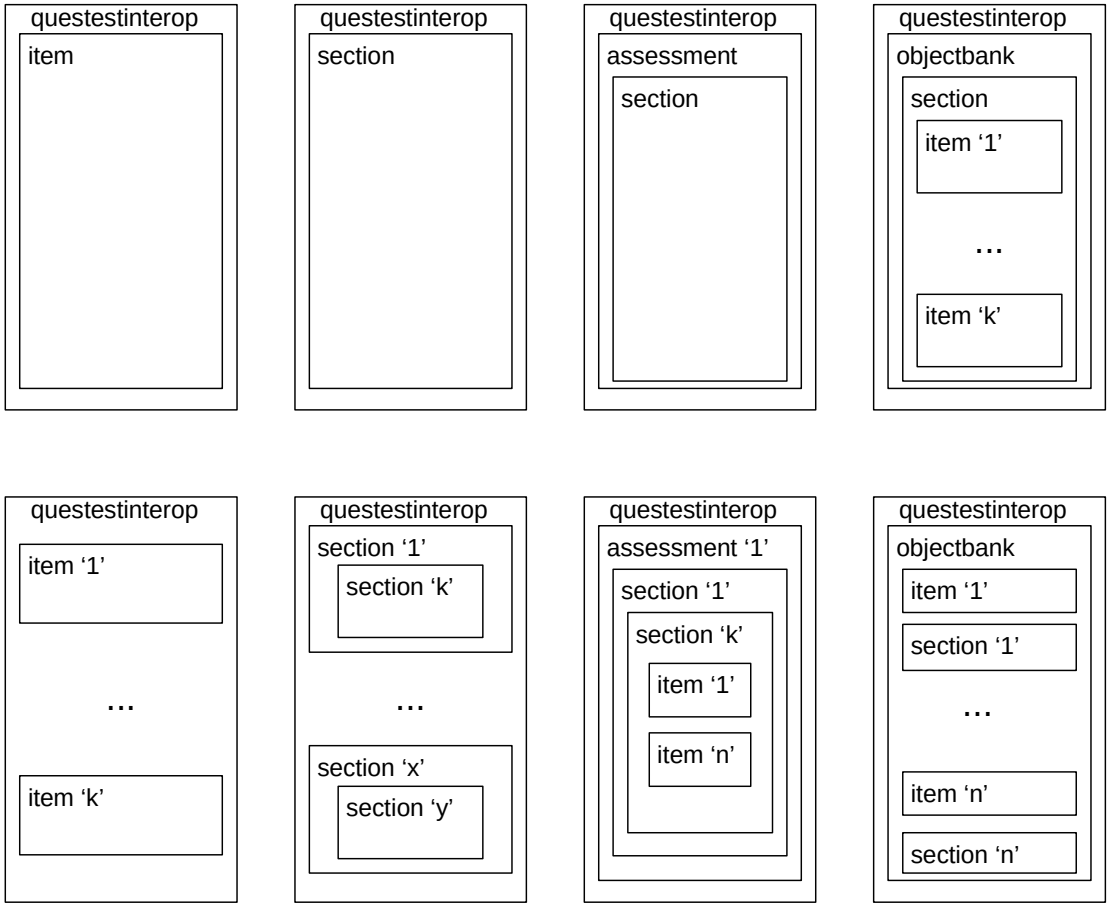


Figure 3.3 ASI interchange data containers reproduced from (IMS 2002a).

An assessment is synonymous with a test meaning it represents all data necessary to present, process and provide feedback associated with a particular assessment piece. An assessment is made up of zero or more sections or items. A section allows the logical grouping of one of more sections or items into categories. It is also useful to constrain the extent of sequencing instructions as per the Selection and Ordering Specification. An item is synonymous with a question and is the smallest

exchangeable object within the QTI. It is responsible for recording all the information related to the type and rendering of a question, the results processing of the candidates response, the feedback to the candidate, and any meta-data pertaining to the question. Finally, another core object of the ASI model is the object-bank. Essentially, an object-bank is a question data-bank; a repository of grouped or ungrouped question items. The contents of an object-bank can be used to construct assessments.

The aforementioned components relate to the core functionality of the ASI Specification. However, as shown in Figure 3.2, the ASI Specification is comprised of further sub-specifications that provide additional functionality to the core.

Selection & Ordering

The selection and ordering specification details how questions should be selected for inclusion in an assessment, and how those selected questions should be sequenced. The specification currently supports two sequencing rules with the intention that further rules will be added in future releases (IMS 2002c). The existing rules are sequential and random. Therefore, a given group of questions can be ordered such that they are shown sequentially (as recorded in the XML document), or ordered randomly. There is great flexibility in selection of questions for a test. This includes selection of (IMS 2002c):

- all identified questions,
- all identified questions, but limited by criteria set for question meta-data,
- a random subset of the identified questions,
- a random subset of the identified questions, also limited by criteria set for question meta-data,

- logical associations between questions, where one question is based upon another.

The inclusion of parameterised questions within the QTI has no direct impact or relationship on the Selection and Ordering Specification; meaning changes to accommodate parameterisation will not affect/break the existing Selection and Ordering Specification. Therefore, the Selection and Ordering Specification will not be examined in this research.

Outcomes Processing

The Outcomes Processing Specification details how the combined marks at the assessment and section levels are calculated (IMS 2002d). An item is responsible for determining the mark assigned based on the response processing of the candidate's answer. The Outcomes Processing Specification details exactly how those marks are collated at the parent levels of the assessment, from the section levels through to the assessment level. For example, two different sections within a complete assessment may have different weightings on their scores. The Outcomes Processing Specification effectively describes how the marks assigned at different parts of the assessment hierarchy should be aggregated to determine a total result.

The functionality supported by the specification is divided into two categories (IMS 2002d):

- Built-in result processing algorithms;

  This category describes the method by which the result processing algorithm is just named so the actual assessment engine is responsible for performing the

calculation. The named algorithms are inherently defined in the specification and the assessment engine is responsible for supporting as such. Examples of the built in algorithms follow.

- Proprietary algorithms;

  A parameterised approach is used to define the relationship between the input and output variables associated with the assessment through XML. This allows greater freedom to define custom algorithms for calculating the results, yet still interoperable as it is defined through the specification.

Some of the built-in result processing algorithms that are supported by the specification are (IMS 2002d):

- count of the number of correct answers,
- count of the number of correct answers, normalised with respect to those objects that have been attempted and not just selected and presented,
- weighted number that is correct, including multivariate responses,
- negative scoring, which negates any possible benefit from guessing answers.

The inclusion of parameterised questions within the QTI has no direct impact or relationship on the Outcomes Processing Specification; meaning changes to accommodate parameterisation will not affect/break the existing Outcomes Processing Specification. Therefore, the Outcomes Processing Specification will not be examined any further.

3.2.2 QTI Lite

The QTI Lite specification is a "cut-down" version of the complete specification allowing a minimalist set of functionality to be interoperable with other QTI Lite compliant systems. The QTI Lite specification has some limitations compared to the full specification such as (IMS 2002e):

- limited question types:
    - Yes/No,
    - True/false,
    - Likert scale[1],
    - Any other form of MCQ,
- simple response processing providing for only a single right answer,
- no support for (IMS 2002e):
    - hints and solutions,
    - meta-data,
    - comments,
    - extensions,
    - options that are "fuzzy",
    - all time-based mechanisms.

The QTI Lite Specification does not support the assessment or section models of the full QTI Specification. Therefore, its main purpose is to provide a minimalist specification for exchanging questions (items) between assessment systems, rather than describing complete assessments. This coupled with the fact that only a limited

[1] The Likert scale refers to the level of agreement one has with a statement, usually expressed as one of: Strongly Agree, Agree, Disagree, and Strongly Disagree.

number of question types are supported, means it would not be worthwhile to implement parameterisation, as compared to the full specification. Therefore, the QTI Lite Specification will not be examined further in this research.

3.2.3 Results Reporting

The Results Reporting Specification is responsible for describing the structures for storing the resulting outcomes from QTI assessments. This is to allow interoperability of information recorded which includes (IMS 2002f):

- assessment context,
- assessment summary,
- assessment results,
- section results,
- item results.

Table 3.1 describes the particular uses of the Results Reporting Specification (IMS 2002f).

Table 3.1 IMS Results Reporting Specification Purpose

Uses	Description
Permanent storage	Long term storage of candidate results, perhaps to meet auditing requirements.
Results aggregation	Assembly of results sets from multiple disparate assessment delivery systems, and the transfer of the assembled results to an analysis system
Processed results for a LMS	The exchange of results between different LMS.
Real-time interactive	Allow real-time interactive exchange of results information between different systems
Internal Interchange Data Language Representation	For systems designed on interoperable modular processes for implementation of the QTI Specification, an interchange representation is provided to allow inter-process communication between different implementations of the modular processes.

The inclusion of parameterised questions within the QTI Specification will have a direct impact and relationship with the Results Reporting Specification. The instantiated parameters would need to be stored along with other results information for each candidate attempt. Furthermore, the Results Reporting Specification provides support for real-time data exchange between different processes of the assessment engine through the use of the Internal Interchange Data Language (IIDL). This language would need to be aware of parameters within an Item, to facilitate inter-process communication with other assessment related sub-systems. Refer to the document: IMS Question & Test Interoperability: Results Reporting Information Model (IMS 2002f) for further explanation on IIDL and the Results Reporting Specification.

In summary, the ASI Sub-specification is responsible for recording data relating to assessments, sections, and items. An assessment is made up of zero or more sections or items; a section allows the logical grouping multiple sections or items into categories; and an item is synonymous with a question, which is the smallest exchangeable object within the QTI (IMS 2002g). The QTI item will be discussed further in the following section.

3.3 The QTI Item

As explained in Section 3.2.1, an item is responsible for recording all the information related to the type and rendering of a question, the results processing of the candidates response, the feedback to the candidate, and any meta-data pertaining to the question. Each of these functions will now be examined.

3.3.1 Question Type and Rendering

Question types (as outlined in sections 2.1.4 and 4.7) are often considered a benchmark for comparison of LMS capabilities, although quite often, vendors will count different combinations of the same question type to increase their share (Paterson 2002). The QTI Specification classifies question types based on the response type of the question. The response type identifies how the candidate will respond to a particular question, whether it is selection based as in a MCQ, or a fill in the blank.

The QTI Specification has a multi-tiered classification for response types. This is illustrated in Figure 3.4 reproduced from the IMS QTI ASI Information Model Specification.

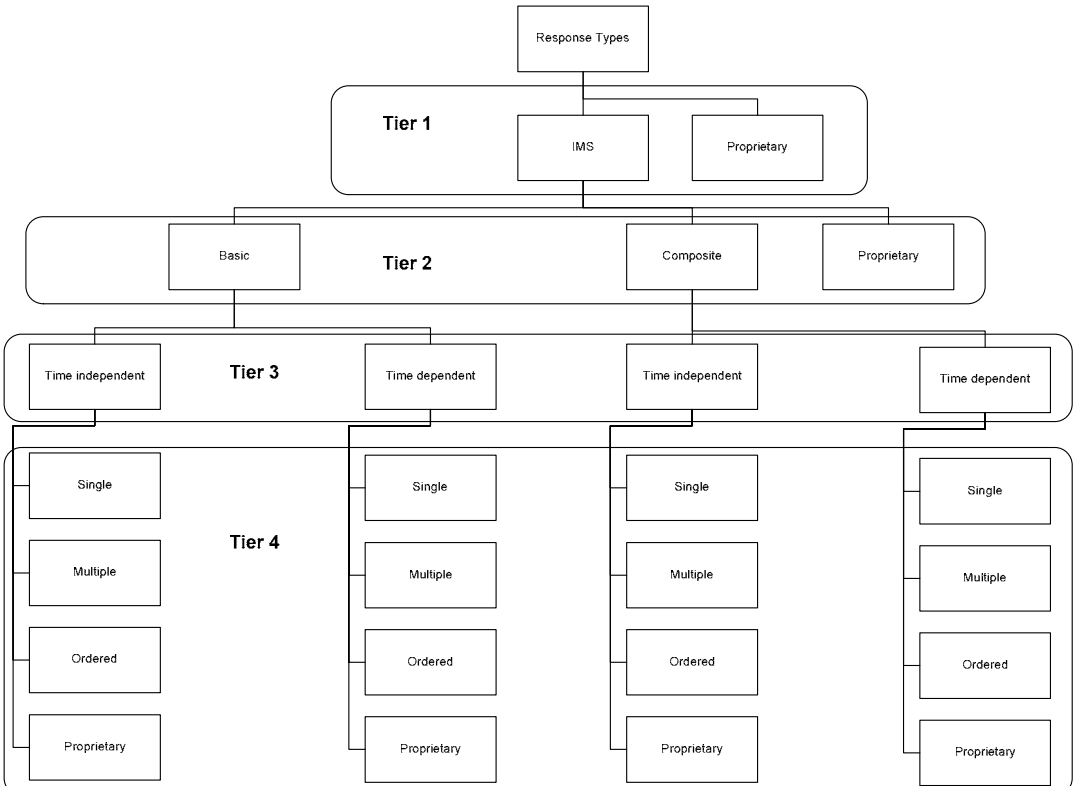


Figure 3.4 Multi-tiered classification for response types reproduced from (IMS 2002g).

On the first tier, the response type may be IMS based, or some proprietary extension. On the second tier, the specification categorises whether a question is of a:

- "basic" response type which only contains a single response type,
- "composite" response type which uses combinations of the basic types,
- "proprietary" response type providing for vendors specific extensions.

A question may be time dependent on the third tier, meaning that the time taken by the candidate to provide a response is significant to the question; or otherwise it is not time dependent. The fourth tier includes response types that can be broken down into the following categories.

- *single*- A single response is only required of the candidate.
- *multiple* - One or more responses is required of the candidate.
- *ordered* - One or more responses is required of the candidate, where the response order is significant.
- *proprietary* - As with other tiers within this model, provision for proprietary extensions is made.

A composite question at the second tier is made possible by allowing a combination of the aforementioned responses. As a result, each of Single, Multiple, Ordered, and Proprietary will have more than one basic response type.

The basic response types supported by the QTI Specification are illustrated in Table 3.2.

Table 3.2 QTI basic response types reproduced from (IMS 2002g)

Response Type	Data Structure	Rendering Formats		
		Single	Multiple	Ordered
Logical Identifier	The response-type identity or list of identities. The order of the list is first choice, second choice, etc.	Multiple choice True/false Slider	Multiple response	Order objects Connect-the-points Match object Order object Drag object Drag target
X-Y Co-ordinates	The 'x-y' co-ordinates of the centre of the object(s) for each response identity or a list of 'x-y' co-ordinates. The order of the list is first choice, second choice, etc.	Image hot spot	Order objects	Connect-the-points
String	The typed string for each response identity.	Fill-in-blank Select text Short answer Essay		
Numerical	The entered number for each response identity.	Fill-in-blank Slider		

The rendering format identifies how the question will be displayed (formatted) to the candidate. The response type classification is independent of the rendering methods for a question (IMS 2002g). For example, a response type may be a logical identifier representing a selection list. Yet, the rendering may be done as a text list (i.e. MCQ), or as an image hot-spot.

The rendering methods supported by the QTI Specification are (IMS 2002a):

- *multiple choice* – The candidate must select one or more options as the solution,
- *image hot-spot* – The candidate must select a region of an image as the solution,
- *slider* – The candidate must select an ordinal value (typically a number) by sliding a marker between two points, thus selecting a value as the solution,
- *object* – The candidate must manipulate a computer displayed object, such as altering its position in relation to other objects as the solution,
- *fill in the blank* – The candidate must provide a textual response as the solution.

In summary, the QTI Specification has a multi-tiered classification for response types, which allows basic, composite and proprietary response types, and combinations thereof, and provides for proprietary extensions. A myriad of rendering formats are also supported, such as MCQ and FIB.

3.3.2 Item Response/Results Processing

Part of the item component of the specification is the description of how to process the candidate's response, and how that response should be marked. This description is provided in the XML Schema designed for the Item element. The specification provides for comparison of the candidate's response, with what is expected for a correct or partially correct answer, through the logical operations presented in Table 3.3:

Table 3.3 QTI response processing logical operators

Logical Operator (XML Elements)	Purpose
varequal	To test if the candidate response is equal to a given value
varlt	To test if the candidate response is less than a given value
vargt	To test if the candidate response is greater than a given value
varlte	To test if the candidate response is less than or equal to a given value
vargte	To test if the candidate response is greater than or equal to a given value
varsubset	To test if the candidate response matches one within a list of possible values
varinside	To test if the candidate response is within a given x,y co-ordinates area
varsubstring	To test if the candidate response is a sub-string of a given value
durequal	To test if the candidate has provided a response in a time equal to a given value
durlt	To test if the candidate has provided a response in a time less than a given value
durgt	To test if the candidate has provided a response in a time greater than a given value
durlte	To test if the candidate has provided a response in a time less than or equal to a given value
durgte	To test if the candidate has provided a response in a time greater than or equal to a given value
not	To perform a logical “not” of the given operators
and	To perform a logical “and” of the given operators
or	To perform a logical “or” of the given operators
unanswered	To test if the candidate did not provide a response for the item
var_extension	To provide for proprietary extensions to the response processing
other	To match when no other logical operators have matched. This could be used as a simple logical else condition where nothing previous has matched

Scoring variables can be declared, which are manipulated based on the matching of these logical operators. The supported manipulations on the scoring variables are: set to value, addition, multiplication, subtraction, and division. For example, if a match is found with *varequal* based on the candidate’s response that is correct, then a value of one could be added to a scoring variable. This scenario is illustrated in Figure 3.5.

```
<questestinterop>
  <item ident="example_scoring">
    ...
    <resprocessing>
      <outcomes>
        <decvar varname="SCORE"></decvar>
      </outcomes>
      <respcondition>
        <conditionvar>
          <varequal respident="FIB1">20</varequal>
        </conditionvar>
        <setvar varname="SCORE" action="Add">1</setvar>
      </respcondition>
    </resprocessing>
  </item>
</questestinterop>
```

Figure 3.5 Example of scoring with IMS QTI.

If the response labelled “FIB” from the candidate has a value of 20, then add one to the “SCORE” scoring variable. If a variable name is not provided, then the default name “SCORE” is assumed.

3.3.3 Item Feedback

The QTI provides support for feedback to the candidate after completing an item. This is particularly useful when using assessment in a formative way. The candidate will receive useful feedback on the item and on their answer as well. This feedback can be in the form of hints in response to the answer provided, or the actual solution to the problem. In a summative assessment, it may be simple feedback such as correct, incorrect, or could be something more specific and detailed. Furthermore, a view can be specified that determines what feedback is shown to which user. For example, solution feedback could be recorded as viewable only to the tutor; thus ensuring the candidate is unable to view the answers to the assessment when completing summative assessment.

3.3.4 Item Meta-data

A great deal of meta-data (see Section 2.1.3) can be recorded for an item through the QTI Specification. This meta-data can also be used for selection and sequencing as per the Selection and Ordering Sub-specification (see "Selection and Ordering" from Section 3.2.1).

Table 3.4 details the meta-data supported for an Item and its purpose as per IMS ASI Information Model Specification:

Table 3.4 QTI Item meta-data reproduced from (IMS 2002a)

Item Meta-data Vocabulary	Purpose
qmd_absolutescore_max	The maximum score that the candidate may attain
qmd_absolutescore_min	The minimum score that the candidate may attain
qmd_computerscored	Whether or not the item can be computer scored
qmd_feedbackapermitted	Whether or not the feedback is to be made available
qmd_hintspermitted	Whether or not the hints are to be made available
qmd_itemtype	The type of Item
qmd_levelofdifficulty	The level of difficulty of the Item
qmd_material	Listing of the types of content supplied in the Item
qmd_penaltyvalue	The penalty value that is to be used with the "Guesspenalty" outcomes processing algorithm
qmd_questiontype	The type of question
qmd_renderingtype	The type of rendering employed
qmd_responsetype	The class of responses required by the Item
qmd_scorereliability	The reliability metric that has been assigned to the score of this evaluation object
qmd_scorestderr	The standard error that has been assigned to the score for this evaluation
qmd_scoringpermitted	Whether or not scoring is enabled
qmd_solutionspermitted	Whether or not the solutions are to be made available
qmd_status	The status of the Item (Experimental, Normal, Retired)
qmd_timedependence	Whether or not the candidate responses are time dependent
qmd_timelimit	The number of minutes or an unlimited duration
qmd_toolvendor	Name of the vendor of the tool creating the assessments
qmd_topic	A brief description of the topic covered by the Item
qmd_typeofsolution	The type of solution supplied by the Item
qmd_versionnumber	A string used to define the version number of the Item
qmd_weighting	The weighting of the item for scoring

3.3.5 Item Miscellany

An Item may also contain other useful information. This includes: objectives in completing the item, pre and post conditions on selection of items, rubrics, and shared presentation content (amongst multiple items).

3.3.6 Item XML Schema

This and following sections will provide an overview of the XML Schema for the *item* element, as per the ASI Sub-specification. The *Item* element structure is the only component of the ASI Sub-specification that will require changes to implement parameterisation into the QTI.

Throughout, diagrams from the "IMS: ASI XML Binding Specifications" document (IMS 2002b) will be reproduced to illustrate the XML DTD structure of the QTI i*tem*, however for brevity the attributes have been omitted. Elements that are completely shaded will not be discussed further and will not be affected by the implementation of parameterisation into the QTI Specification, which is the focus of the work reported in this paper. Elements that are partly shaded will be affected by parameterisation, but are not essential components of the ASI Sub-specification to support parameterisation, and will not be examined further in this research. Finally, those elements that are not shaded are core components of the specification for supporting parameterisation, and will be discussed in the following sections. Refer to the legend as illustrated in Figure 3.6 as an example, demonstrating that the *presentation*, *resprocessing* and *itemfeedback* elements will be directly affected by parameterisation, yet the remainder will be completely unaffected.

The diagrams also represent the order in which sub-elements should be included, and the cardinality of such elements. This is represented by a circle and the character within. The question mark suggests there can only be one or zero of the given element; an asterisk zero or more; the plus sign one or more; and the number one suggests there must be one of that particular element. The lines connecting the parent and child elements specify the groupings in conjunction with the cardinality, such as the parenthesis grouping within the DTD standard. Unless a cardinality symbol is adjacent to an element, it is assumed that there is to be one, and only one of that element. Discussion will commence with the top level *item* element. After discussion of each of the item element structures, Section 3.3.17 provides XML examples, which demonstrate the use of these structures.

3.3.7 item element

Figure 3.6 illustrates the top-level structure of the i*tem* element. As explained in Section 3.2.1, an item essentially represents a single question entity. Therefore, all information relating to a particular question is recorded within an item structure.

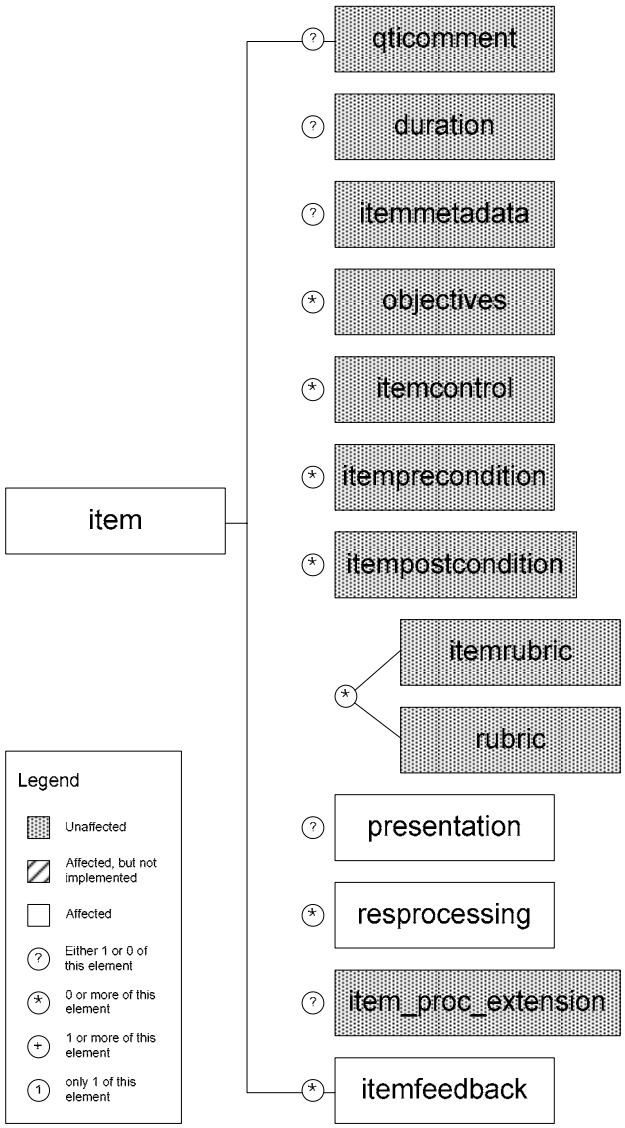


Figure 3.6 QTI item XML structure[1].

As with most of the major elements within the QTI, i*tem* allows a *qticomment* element, so the author can provide annotations. *Duration* determines the amount of time that a candidate has to complete this particular item. The *itemmetadata* structure provided a means for recording meta-data associated with a particular item. The use of this element is now depreciated and the *qtimetadata* element declared elsewhere should be used instead. The educational objectives of this particular item for the

[1] It is recognised that some of the elements within the QTI Item structure will be affected by the parameterisation of questions, however they will not be the main focus of this research.

candidate can be recorded within the *objectives* element. The *itemcontrol* element is a switching facility to either enable or disable the display of hints, solutions or feedback relating to the item. The *itemprecondition* and *itempostcondition* elements are to provide criteria for whether the current or next item should be activated, respectively. These two elements are for further study in newer releases of the QTI Specification. The *itemrubric* and *rubric* elements can contain useful information in the context of the item that may or may not be of assistance in answering the question. The *itemrubric* element is depreciated, superseded by *rubric*. *Item_proc_extension* provides for proprietary extensions to be implemented for response processing. The aforementioned elements would be unaffected by the implementation of parameterisation. However, this is not the case for the *presentation, resprocessing,* and *itemfeedback* elements.

The *presentation* element contains the relevant information required to present the item to the candidate. Its structure is illustrated in Figure 3.7. The *resprocessing* element as shown in Figure 3.13 describes how the response from the candidate should be processed, to determine an outcome. Finally, the *itemfeedback* element is responsible for recording all feedback information that should be presented as a result of the candidate's response. This feedback can be broken down into hints or solutions relating to the item. Providing feedback to a candidate after completing assessment is vitally important when using the assessment for formative purposes, although in a summative assessment, not so important. With parameters in an item, it may be necessary to use the instantiated values of the parameters within the feedback information. This would provide more specific help/guidance to the candidate. The next section takes a deeper look into the *presentation* element structure.

3.3.8 presentation element

The structure off the *presentation* element is shown in Figure 3.7.

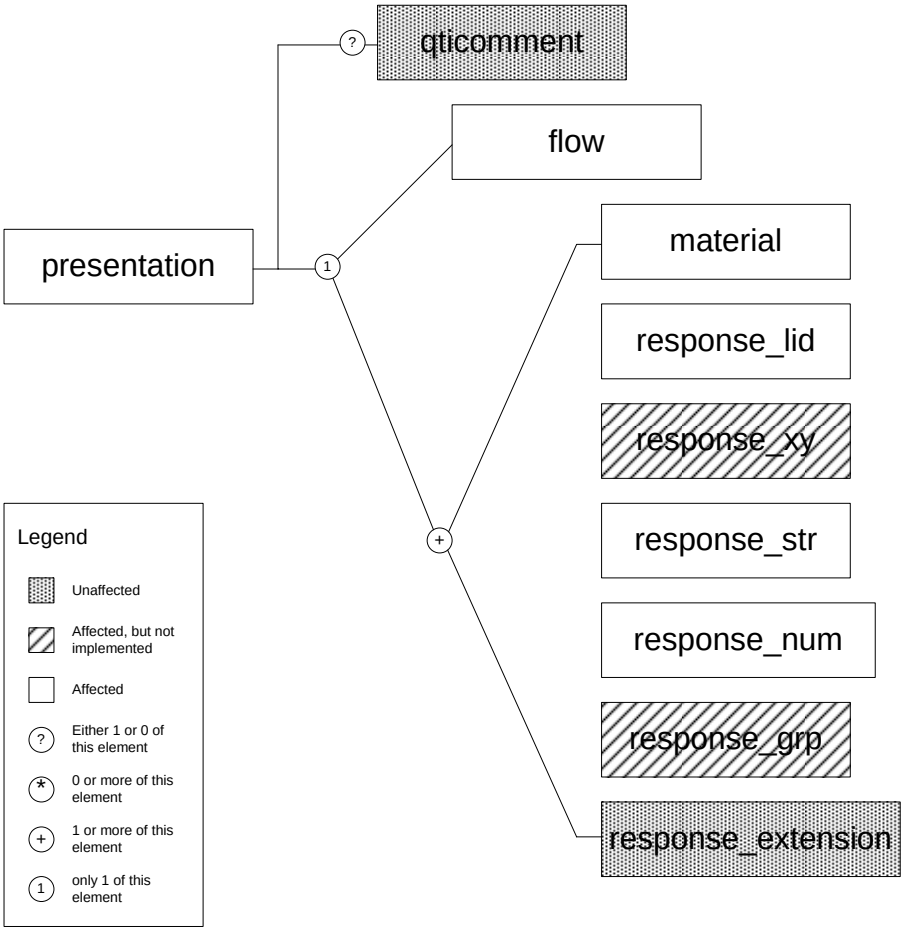


Figure 3.7 QTI presentation XML structure.

The *presentation* element is comprised of exactly one *flow* element. A flow within the QTI is a formatting object that is designed to allow the logical grouping of presentation material. The concept is anything within a flow is essentially grouped together, not unlike a paragraph. Therefore if it was necessary to have a line break within the material, use of separate flows would be appropriate. Flows may also be embedded within flows.

Within the *flow* element, one or more of the sub-elements shown in Figure 3.7 must be provided. The *material* element is responsible for containing all the different types

of material that is to be presented to the candidate to answer the question (see Section 3.3.12). The elements listed, starting with “response” determines the response type/s of the item, through which the candidate would provide their solution.

Parameterisation could also be implemented through the response elements *response_xy* and *response_grp* (see Chapter 7 - Conclusions).

3.3.9 response_lid element

This section will examine the first response element, *response_lid* as illustrated in Figure 3.8.

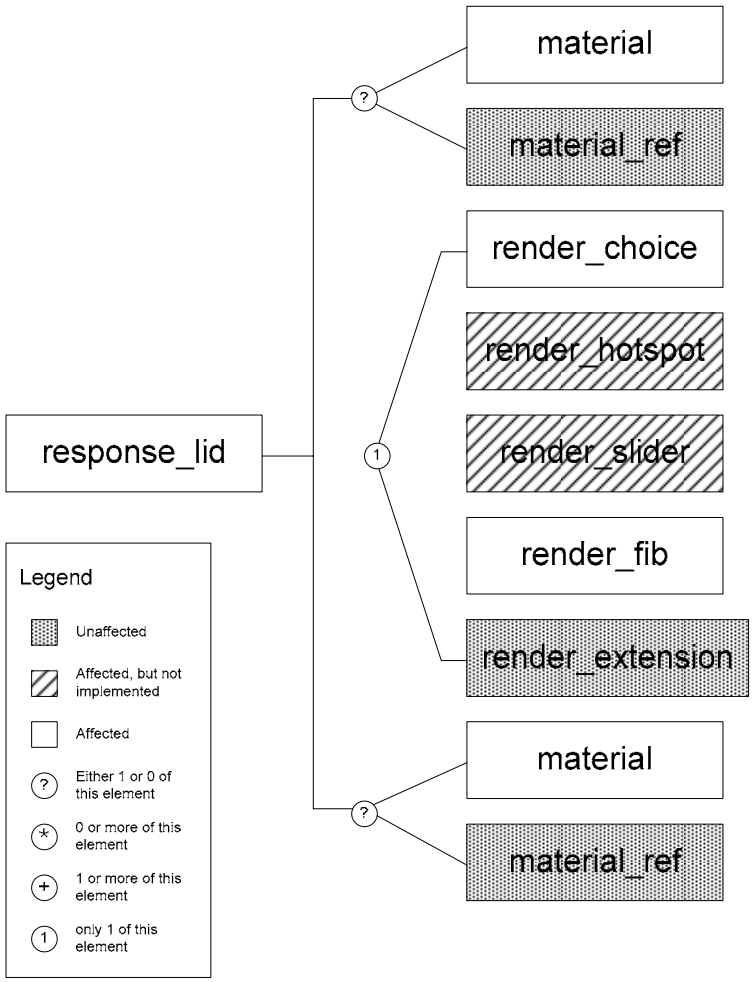


Figure 3.8 QTI response_lid XML structure.

The *response_lid* element describes and labels a response from the candidate, which is identified by its *ident* attribute. As an item may require multiple responses from the candidate, this *ident* attribute is used by the response processing component of the specification (see Section 3.3.16) to identify particular responses from the candidate, to perform logical tests for assessment. Other response type elements include *response_str* and *response_num* which expect to receive a response in the form of a string of characters or a numeric value respectively. The format of the response for the *response_lid* element is a logical identifier for a selectable value. All of the response type elements have an identical structure as shown in Figure 3.8 for the *response_lid* element. Within the response type elements, there must be a render type element, which describes how to render the response input widget to the candidate.

3.3.10 render_choice element

A typical render type element to be used with *response_lid* is *render_choice*. The *render_choice* element allows the rendering of multiple choices to the candidate. The structure of the *render_choice* element is shown in Figure 3.9. Each of *render_hotspot*, *render_fib*, and *render_slider* elements also has an identical structure.

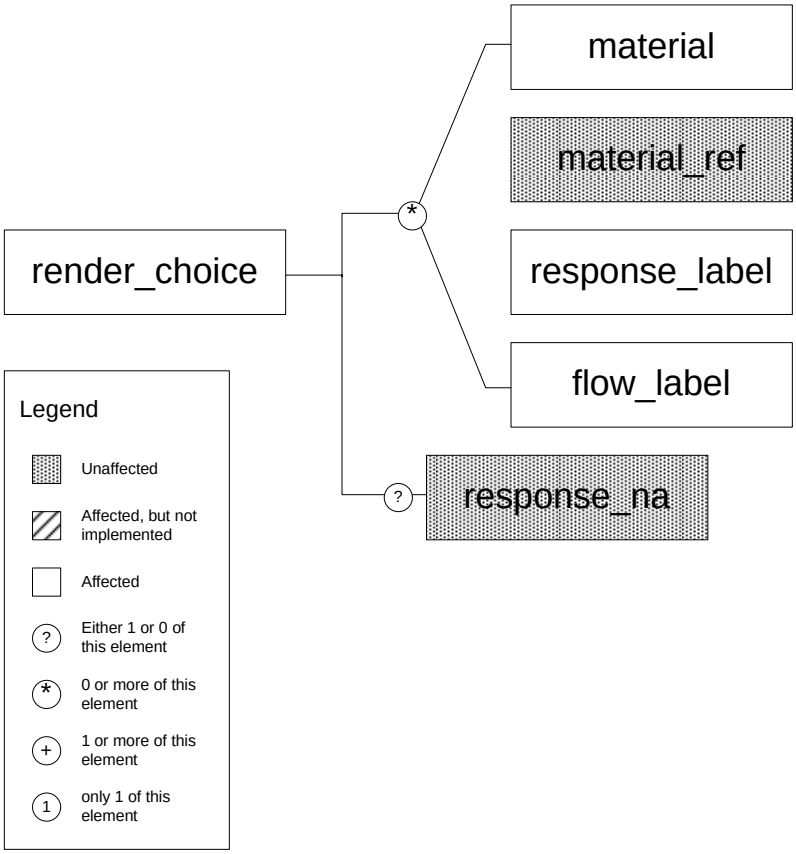


Figure 3.9 QTI render_choice XML structure.

For the *render_choice* element, each of the multiple choices presented to the candidate are represented by *response_label* elements (described further in Section 3.3.11). *Reponse_label* is responsible for uniquely identifying a particular selectable response available to the candidate as a logical identifier (LID), via its *ident* attribute. The *response_lid ident* attribute identifies a particular response from the candidate, and the *ident* attribute value of a *response_label* is assigned as the LID response. Figure 3.16 in Section 3.3.17 provides an example of a MCQ using the *response_lid*, *render_choice* and *response_label* elements. For the *response_str* and *response_num* elements, the typical render element would be *render_fib*, allowing the candidate to enter their response via a fill in the blank text box widget. This is demonstrated through Figure 3.18, which illustrated a FIB question using the *response_str* and *render_fib* elements.

The *flow_label* element is responsible for providing grouping of choices in the same way that the *flow* element does for the *presentation* element (see Section 3.3.8). This can provide separation between different groups of choices within a multiple choice question for example. See Figure 3.16 in Section 3.3.17 which demonstrates the use of *flow_label* elements.

Although only *render_choice* and *render_fib* have been discussed, parameterisation could also be implemented through the rendering elements *render_hotspot*, *render_slider*, and *render_extension* (see Chapter 7 - Conclusions).

3.3.11 response_label element

As described in the previous section, the *render_choice* element contains a *response_label* element, which is responsible for uniquely identifying a possible response provided by the candidate to the question. Its structure is shown in Figure 3.10.

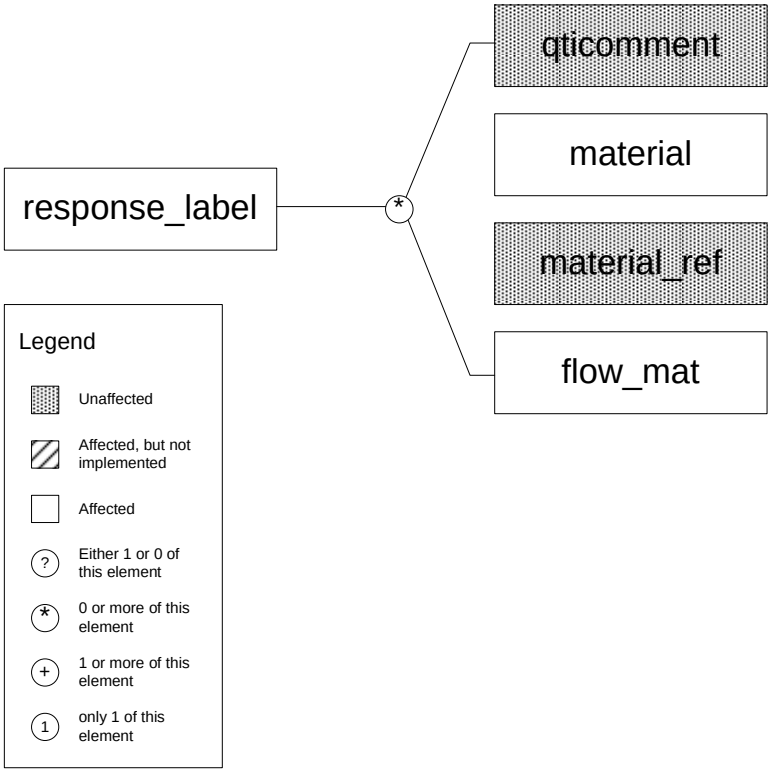


Figure 3.10 QTI response_label XML structure.

The type of response is determined by the parent render element. For example, if the *response_label* element is within *render_choice*, then the *response_label* will uniquely identify a particular choice option in a multiple choice question (see Figure 3.16). If it is within *render_hotspot*, then *response_label* will uniquely identify the hotspot area in which the candidate selected (refer to IMS Question & Test Interoperability: ASI Best Practice & Implementation Guide for examples). For certain rendering types, it may be necessary to provide material associated with the *response_label*, namely for choice rendering (refer to Figure 3.16 for an example).

The *flow_mat* element is responsible for providing grouping of material in the same way that the *flow* element does for the *presentation* element (see Section 3.3.8). This provides grouping/paragraphing for material associated with a choice response, such as the example shown in Figure 3.16.

3.3.12 material element

The *material* element is the core element responsible for describing content. It is used by many objects within the item structure to display a variety of media types. The structure of the *material* element is illustrated in Figure 3.13.

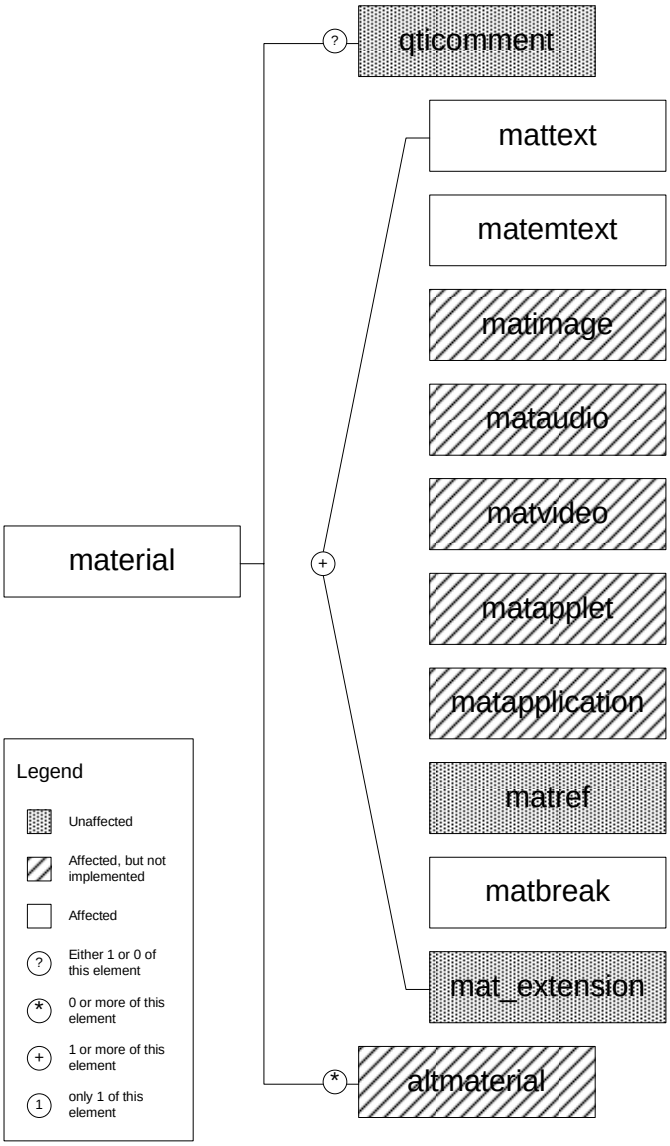


Figure 3.11 QTI material XML structure.

The media types supported by the *material* element include: text, emphasised text (for example bold typeface), images, audio, video, browser applets, and application launching. Manual line breaks can be inserted by including a *matbreak* element. The

material element must contain at least one of the aforementioned media types, but can contain more of the same or different types. For example, there may be a question described within a *mattext* element, which asks about an image within a *matimage* element. Both pieces of information can be provided within the one *material* element, where they would be presented using the same order as they are provided in the XML. It is the *material* element that will require direct changes to support parameterisation, in order to define how parameters are displayed within the question material.

3.3.13 itemfeedback element

The *itemfeedback* element describes feedback to provide to the candidate, identified by the attribute *ident.* It is a child element to the *item* element and can be used zero or more times to describe different feedback material. Which of the feedback material to be displayed is determined by the outcome of the response processing (see Section 3.3.14). The structure of the *itemfeedback* element is illustrated in Figure 3.12.

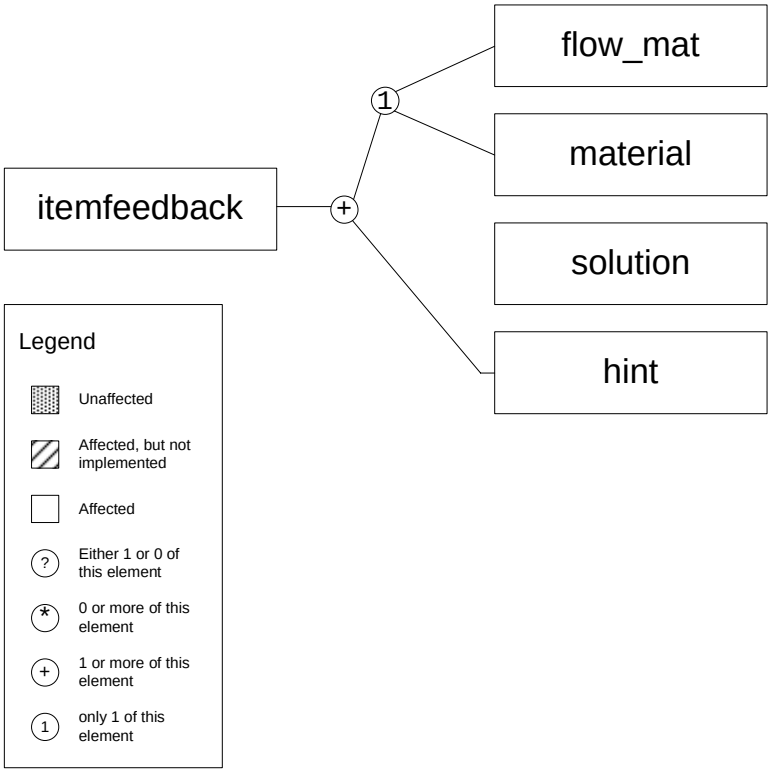


Figure 3.12 QTI itemfeedback XML structure.

The *flow_mat* element is responsible for providing grouping of material in the same way that the *flow* element does in the *presentation* element (see Section 3.3.8). This provides grouping/paragraphing for material associated with response feedback. Alternatively, a *material* element can be placed directly within the *itemfeedback* element, where no flow blocking will be used. This material is considered as response feedback to the candidate. Response feedback is used to provide feedback to the candidate, in response to their submitted solution. Typically, it can be as simple as "Correct" or "Incorrect". It can also be more specific, based specially on the response provided by the candidate. Figure 3.18 in Section 3.3.17 provides an example QTI item that uses a response *itemfeedback* element, with an *ident* "Correct". If this *itemfeedback* is selected to be displayed to the candidate, then the material "Yes, the season of Autumn." would be presented. The *itemfeedback* element can also describe hint and solution materials.

The *solution* element provides a structure in which *material* elements can be provided to describe the solution to this particular item. Likewise, the *hint* element can also provide *material* elements describing hints to the candidate in answering the item. For more specific details on the *hint* and *solution* elements, refer to the document: IMS Question & Test Interoperability: ASI XML Binding Specification (IMS 2002b). The commonality between all of these feedback types is that the feedback material is all described through the standard *material* element, as per Section 3.3.12.

3.3.14 resprocessing element

The elements that have been presented thus far have been related to the presentation component of the *item* element. Now the response processing component of the specification will be described, starting with the *resprocessing* element. Its structure is illustrated in Figure 3.13.

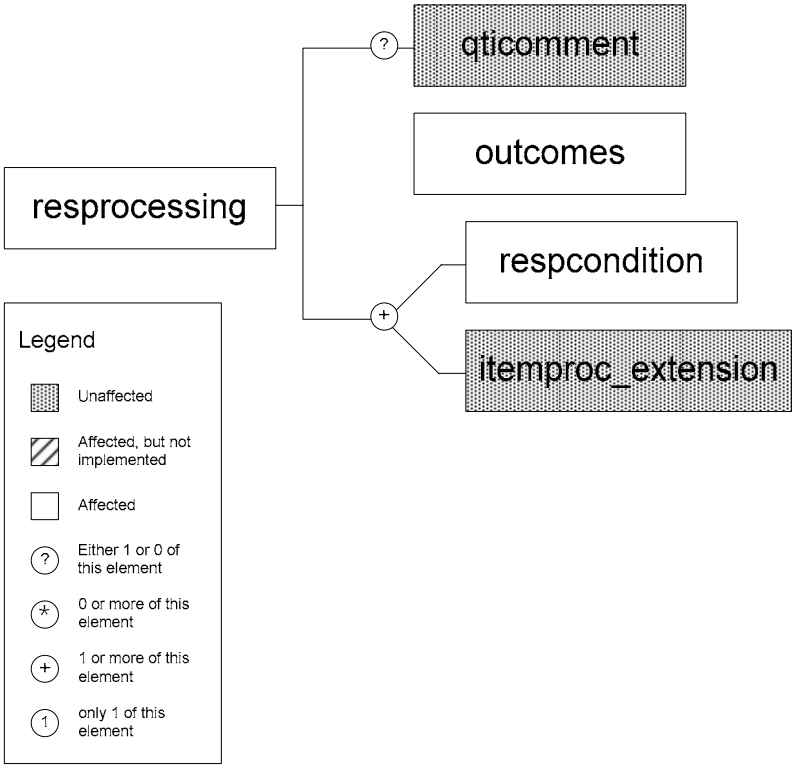


Figure 3.13 QTI resprocessing XML structure.

The *resprocessing* element is responsible for defining how the item should process the response/s provided by the candidate, in answering the item. It is basically broken down into 2 components: response conditionals, and resulting outcomes.

The *respcondition* structure is used to define how to process and compare the response/s provided by the candidate resulting in an assigned score, and feedback to the candidate; while the *outcomes* structure is responsible for defining what scoring variables are required by the response conditions in calculating a mark for the item. More than one *respcondition* element can be provided allowing for testing of different outcomes based on the responses provided by the candidate. For example, one *respcondition* could be used to determine if the correct answer was provided; then another could be used to determine if part of the answer provided is correct for part marks.

3.3.15 respcondition element

The *respcondition* element is illustrated in Figure 3.14. It must contain one *conditionvar* element, used to perform conditional tests on the responses provided by the candidate. If this evaluates to true, the *setvar* elements will be enacted, which will set the outcomes variables as declared in the *outcomes* element within *resprocessing.*

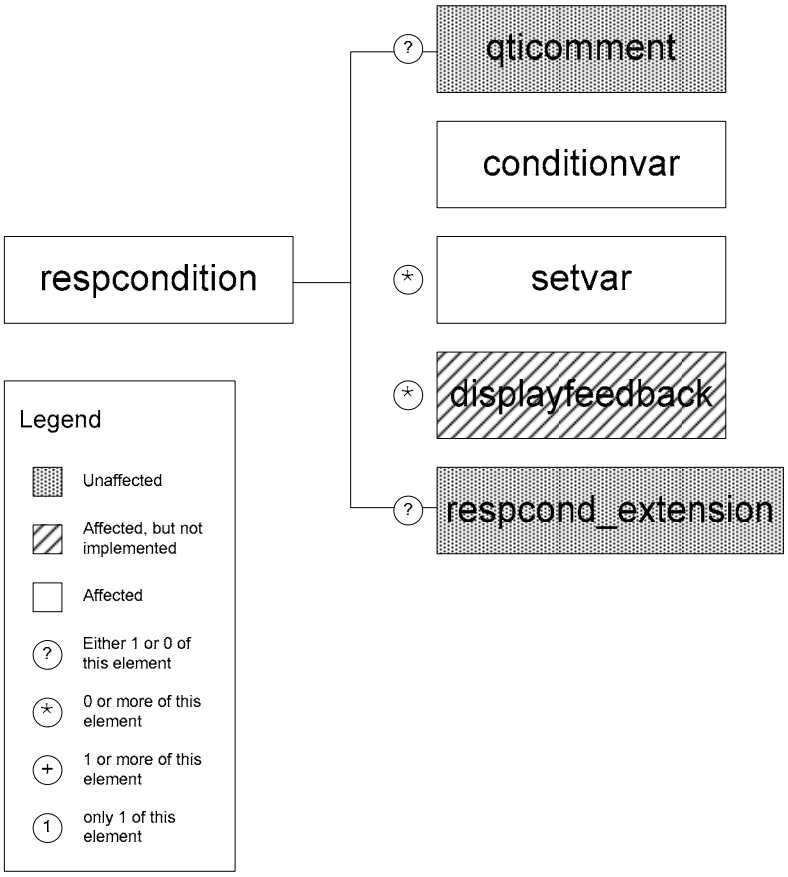


Figure 3.14 QTI respcondition XML structure.

The *displayfeedback* element identifies which material should be presented if the *conditionvar* evaluates to true. This way, solutions, hints, and response feedback can be provided to the candidate based on the evaluation of their response. See Figure 3.16 and Figure 3.18 in Section 3.3.17 for example items using *respcondition* elements.

3.3.16 conditionvar element

Figure 3.15 shows the structure of the *conditionvar* element. The *conditionvar* structure is responsible for evaluating conditional tests on the responses provided by the candidate. Where a *conditionvar* element matches the criteria within, the associated *setvar* statements will be enacted determining marks for the item (see Section 3.3.15).

The testing elements that are supported are shown in Figure 3.15, and are also described further in Table 3.3 of Section 3.3.2. For each of these testing elements, the attribute *respident* is used to identify which response element (see Section 3.3.9) is being compared with the information stored within the testing element's PCDATA (Parsed Character Data). In the example shown in Figure 3.18, the *response_str* element has an identifier of "FIB01", and is then compared with the string "Autumn" in the *varequal* element. In the case of a MCQ, Figure 3.16 shows the use of the *response_lid* element, which has an identifier of "MCb_01". The *varequal* element is then comparing this response "MCb_01" from the candidate with a label ident (LID) value of "B" being the correct selectable option. Therefore, in this example the candidate is correct if they selected the option "IEEE 802.5". This demonstrates how the response from the candidate is linked with the testing element and its data. These testing elements will be directly affected by the implementation of parameterisation, as the value to compare may not necessarily be a static value recorded as PCDATA, but will need to be represented by other elements identifying parameters.

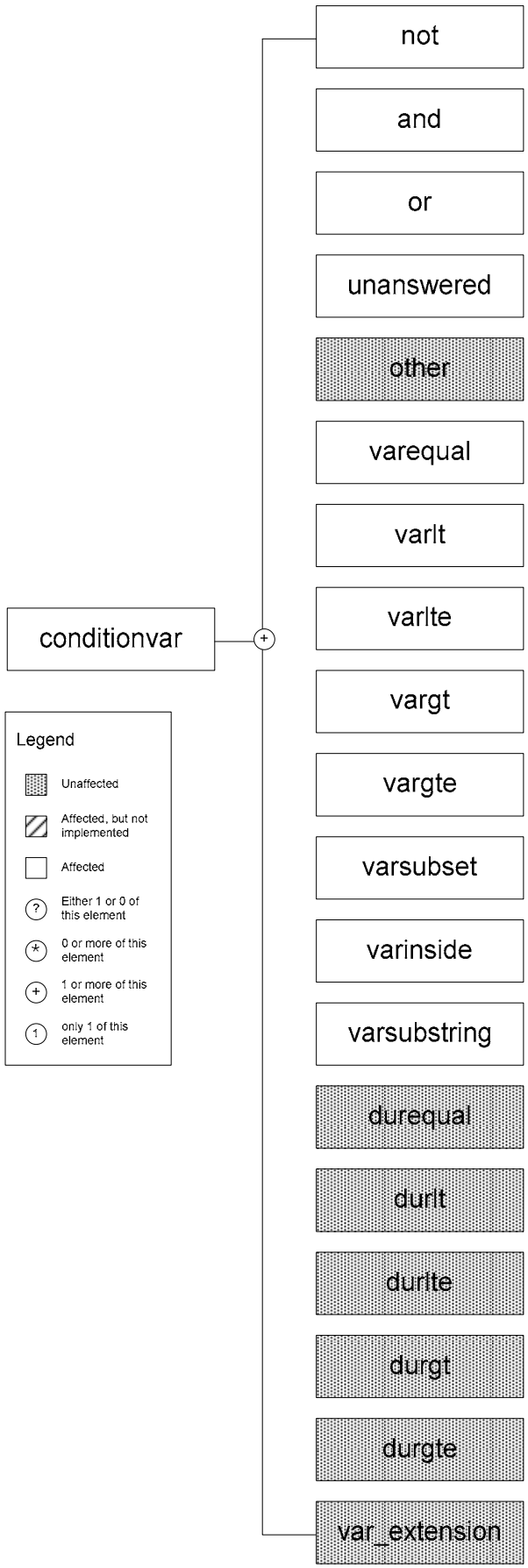


Figure 3.15 QTI conditionvar XML structure.

Summary

This section has provided an overview of the QTI specification, including the XML schema for the QTI item. The following key elements have been discussed in more detail, as they will be directly or indirectly impacted by the implementation of question parameterisation.

item	*presentation*
response_lid	*render_choice*
response_label	*material*
itemfeedback	*resprocessing*
respcondition	*conditionvar*

After covering the schema of the QTI Specification, some examples will be provided to illustrate their use.

3.3.17 Item XML Examples

The following provides some example QTI Items written in XML, and their rendered output to demonstrate the elements described in the sections 3.3.6 through 3.3.16. These examples have been reproduced from the IMS: ASI Best Practice & Implementation Guide (IMS 2002a). Figure 3.16 shows the XML schema for a typical multiple choice question.

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no"?>
<!DOCTYPE questestinterop SYSTEM "ims_qtiasiv1p2p1.dtd">

<!-- Author:   Colin Smythe                          -->

<!-- Date:     14th February, 2003                   -->

<!-- Example:  Example006                            -->

<!-- QTILite Example with five choices.              -->

<!-- Includes response processing for the correct    -->

<!-- answer.                                         -->
<questestinterop>
  <item title = "Standard Multiple Choice Item" ident = "IMS_V01_I_QTIExample006">
    <presentation label = "QTIExample006">
      <flow>
        <material>
          <mattext>Which one of the listed standards committees is responsible for
developing the token ring specification ?</mattext>
        </material>
        <response_lid ident = "MCb_01" rcardinality = "Single" rtiming = "No">
          <render_choice shuffle = "Yes">
            <flow_label>
              <response_label ident = "A">
                <flow_mat class = "list">
                  <material>
                    <mattext>IEEE 802.3</mattext>
                  </material>
                </flow_mat>
              </response_label>
            </flow_label>
            <flow_label>
              <response_label ident = "B">
                <flow_mat class = "list">
                  <material>
                    <mattext>IEEE 802.5</mattext>
                  </material>
                </flow_mat>
              </response_label>
            </flow_label>
            <flow_label>
              <response_label ident = "C">
                <flow_mat class = "list">
                  <material>
                    <mattext>IEEE 802.6</mattext>
                  </material>
                </flow_mat>
              </response_label>
            </flow_label>
            <flow_label>
              <response_label ident = "D">
                <flow_mat class = "list">
                  <material>
                    <mattext>IEEE 802.11</mattext>
                  </material>
                </flow_mat>
              </response_label>
            </flow_label>
            <flow_label>
              <response_label ident = "E" rshuffle = "No">
                <flow_mat class = "list">
                  <material>
                    <mattext>None of the above.</mattext>
                  </material>
                </flow_mat>
              </response_label>
            </flow_label>
          </render_choice>
        </response_lid>
      </flow>
    </presentation>
    <resprocessing>
      <outcomes>
        <decvar vartype = "Integer" defaultval = "0"/>
```

```
      </outcomes>
      <respcondition title = "Correct">
        <conditionvar>
          <varequal respident = "MCb_01">B</varequal>
        </conditionvar>
        <setvar action = "Set">1</setvar>
        <displayfeedback feedbacktype = "Response" linkrefid = "Correct"/>
      </respcondition>
    </resprocessing>
    <itemfeedback ident = "Correct" view = "Candidate">
      <flow_mat>
        <material>
          <mattext>Yes, you are right.</mattext>
        </material>
      </flow_mat>
    </itemfeedback>
  </item>
</questestinterop>
```

Figure 3.16 Multiple Choice Question Item Example (XML).

This XML will render the output shown in Figure 3.17.

Which one of the listed standards committees is responsible for developing the token ring specification?

- IEEE 802.3
- IEEE 802.5
- IEEE 802.6
- IEEE 802.11

Figure 3.17 Multiple Choice Question Item Example (Rendering).

Figure 3.18 shows the XML schema for a typical fill-in-the-blank question.

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no"?>
<!DOCTYPE questestinterop SYSTEM "ims_qtiasiv1p2p1.dtd">

<!-- Author:   Colin Smythe                                                -->

<!-- Date:     14th February, 2003                                         -->

<!-- Version 1.2 Compliant Example: BasicExample012b -->

<!-- Basic Example with response processing.               -->
<questestinterop>
  <qticomment>This is a standard fill-in-blank (text) example.</qticomment>
  <item title = "Standard FIB string Item" ident = "IMS_V01_I_fibs_ir_001">
    <presentation label = "BasicExample012b">
      <flow>
        <material>
          <mattext>Complete the sequence: </mattext>
        </material>
        <flow>
          <material>
            <mattext>Winter, Spring, Summer, </mattext>
          </material>
          <response_str ident = "FIB01" rcardinality = "Single" rtiming = "No">
            <render_fib fibtype = "String" prompt = "Dashline" maxchars = "6">
              <response_label ident = "A"/>
              <material>
                <mattext>.</mattext>
              </material>
            </render_fib>
          </response_str>
        </flow>
      </flow>
    </presentation>
    <resprocessing>
      <outcomes>
        <decvar varname = "FIBSCORE" vartype = "Integer" defaultval = "0"/>
      </outcomes>
      <respcondition>
        <qticomment>Scoring for the correct answer.</qticomment>
        <conditionvar>
          <varequal respident = "FIB01" case = "Yes">Autumn</varequal>
        </conditionvar>
        <setvar action = "Add" varname = "FIBSCORE">1</setvar>
        <displayfeedback feedbacktype = "Response" linkrefid = "Correct"/>
      </respcondition>
    </resprocessing>
    <itemfeedback ident = "Correct" view = "Candidate">
      <flow_mat>
        <material>
          <mattext>Yes, the season of Autumn.</mattext>
        </material>
      </flow_mat>
    </itemfeedback>
  </item>
</questestinterop>
```

Figure 3.18 Fill-in-the-blank Item Example (XML).

This XML will render the output shown in Figure 3.19.

Complete the sequence:

Winter, Spring, Summer, _____

Figure 3.19 Fill-in-the-blank Item Example (Rendering).

The QTI Specification provides a standard format for recording questions, tests, and the subsequent scoring results (IMS 2003a) using the XML language. The driving force of the QTI is to enable interoperability without constraining innovation. The QTI is one of many specifications developed by IMS. Other specifications include: Meta-data Specification, Content Packaging Specification, Learner Information Packaging Specification and Sequencing Specification (IMS 2002a). Great effort has gone into ensuring there is integration between each of the IMS specifications to reduce redundancy of effort and improve coupling between the related data sets (IMS 2002a).

The QTI Specification is comprised of multiple sub-specifications including: Results Reporting, QTI Lite, and ASI. Support is provided for multiple response and rendering types, ensuring items can be presented and completed in a variety of ways. Yet, despite the IMS QTI being quite a comprehensive specification, unfortunately it does not provide support for parameterised quiz questions.

4 Conceptualisation of Parameterised Quiz Question Characteristics

This chapter provides a more detailed discussion on parameterised quiz questions. In particular, it covers the following topics.

- A more detailed definition of parameterised questions.
- Identification of what type of content is well suited to parameterisation.
- An investigation into parameterised media and parameter types.
- Examples of useful parameterised questions.
- Discussion on candidate response processing.
- Discussion on candidate responses and results reporting.
- Investigation into parameterisation and question types.

4.1 What is quiz question parameterisation?

Parameterisation of quiz questions involves inserting variables into the question stem and defining the scope of possible values for the variables, which are randomly chosen for each instance of the question. For example, a typical question might be that shown in Figure 4.1.

What is 12 multiplied by 6?

Figure 4.1 Example of simple question that could be parameterised.

To parameterise the question in Figure 4.1, replace 12 and 6 with variables and define a range of possible values for each of those two variables. For example, the existing number 12 could be replaced with variable “a”, and 6 could be replaced with variable “b”. The variable “a” could be defined as a random integer chosen between 1 and 15, and “b” between 1 and 10. When instantiated, the question will then be randomly

generated based on the two variables (as shown in Figure 4.2). Each instance still assesses the ability to multiply, yet the objects of the multiplication are different.

What is 5 multiplied by 2?

Figure 4.2 Alternate instance of a parameterised question.

4.2 What question content is well suited to parameterisation?

Parameterisation is most effective in assessing objective tasks requiring some form of algorithmic processing that can be easily automated through computer software. Parameterised questions have traditionally been used in mathematics and science courses, where the question solutions can be calculated automatically and objectively through a mathematical expression (Pathak and Brusilovsky 2002). For example, the question illustrated in Figure 4.1 of Section 4.1 is well suited to parameterisation as the solution can be easily calculated via computer software through a simple mathematical expression. However, mathematics is not the only objective material that can be parameterised.

Other suitable content includes the parameterisation of computer programming evaluation questions, in which the computer can automatically execute the given code fragments to determine the solution to the question (Pathak and Brusilovsky 2002). An example of a rendered parameterised programming evaluation question is illustrated in Figure 4.3.

Given the following C code:

```
#include <stdio.h>

int main(void)
{
  int a ;
  int b ;
  int c = <param ident="varc">1..5</param> ;

  for(a=3;a<10;a++)
    for(b=<param ident="varb">5..7</param>;b<10;b+=2)
      c = c + b + a ;

  printf ("c=%i\n",c) ;
  exit(0) ;
}
```

The output will be:

c=<text box>

Complete the text box above showing the output from this code.

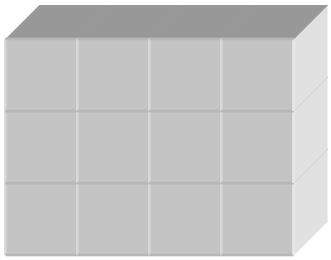
Figure 4.3 Example parameterised programming evaluation question.

The initial values of the variables "b" and "c" in the C program shown in Figure 4.3 are set to a random value in the range of 5 to 7, and 1 to 5 respectively. As a result of these random values, the output from the 2 nested "for" loops will be different for each instance of the question. However, the software can simply execute the C program using the instantiated values for "b" and "c" to calculate what the true output will be, for comparison with the candidate's solution.

4.3 Parameterised Media and Parameter Types

Parameterisation can be implemented in other ways besides textual information in a question stem. This has been demonstrated through the CAPA system as described in Section 2.1.3, where the parameterisation may provide variation of a diagram (image) to present to the candidate, and the candidate must answer questions relating to this diagram (Kashy, Thoennessen et al. 1997). Another example of parameterised images

is provided by a sample question with the commercial product “ExamView” (http://www.examview.com), where a parameterised image of a cube is instantiated with varying dimensions and the question asks for the volume as shown in Figure 4.4.



What is the volume of this box?

Figure 4.4 Example image based parameterised question.

The candidate is required to calculate the volume of the displayed cube by multiplying its width, height and depth. The parameterisation changes the image of the cube by using different dimensions for its size. The example shown in Figure 4.4 has a width of 4, height of 3 and a depth of 1. These dimensions could be any value. Of course, there is the potential to parameterise other media types.

Table 4.1 provides a breakdown of the media types that can be parameterised, along with the type of parameters (and examples) that could be used with each media type.

Table 4.1 Media and Parameters

Media	Type of parameters	

Text	Range
	Select from a sequence of possible values defined by the question, with option step increment
		`<integer range="1..20"/>`
		
		`<character range="a..z"/>`
		
		`<comment>The following would only select odd numbers between 1 and 99</comment>`
		`<integer range="1..100" step="2"/>`
		
		`<comment>The following would select real numbers with a resolution of 0.01</comment>`
		`<integer range="1..2" step="0.01"/>`

	Enumeration
	Specify an enumerated list of values defined by the question
		`<enum>`
		`  <enum option="apple">apple</enum>`
		`  <enum option="orange">oracle</enum>`
		`  <enum option="lemon">lemon</enum>`
		`</enum>`
		
		`<enum>`
		`  <enum option="1">1</enum>`
		`  <enum option="2">2</enum>`
		`  <enum option="3">3</enum>`
		`  <enum option="5">5</enum>`
		`  <enum option="7"><enum>`
		`</enum>`

	Formula Specify a formula potentially based on other instantiated parameters	`<integer formula="=param1 x param2"/>`

	Condition
	Specify conditions potentially based on other instantiated parameters
		`<condition set="param2">`
		`  <match test="param1>10">`
		`  <integer range="1..9"/>`
		`  </match>`
		`  <match test="param1<10">`
		`  <integer range="10..20"/>`
		`  </match>`
		`</condition>`

	External Program	`<text execute="/path/to/program.pl">options</text>`
Image	Enumeration Select from an enumerated list of different raster images, as defined by the question	`<image path="/path/to/image/files">Q42*.jpg</image>`

	Vector Image
	Generate a vector based image as per the following types: graph, geometry, trigonometry, or other objective images.
		`<vecimage type="graph" formula="y=2x+4"/>`
		
		`<vecimage type="trigonometry">`
		`  <triangle type="right angle" angle="60"/>`
		`</vecimage>`

	External Program Flexible to generate any type of raster or vector based image to display	`<extimage execute="/path/to/program.pl">options</extimage>`
Video	Enumeration Select from an enumerated list of different videos, as defined by the question	`<video path="/path/to/video/files">Q42*.mpeg</video>`
	External Program Flexible to generate any type of video, composed by whatever means	`<extvideo execute="/path/to/program.pl">options</extvideo>`
Audio	Enumeration Select from an enumerated list of different audio tracks, as defined by the question	`<audio path="/path/to/audio/files">Q42*.mp3</audio>`

	Text to speech
	Use of TTS technology to parameterise testing language in the areas of: spelling, multi-lingual, vocabulary, etc
		`<ttsaudio>`
		`  <tts>Hippopotami</tts>`
		`  <tts>Hippopotamus</tts>`
		`</ttsaudio>`
		
		`<ttsaudio>`
		`  <tts source="url">http://server.com/wordlist.txt</tts>`
		`</ttsaudio>`

	External Program	`<extaudio execute="/path/to/program.pl">options</extaudio>`

This is by no means an exhaustive list, but rather an overview of the types of parameterisation that can be implemented for certain media.

4.4 Examples of Useful Parameterised Questions

Figure 4.5 illustrates some practical rendered examples of parameterised questions with content ranging from mathematics, computer graphics, computer communication and programming.

Question 1

A person is facing along the vector (<param ident="a">0..9</param>, <param ident="b">0..9</param>, <param ident="c">0..9</param>). He wants to turn through the smallest angle to end up facing along the vector (<param ident="x">0..9</param>, <param ident="y">0..9</param>, <param ident="z">0..9</param>).
Which way should he turn?

- Left
- Right
- Neither

Question 2

If a world window has the coordinates (<param ident="Wl">0..500</param>, <param ident="Wr">Wl..1000</param>, <param ident="Wb">0..1000</param>, <param ident="Wt">Wb..500</param>) and a view port has the coordinates (<param ident="Vl">0..500</param>, <param ident="Vr">Vl..1000</param>, <param ident="Vb">0..500</param>, <param ident="Vt">Vb..1000</param>) the window-to-viewport mapping will be:

sx=<textbox>x + <textbox>

sy=<textbox>x + <textbox>

Question 3

What is the value of a, given the expression below?
a = <param ident="c">1..5</param> + <param ident="d">3*b</param> - <param ident="b">1..4</param> x <param ident="e">-5..0</param> - <param ident="d"/> ÷ <param ident="b"/>

- -16.3333333333333
- 23
- -7

Question 4

What is the value of a, given the expression below?
a = <param ident="c">1..5</param> + <param ident="d">3*b</param> - <param ident="b">1..4</param> x <param ident="e">-5..0</param> - <param ident="d"/> ÷ <param ident="b"/>

- -16.3333333333333
- 23
- -7
- None of the above

(note: This question is identical to that of question 3, with the exception that the option “None of the above” could also be parameterised such that it could be the correct answer)

Question 5

Complete the Regular Expression below, such that it will replace all occurrences of <param ident=”oldshell”>/bin/false,/bin/bash,/bin/tcsh</param> with <param ident=”newshell”>/bin/false,/bin/bash,/bin/tcsh</param> in the shell field of the /etc/passwd file?

```
sed –e ‘s/<textbox>/<textbox>/g’ /etc/passwd
```

Question 6

What is <param ident=”num1”>1..20</param> multiplied by <param ident=”num2”>1..15</param>? <textbox>

Question 7

What is the radius of a circle if its area is given as <param ident=”area”>10..50</param>? <textbox>

Question 8

Given the quadratic:

<param ident=”a”>1..8</param>x^2 + <param ident=”b”>1..20</param>x + <param ident=”c”>1..8</param> = 0

How many real solutions for x?

- 2
- 1

- No real solutions for x

If there is a solution, what are the value(s) of x (to at least 1 decimal place? <textbox>, <textbox>

Question 9

For the TCP/IP protocol, in which OSI relational model layer does the <param ident=”protocol”>TCP,UDP,IP</param> protocol belong?

- Layer 1
- Layer 2
- Layer 3
- Layer 4
- Layer 5

Question 10

What will be the output of the C program shown below?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int a ;
  int b ;
  int c = <param ident=”varc”>1..5</param> ;

  for(a=3;a<10;a++)
    for(b=<param ident=”varb”>5..7</param>;b<10;b+=2)
      c = c + b + a ;

  printf("c=%i",c) ;
  exit(0) ;
}

c=<textbox>
```

Complete the text box above showing the output from this program.

Figure 4.5 Practical examples of parameterised questions.

The mark-up shown in Figure 4.5 is designed to illustrate the parameterisation in the examples; using a comma to separate an enumerated list of different options, and ".." to denote a range of optional values.

4.5 Candidate Response Processing

Generating the dynamic question stem based on the parameters is only part of the equation. There is still a need to process the candidate's answer and ascertain if it is correct. In a traditional question, the correct answer to the question is pre-determined and already recorded in the question definition. However, with a parameterised question the answer is not static, and therefore must be calculated based on the instantiated values in the question stem. This calculated value can then be used to determine the correctness of the candidate's response.

4.5.1 Determining the correct answer

Although the correct answer could be mapped to particular parameter values in the question stem, this approach negates much of the benefits of automation from the parameterisation. If the answers are mapped directly to parameter values in the question, then each possible instance of the question has been manually evaluated by the question author and programmed into the response processing. This varies little from the traditional approach of preparing multiple static questions, each with different values from the parameters and the answer statically assigned to each.

Therefore, for parameterised questions to be more effective, the correct answer needs to be recorded as a function of the instantiated parameters in the question stem. This way, the answer to the question can be calculated for every possible outcome of the question parameters. Still, this function can be calculated in a variety of ways. It may be the result of a condition, a formula or a call to an external program depending on the content and complexity of the question (refer to text media in Table 5.1 of Section 4.3).

4.5.2 Calculating Distracters

Certain question types (see Section 4.7) require that distracters be presented to the candidate from which they must select the correct answer. A common question of this type is a multiple choice questions (MCQ). If a MCQ has been parameterised, not only must the correct answer be determined, but also the appropriate distracters.

There can be certain difficulties associated with the automatic generation of distracters for a question. The quality of a multiple choice question can be largely dependent on the quality of the distracters, in that they must be unambiguous and plausible (Lister 2000). Therefore, careful consideration of functions used to calculate the distracters is of the utmost importance. For example, ambiguity can be introduced where two or more automatically generated distracters (or the question key) intersect. There are a couple of different approaches to solve this problem.

- *Intersection detection* - A detection method could be implemented, such that a global condition is introduced to the question definition that identifies which instantiated values from the parameters must be unique. Such an approach is used by the commercial product "Exam View" (http://www.examview.com). If this condition is not met, meaning at least two parameter values intersect then all the parameters are re-calculated again using new random values. A threshold could be assigned limiting the number of re-calculations before giving up on finding appropriate values for the question. Refer to Figure 5.28 in Section 5.4.5 for an example question using a condition to detect parameter value intersection.
- *Intersection avoidance* - An avoidance method such as a test for each distracter that compares it with all previously calculated distracters. If a match is found, either the distracter is removed completely, or some other alternative

can be assigned, such as a simple offset (for mathematical type distracters). Removing the distracter completely would be the simplest approach, but then also reduces the number of distracters, which may present an advantage to some candidates. Refer to Figure 5.27 in Section 5.4.5 for an example question using conditions (in XML) to avoid parameter value intersection.

These approaches would help alleviate certain problems of ambiguity introduced by the automated generation of distracters.

To ensure distracters are plausible is a more difficult exercise to automate; given it can be quite a subjective judgement. This responsibility may rest largely on the author's design of the question and the algorithm/function used to generate the distracters.

4.6 Candidate Responses/Results Reporting

Typically it would be necessary to record the responses provided by the candidate along with their final results for an assessment piece for auditing purposes, particularly for summative assessment. Therefore a typical quiz system may store the response from the candidate, the feedback returned to the candidate based on their response, and perhaps a total for marks. This would be quite adequate where the question stem does not change, and is consistent for each student attempting the question.

This would not be sufficient once parameterisation of the questions was introduced, as the question stem would be different for each attempt. The context in which the candidate submitted their answer would not be recorded. Therefore, another

consideration for implementing parameterisation is to ensure that the question stem is recorded along with the candidate's response to the question, such that what the candidate was presented with, and their response can be reproduced at a later time (refer to chapter 7 for further details).

4.7 Question Types

The IMS QTI Specification supports a myriad of question types as discussed in Section 3.3.1. Parameterisation essentially could be implemented for any of these question types; however, at this stage focus of this research will be on selection rendering, and fill in the blank rendering types. In particular, focus will be on standard multiple choice, and the generic fill in the blank question types. This is because multiple choice is a very popular format due to its objectivity, and fill in the blank is popular for its flexibility (i.e. either ends of the scale). Furthermore, as this is the first parameterisation addendum to the QTI Specification, starting with unambiguous question types allows for more focus on integration and less on fine details for processing of the remaining question types.

Examples of how parameterisation can be implemented into both MCQ and FIB questions have been provided in Figure 4.5 of Section 4.4.

4.7.1 Multiple Choice Questions

Multiple choice questions require that distracters or incorrect alternatives be presented to the candidate, such that they must choose the correct answer or answers. To ensure the student is challenged in selecting the correct answer, the distracters must be

plausible, yet still incorrect. Keys or correct answers to MCQs could potentially be mapped directly to possible parameter values, such that:

- if parameter is equal to x, then the key is mapped to a,
- if parameter is equal to y, then the key is mapped to b,
- if parameter is equal to z, then the key could also be mapped to b.

An example of such an approach is shown in Figure 4.6 represented using XML.

4.7.2 Example of mapped distracters/key:

```
<mattext>For the TCP/IP protocol, in which OSI Relational Model Layer does the <param
label=”protocol”><enum><enum option=”tcp”><![cdata[TCP]]></enum><enum>
option=”udp”><![cdata[UDP]]></enum><enum option=”ip”><![cdata[IP]]></enum></param>
protocol belong?</mattext>

<option correct=”no”><mattext>Layer 1</mattext>
<option correct=”no”><mattext>Layer 2</mattext>
<option correct=”mapped”><map option=”ip”/><mattext>Layer 3</mattext>
<option correct=”mapped”><map option=”tcp”/><map option=”udp”/><mattext>Layer
4</mattext></option>
<option correct=”no”><mattext>Layer 5</mattext>
```

Figure 4.6 Example MCQ expressed using XML (Mapped Distracters/Key).

The XML in Figure 4.6 would render something like Figure 4.7.

For the TCP/IP protocol, in which OSI Relational Model Layer does the TCP protocol belong?

a) Layer 1
b) Layer 2
c) Layer 3
d) Layer 4
e) Layer 5

Figure 4.7 Example MCQ rendering (Mapped Distracters/Key).

The distracters Layer 1, 2, and 5 are completely static and will always be incorrect. The distracters/keys Layer 3 and 4 are mapped to the option that was instantiated when the question was generated.

With this approach, the problem is that there are only three possible outcomes for this question: TCP, UDP or IP. Therefore, from one parameterised question, there are only 3 different resulting questions. The time taken to develop and write this question would be at least equal to, and perhaps a little longer given the complexity of mapping the keys than say writing the same question 3 times where they would be directly mapped anyway.

The parameterised question could have a much higher number of possible outcomes, yet the process is still the same in that instances of the question are manually mapped with the answers. This approach is identical to that of a traditional quiz question (non-parameterised). The question is essentially structured such that the key is statically mapped to the particular details of the question.

The key benefit to parameterised questions is a more efficient way of creating a substantial number of questions, from a standard question template. This is leveraged through computer automation. Using statically mapped answers in a parameterised question is effectively using a manual process to design each instance of the question. Therefore, if distracters are necessary, they really must be automatically generated by the system to maximise benefit. An example of a question with automated distracters is shown in Figure 4.8.

4.7.3 Example of dynamically calculated distracters/key:

```
<mattext>What is <param label=”param1”><integer range=”1..20”/></param> multiplied by
<param label=”param2”><integer><condition><![cdata[if(param1 > 10) {param2=1..10)}
else {param2=10..20}]]></condition></integer></param>?</mattext>

<option><param
label=”ans1”><integer><formula><![cdata[ans1=param1+param2]]></param></option>
<option correct=”yes”><param
label=”ans2”><integer><formula><![cdata[ans2=param1*param2]]></param></option>
<option><param label=”ans3”><integer><formula><![cdata[ans3=param*param2-
1]]></param></option>
<option><![cdata[None of the above]]></option>
```

Figure 4.8 Example MCQ expressed using XML (Dynamic Distracters/Key).

The XML in Figure 4.8 would render something like Figure 4.9.

What is 18 multiplied by 8?

26
144
143
None of the above

Figure 4.9 Example MCQ rendering (Dynamic Distracters/Key).

The question stem has a simple param1, which is in the range of 1 to 20. Whether param1 is greater than or less than 10 determines the parameter param2. The simple idea is to constrain the question to only having one of the factors greater than 10, to restrict the difficulty of the question. For example, the question “What is 192354 multiplied by 32587?” might be beyond the level of the targeted student.

The distracters and the key use the same XML param element structure that is used to calculate appropriate distracters and to calculate the key for the given question. “26” is the addition of param1 and param2 (in case the candidate confuses the operator). “144” is the correct answer (`<option correct=”yes”>`), “143” is the correct answer off by 1, and finally there is “None of the above”. In this example, “None of the above” will never be correct. It may be the case that not all distracters need to be

automatically generated. There may be distracters that are unambiguous and plausible for all instances of the parameterisation.

As can be seen, this question has 10 x 20 = 200 possible outcomes. The distracters are automatically generated based on a mathematical formula, so it is not necessary for the author to map the outcome of the numbers with the possible distracters/keys. For the price (time) of one question, the author now has 200.

The fact that this question has substantially more outcomes than the previous example is not directly related to the fact that the distracters are automated, but rather the subject content being mathematics. However, the automated distracters have supported the implementation of 200 potential outcomes without the time overhead of pre-mapping the distracters/keys. There are of course question types that do not require distracters.

4.7.4 Fill in the Blank Questions

Other question types do not require the system to generate a selection from which the candidate must identify the correct solution, but rather express their own answer via a myriad of interfaces. A typical method is to use a Fill in the Blank type question. For this type of question the candidate is provided with a text box to type their response. The placement of this text box or boxes can improve the readability of the question, but the processing remains the same.

In the case of a string comparison (textual response) the system would calculate the correct string answer based on the instantiated value/s, and would then compare this

with the response from the candidate. Due to the fact that the candidate has fewer constraints by which to formulate their response using this rendering type, some form of text comparison of the candidate's response, with the calculated correct answer needs to be implemented. Depending on the objectivity of the question; this can range from a simple exacting comparison, to complex matching patterns or AI (Artificial Intelligence) algorithms. The comparison process can work in two different ways. The system could determine the correct response expected from the candidate, and then compare this with the candidate's response. Or alternately, where an algorithm, process or formula is involved and the solution from the candidate can vary to such an extent that a direct comparison is too problematic, the candidate's response could be used to calculate the outcome of the algorithm, process or formula. This calculated outcome can then be tested to see if it matches what is expected as part of the question definition (refer to Question 5 in Figure 4.5 for an example).

The first approach is general and could be applied to questions of any content type. The use of this method can be problematic however, particularly for string comparisons given the freedom of response from the candidate. Matching of free text can be quite difficult in such cases and may require the use of complex pattern matching or AI algorithms, which can potentially increase the complexity and still not be 100% accurate. An example of such a case is illustrated in Figure 4.10.

On a Red Hat Linux System, what command can be used to number each line of a given text file? <text box>

Figure 4.10 Example FIB question requiring complex string pattern matching.

The problem with this example is there is more than one command that can be considered a correct answer, being the “nl” command and the “cat” command. In fact, there could be other commands that also provide the same functionality, therefore the string processing needs to check for multiple occurrences. Furthermore, the candidate may answer the question with “nl”, or “nl command”, both strings of which would be correct. There are other methods that can be used to constrain what the candidate can respond with, for example limiting the number of characters that the candidate can use in their response. Unfortunately, this approach indirectly provides a hint to the candidate in identifying how many characters must be in the solution. Moreover, it is also less effective when there could be multiple correct answers of which each has varying length strings. This problem is not limited to text comparisons.

In the case of numeric comparisons, the system would calculate the answer to the question based on the instantiated values and problem, and would then compare this value with the response from the candidate. Typically, a match would indicate correctness. However, even for a mathematics problem, there can still be some variation in the correct response from the candidate, such as numeric precision and rounding when the solution is not an integer value for example. The typical approach to this problem is to allow a threshold of precision in which the candidate’s response is tested as being above a certain threshold, and below another threshold. This can be demonstrated by the following example.

If the correct answer to a question has been calculated as 3.141592654; rather than trying to determine if the candidate's response is equal to this value, instead perform the following comparison:

3.14 < candidate's response < 3.15

where 3.14 is the lower threshold and 3.15 is the upper threshold for what will be considered a correct solution.

Although there are some techniques that can aid in the difficulty of matching a correct FIB response, another method is to use the candidate's response in a test to see if it solves the question. This approach is well suited to procedural or outcome based problems. In the example question illustrated in Figure 4.10, the candidate is expected to input the name of a Red Hat Linux command that will number each line of an input file. By using this alternate approach, the question could be re-worded as shown in Figure 4.11.

Complete the shell command below such that the output will be the /etc/fstab file's contents with each line numbered starting from 1

bash$ <text box> /etc/fstab

Figure 4.11 Example FIB question using the response from candidate to test that it is correct.

The candidate would fill in the text box completing the bash shell command such that the output will be the contents of the /etc/fstab file with each line numbered. The system would take the response from the candidate, and then pass it to the bash shell program with "/etc/fstab" as the last argument and execute the command, storing away the resulting output. The system would then execute via the bash shell again, a known correct solution (nl command for example) and then compare this output with

that of the candidate's solution. If a match is found between the resulting outputs, then the candidate's solution must be correct. This approach will also address the problem of there being other correct solutions, aside from those known by the question author. The execution of these commands would be handled by the program type parameter functionality, as described in Table 4.1.

Furthermore, this approach could also be applied to a programming question in which the candidate must complete a template program such that it performs the tasks set by the question stem. A comparison of output from the candidate's solution and the model solution would determine if the candidate was correct.

Quiz question parameterisation involves the insertion of variables into the question stem and randomly selecting values for the variables, based on the values defined for each variable. Questions that cover content, which can be objectively assessed via some form of algorithmic processing, are best suited to parameterisation quiz questions. This is traditionally mathematics and sciences. Parameterisation does not necessarily need to be limited to textual media. Other media types that could be parameterised include: images, video, and audio. Textual parameters can be classified based on the method in which they are calculated. Such classifications include: range parameters, enumeration parameters, formula parameters and program parameters. Parameterisation could be implemented into a wide range of question types. This research has focused in particular on MCQ and FIB question types. In the case of MCQ, distracters can be either statically provided into the question, or alternatively dynamically generated based on a formula or computer algorithm. Implementation of parameterisation support into a quiz system must also include the

ability to process the response/s from the candidate, such that these responses can be compared with parameter values.

The following chapter will discuss the development of an addendum to the IMS QTI Specification to include support for parameterised questions.

5 Parameterised Question & Test Interoperability Addendum

This chapter will be focused on a parameterised extension for the IMS QTI Specification. Parameterisation has been implemented as an addendum to the existing QTI Specification, and has been named The Question & Test Interoperability Parameterised Addendum (QTIPA). The following information is provided within this chapter.

- The integration goals of the specification addendum (Section 5.1).
- The framework for the implementation of parameterisation into the IMS QTI Specification (Section 5.2).
- An introductory overview of the specification addendum (Section 5.3).
- As per the parameterisation implementation framework, the integration of new elements, the re-use of existing QTI elements, and discussion of alternate approaches to support parameterisation is examined in detail:
    - Parameter Declaration (Section 5.4),
    - Parameter Presentation (Section 5.5),
    - Response Processing (Section 5.6).
- example XML excerpts demonstrating the new QTIPA Specification (presented throughout the chapter),
- global design considerations in development of the QTIPA Specification (Section 5.7).

5.1 Parameterisation integration goals

There are many challenges involved in enhancing an existing industry specification. In the context of this research, one such challenge is ensuring seamless integration of

parameterisation support into the existing QTI Specification such that the following goals are achieved:

- backward compatibility,
- re-use of existing QTI Elements,
- maintain consistency with existing naming conventions.

It is important that changes made to the existing QTI to provide support for parameterisation, do not break compatibility with existing QTI XML documents that have been written and validated against the previous version (see Appendix B for further information on XML validation). This will be less disruptive to implementers in migrating to the new version of the specification, as their existing assessment material investments will be compatible with the new version. Another consideration is the re-use of existing QTI elements where possible to support parameterisation. This has minimised the number of changes required to support parameterisation by eliminating duplication, which as a result, has simplified the overall design of the QTIPA. However, backward compatibility takes precedence over re-use efficiency. For example, if re-using an existing element breaks compatibility with previously written QTI XML documents, then a new element will be introduced instead. The following rules have been followed, to ensure that changes to the IMS QTI Specification do not break compatibility with existing QTI documents:

1. Existing elements and their attributes have not been removed,
2. Existing element names and their existing attributes remain unchanged,
3. Existing elements are re-used if their logical purpose within the existing specification exactly matches their re-used purpose, and provided no changes are required to the element that would break their use in its original purpose,

4. Where elements cannot be re-used to meet criteria 3, new elements are introduced instead.

By following this structure, existing elements and attribute do not change with the exception that they may allow as children, new elements. In other words, only additions to the specification have been made, rather than changing what was already provided. Finally, consideration of existing naming conventions is also important, such that there is consistency between the new and old element and attribute names. This provides a more readable and seamless specification.

Now that the integration goals of the QTIPA have been discussed, the following section introduces categories, which describe the major top level components of a QTI item in which parameterisation support has been introduced.

5.2 Parameterisation Implementation Framework

Implementation of parameters into a QTI item can be placed in 3 fundamental categories as shown in Table 5.1.

Table 5.1 Categories for parameters within the ASI XML Binding

Category	Description
Declaration	The parameters themselves need to be declared and the values defined as per the parameterisation XML syntax
Presentation	The instantiated parameter variables must be able to be presented within the existing presentation structures of the specification
Response Processing	The instantiated parameter variables must be able to be used within the response processing structures of the existing specification

Within each of these 3 categories, there are 4 common problems that have been researched and solved. They are:

- where to place new structures supporting parameterisation,
- how the new structures are implemented,
- what naming convention should be used,
- how the structures relate to one another through the 3 categories as per Table 5.1.

Each of these common problems will be discussed in greater detail, starting from Section 5.4. The next section provides an overview of the QTIPA Specification.

5.3 QTIPA Implementation Overview

The QTIPA Specification structure is represented in Figure 5.1, which shows all the new parameterisation elements (unshaded), and how they integrate into the existing QTI. The structures are grouped into the categories identified in Table 5.1 from Section 5.2. Elements labelled with an ellipsis "..." represent the remainder of a given element structure which has been omitted from the illustration for brevity. It can be viewed by referring to the IMS Question & Test Interoperability Specification: ASI XML Binding Specification Document (IMS 2002b).

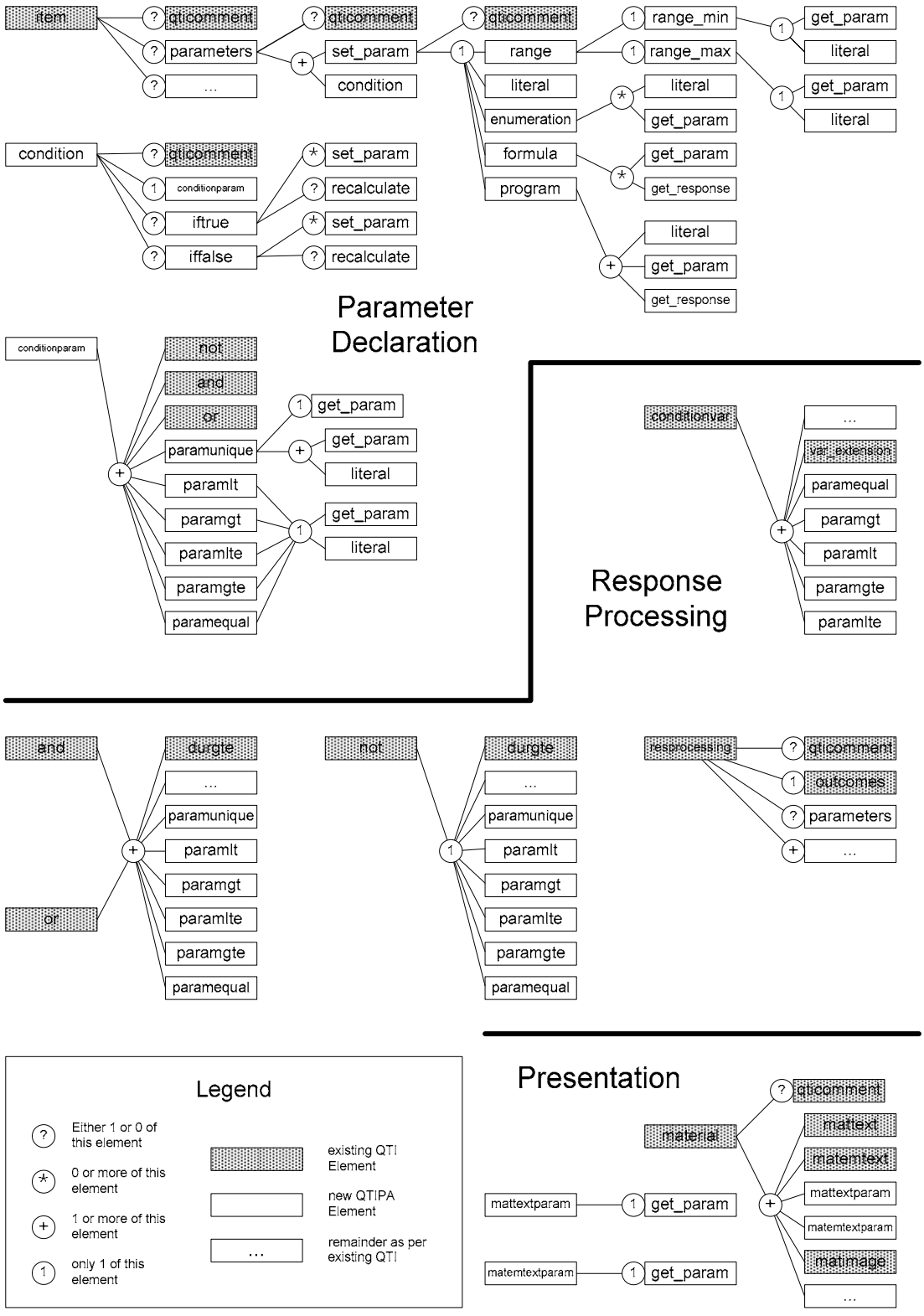


Figure 5.1 QTIPA XML Structure.

The following elements have been implemented into the parameter declaration component of the specification.

- A *parameters* element has been included as a child of the *item* element, and is the main container element for parameter declaration.
- Within the *parameters* element structure is an optional *qticomment* element for providing a comment on the parameters for the item, and one or more *set_param* or *condition* elements.
- The *set_param* element is responsible for assigning a value to a parameter variable, which is identified by an attribute *ident*.
- The type of parameter is determined by the child element of *set_param*, and must be one of *range*, *enumeration*, *formula*, *program* or *literal*.
- The *literal* element allows a constant literal value to be provided, and is used throughout the specification. Each of the remaining child elements corresponds to those identified in the text media row of Table 4.1 in Section 4.3.
- The *condition* element provides for a complex decision structure, which is also discussed in Table 4.1. It consists of an optional *qticomment* element, followed by one *conditionparam* element.
- The *conditionparam* element structure describes the conditional test to be performed on the parameters contained within, and is synonymous with the existing QTI *conditionvar* element. More discussion on the conditionparam structure follows. Trailing *conditionparam* is the *iftrue* and *iffalse* elements.
- The *iftrue* and *iffalse* elements may contain zero or more *set_param* elements, which allow the assigning of values to parameter variables, based on the true

or false outcome of the *conditionparam* test. Furthermore, each of *iftrue* and *iffalse* may also contain one optional *recalculate* element.

- The *recalculate* element forces the system to recalculate all parameter values over again and is commonly used in the case where parameter value intersection or overlap has occurred (see Section 4.5.2 for an explanation of overlapping parameter values).
- Referring back to the *conditionparam* element, it may take a range of conditional child elements such as: *paramunique*, *paramlt*, *paramgt*, *paramlte*, *paramgte* and *paramequal*. Each of these elements are described in Table 5.5 in Section 5.4.5 which relates their purpose with the existing var type elements already provided by the QTI for response processing (where such a relation exists).

The *material* element as part of the presentation component is the core for describing material to be presented to a range of participants, including the candidate. As such, parameter values must be supported within this structure, so they can be presented interspersed with the existing material. The following elements form part of the parameter presentation component. Figure 5.1 shows the *material* element structure, which illustrates the integration of two new elements, *mattextparam* and *matemtextparam*. These elements allow the presentation of parameter values, and emphasised rendering of parameter values (for example, bold typeface) respectively. They must take a single child element, being *get_param*, which is responsible for retrieving the value of a parameter identified by its *ident* attribute. The final component of the parameterisation implementation is the response processing.

The response processing component of the QTI Specification must allow the comparison of parameter values with responses from the candidate. To facilitate this, the QTI *conditionvar* element has been modified to incorporate the re-use of the elements: *paramequal*, *paramgt*, *paramlt*, *paramgte*, and *paramlte* from the *conditionparam* element structure. How they are used is described in Table 5.16 of Section 5.6.1. Furthermore, the *and*, *or*, and *not* elements have been modified to also include the same child elements, such that they too can be used in complex decision structures. Figure 4.11 of Section 4.7.4 illustrates an example FIB question where the response provided by the candidate can be used in a calculation or algorithm to determine if it is in fact correct. To facilitate support for this functionality, it is necessary to be able to declare and process parameters within the response processing component of the specification such that a *program* or *formula* element can be called passing the candidate's response. Therefore, an optional *parameters* element is available within the *resprocessing* element, as shown in Figure 5.1. For further explanation of this concept, refer to Section 5.6.

In the following sections, each of the categories as per Table 5.1 in Section 5.2 will be examined in greater detail. Included will be discussion on the advantages and disadvantages of alternate approaches that were investigated for implementation of parameterisation. The parameters declaration section, which examines all the elements as per the declaration component of the addendum is discussed in the next section.

5.4 Parameters Declaration

The parameters for a particular item must be declared in some way, such that their values are instantiated prior to presentation of the question to the candidate. This section will provide further discussion on the implementation of parameter declaration, starting with its placement within the existing scheme.

The parameter elements have been implemented into the specification using a similar model to that employed by the item outcomes scoring variables. The scoring variables are values that are used for calculating a mark for the item. A parameter is not unlike a scoring variable, but is merely a variable within the question stem itself. Once a scoring variable is declared, it is the application's responsibility to keep track of its value, and it must use the respcondition element structure as instruction on how to manipulate the scoring variables' values. Likewise, with the parameter variables, these too must be maintained by the application and be manipulated and displayed by the *resprocessing* and *presentation* elements.

The scoring variables are defined within the outcomes element which is a child element of *resprocessing* (refer to Figure 3.13 in Section 3.3.14). There should be only one outcomes element, and it should be before the *respcondition* or *itemproc_extension* elements. This way the scoring variables are defined and available for manipulation within the respcondition element structure so that the candidate's mark for the item may be updated as per the response conditions for the item. The scoring variables are not manipulated by any ancestral elements of *resprocessing*, therefore its scope is localised within *resprocessing*. It is a sibling element of *respcondition*. Likewise, the parameter declaration element structure has been implemented as a sibling to *presentation* and *resprocessing*. As the parameter

declarations are not manipulated by any ancestral elements of *item*, they are a child to the *item* element. Further discussion on the re-use of the parameters structure within response processing can be found in Section 5.6.2.

Figure 5.2 shows the structure of the new *parameters* element for the declaration and instantiation of parameters, and its placement within the *item* element.

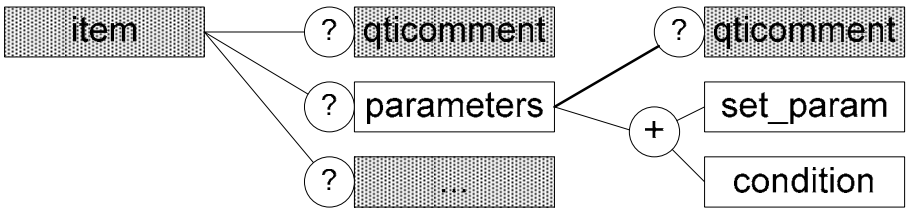


Figure 5.2 QTIPA parameters XML structure and placement.

If a *parameters* element is provided at the top level of the *item* element, there can be only one. Otherwise, if the question item does not include parameterisation, then there need not be a *parameters* element at all. If it is provided it should be immediately before a *qticomment* element (if included), otherwise the very first child element to *item*. This will ensure that their values are defined at the outset, before their values are referenced by any other elements within the item structure. However, this will only support manipulation of parameter values prior to presentation to the candidate. Therefore the *parameters* element has also be included within the resprocessing structure, which would only be processed when evaluating the candidate's response (i.e. after the candidate has submitted answers, rather than before presentation of the question to the candidate). By allowing a second *parameters* element to be included within the *resprocessing* element, parameters can be instantiated before processing the candidate's response, thus allowing the candidate's response to be used in calculating the correct answer (see Section 5.6 for further

information and examples). Figure 5.3 shows the placement of the *parameters* element also within the *resprocessing* element.

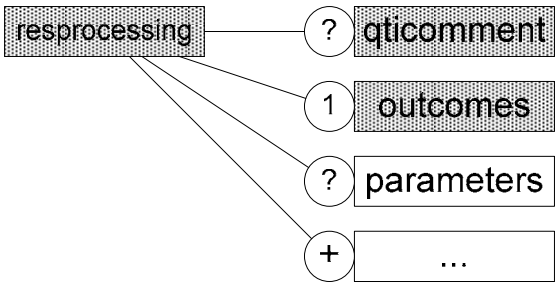


Figure 5.3 Placement of parameters element within resprocessing.

Consistent with the placement of *parameters* as a child of *item*, its placement within *resprocessing* is limited to zero or one occurrence, and is placed immediately after the outcomes element structure. Now the structure of the child elements contained within the *parameters* element will be examined.

5.4.1 set_param and get_param elements

The key functionality of the *parameters* element structure is to define the parameters, and generate their associated values for use in the question. The specification needs to be able to assign a value to a parameter variable, and also reference those stored values. Therefore, two elements have been implemented into the *parameters* element structure to support this requirement. They are represented in Figure 5.4. The *set_param* element is responsible for storing a value associated with a particular parameter. The parameter is also uniquely identified by its attribute *ident*. Conversely, the *get_param* element is responsible for accessing an already stored value associated with a particular parameter, also through the attribute *ident*. The *get_param* element does not contain any data. The use of underscore characters in

element names is prevalent through the existing specification, and has therefore been used within the QTIPA.

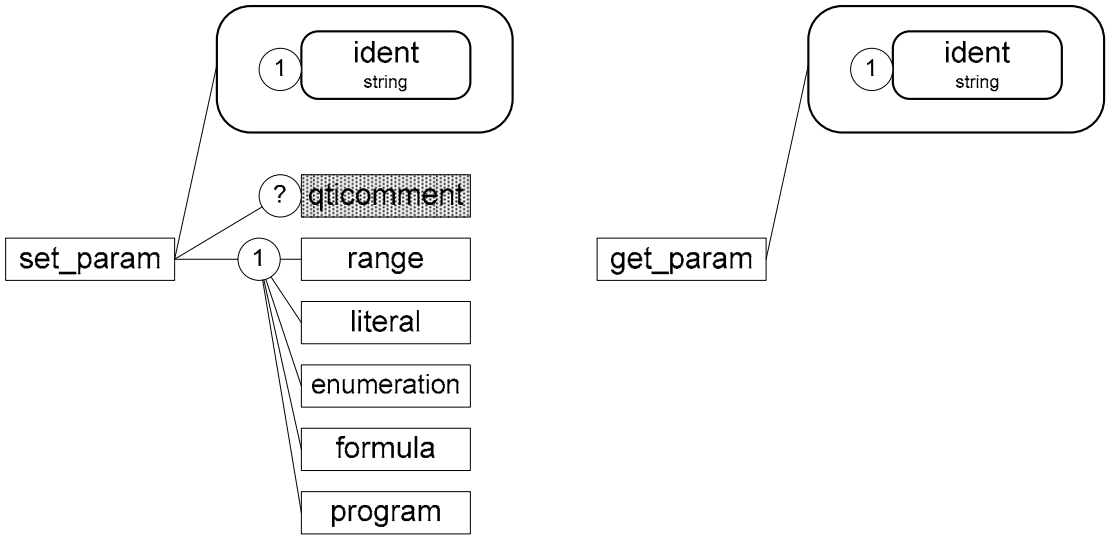


Figure 5.4 set_param and get_param elements structures.

The sub-elements of *set_param* specify the type of the parameter (with the exception of conditions – see Section 5.4.5). This is represented by the method the parameter is calculated. The possible types for text media are represented in Table 4.1 in Section 4.3. Each of these types and corresponding elements are described further in the following sections, including alternate methods in which they can be implemented into the specification. Example parameterised questions from Figure 4.5 in Section 4.4 will be used throughout to help illustrate how the *parameters* element structure can be implemented into the specification.

5.4.2 range element

A range based parameter is defined as one which is calculated based on the random selection of one value from an ordinal series, constrained with lower and upper boundaries. A simple example of range parameters in a question is shown in Figure 5.5.

```
What is <param ident="num1">1..20</param> multiplied by <param
ident="num2">1..15</param>?
```

Figure 5.5 Example Range Parameter Type.

The parameter num1 can be a randomly selected number between 1 and 20, and num2 can be a randomly selected number between 1 and 15. Of course, the ordinal values could also be ASCII (American Standard Code for Information Interchange) characters, rather than just limited to numbers. Figure 5.6 represents the structure of the *range* element.

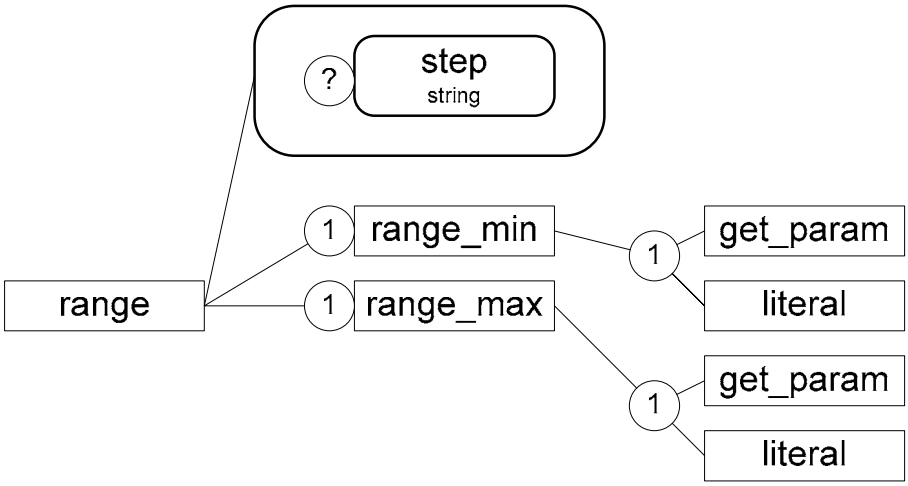


Figure 5.6 range element structure.

The QTIPA XML representing the question in Figure 5.5 is illustrated in Figure 5.7.

```
<parameters>
  <set_param ident="num1">
    <range>
      <range_min><literal>1</literal></range_min>
      <range_max><literal>20</literal></range_max>
    </range>
  </set_param>

  <set_param ident="num2">
    <range>
      <range_min><literal>1</literal></range_min>
      <range_max><literal>15</literal></range_max>
    </range>
  </set_param>
</parameters>
```

Figure 5.7 Example XML for range element.

There is no ambiguity as to which value is the upper boundary and which is the lower, as separate sub-elements are used to represent each value being *range_max* and *range_min* respectively. The DTD requires that one and only one of *range_min* and *range_max* be sub-elements to *range*. The use of the *literal* element is also shown in Figure 5.7. The purpose of the *literal* element is to ensure that all data within the parameters structure is tagged in some way, therefore preventing the need for mixed content elements. For example, it may be necessary to have parameterised boundaries on the range of a parameter, such that the bounds of a parameter are based on the value of another parameter. An XML representation is shown Figure 5.8.

```
<parameters>
    <set_param ident="Wl">
      <range step="1">
        <range_min>
          <literal>0</literal>
        </range_min>
        <range_max>
          <literal>500</literal>
        </range_max>
      </range>
    </set_param>
    <set_param ident="Wr">
      <range step="1">
        <range_min>
          <get_param ident="Wl"/>
        </range_min>
        <range_max>
          <literal>1000</literal>
        </range_max>
      </range>
    </set_param>
    <set_param ident="Wb">
      <range step="1">
        <range_min>
          <literal>0</literal>
        </range_min>
        <range_max>
          <literal>1000</literal>
        </range_max>
      </range>
    </set_param>
    <set_param ident="Wt">
      <range step="1">
        <range_min>
          <get_param ident="Wb"/>
        </range_min>
        <range_max>
          <literal>500</literal>
        </range_max>
      </range>
    </set_param>
```

Figure 5.8 Example of parameterised boundaries on range parameter.

The lower boundary for the Wr parameter is a parameter value itself being the Wl parameter, which essentially ensures that the value of Wr will always be greater than or equal to Wl. This illustrates that by introducing a specific element solely for the purpose of storing PCDATA, this easily allows the mixing of literal values (such as 0 or 500) and of represented values (being that stored by the *get_param* element). So far, the boundaries of the range have been discussed, but not the resolution of the ordinal series.

In the example represented in Figure 5.5, the number selection is assumed to have a step of 1, meaning that the ordinal dataset is in increments of one. It may be necessary to have a range selection with a fractional step. This can be illustrated in the example question shown in Figure 5.9.

```
Represent the decimal number <param ident=”fraction” step=”0.05”>0..1</param> as a
fraction.
```

Figure 5.9 Example Range Parameter Type with Step.

The question will randomly select a number between zero and one where the step increments are at 0.05. Possible values for this parameter are: 0, 0.05, 0.1, 0.15 up to and including 1. Therefore, given the value of 0.015 for the parameter fraction, the correct solution to the question is 15/100. This example demonstrates one possible use of fractional step values, although it cannot accurately represent all possible fractions, for example 0.33333 is not exactly equal to 1/3. Alternative uses for the range element include a step value of two, which could be used to allow every other number to be selectable (e.g. 1, 3, 5 ...).

Figure 5.10 shows the QTIPA XML for the *parameters* element structure, representing the question from Figure 5.9 in which a step of 0.05 is specified through the attribute step. If this attribute is not provided, then a step of one is assumed.

```
<parameters>
  <set_param ident="fraction">
    <range step="0.05">
      <range_min><literal>0</literal></range_min>
      <range_max><literal>1</literal></range_max>
    </range>
  </set_param>
</parameters>
```

Figure 5.10 Example XML for range element (including step).

An alternate approach to identifying the boundaries could see them implemented as attributes to the *range* element, similarly to the *step* attribute. There is much conjecture on how to decide whether data is stored as an attribute or element data (Johnson 2001). A common belief is meta-data should be stored as an attribute and data as element data (Kimber 1997; Megginson 1997). Consider that the step value is meta-data (describing how the ordinal range should be incremented), and the boundaries are the actual data defining the element itself, then the lower and upper boundaries for the range should be implemented as element data, as has been done.

The *range* element describes a step increment, a lower boundary for the range, and an upper boundary for the range, which can then be assigned to a parameter variable through its parent *set_param* element. Another parameter type is represented by the *enumeration* element and is discussed in the next section.

5.4.3 enumeration element

An enumeration parameter is one in which a randomly selected value is chosen from an enumerated list of possible values. Unlike the range parameter type, the

enumeration type can contain an arbitrary list of possible values that need not be ordinal. An example of a parameterised question using enumerated parameters is shown in Figure 5.11.

```
Complete the Regular Expression below, such that it will replace all occurrences of
<param ident="oldshell">/bin/bash,/bin/tcsh,/bin/false</param> with <param
ident="newshell">/bin/tcsh,/bin/false,/bin/bash</param> in the shell field of the
/etc/passwd file?

sed -e 's/<text box>/<text box>/g' /etc/passwd
```

Figure 5.11 Example Enumeration Parameter Type.

The parameter oldshell can contain a randomly chosen value from the enumerated list: /bin/bash, /bin/tcsh, /bin/false. Then the parameter newshell will be assigned a randomly chosen value from the same enumerated values (which of course could be any arbitrary list of values). A possible instantiation of this question is shown in Figure 5.12.

```
Complete the Regular Expression below, such that it will replace all occurrences of
/bin/bash with /bin/false in the shell field of the /etc/passwd file?

sed -e 's/<text box>/<text box>/g' /etc/passwd
```

Figure 5.12 Example Instantiation of Enumeration Parameter Type Question.

The enumeration parameter type simply needs to describe a list of unrelated potential values for a parameter. To support this functionality, the *enumeration* element has been implemented and is illustrated in Figure 5.13.

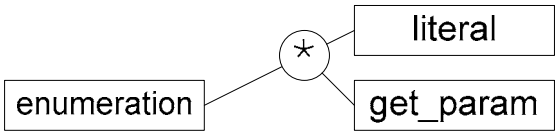


Figure 5.13 enumeration element structure.

The *enumeration* element structure for the question described in Figure 5.11 is represented as QTIPA XML in Figure 5.14.

```
<parameters>
  <set_param ident="oldshell">
    <enumeration>
      <literal>/bin/bash</literal>
      <literal>/bin/tcsh</literal>
      <literal>/bin/false</literal>
    </enumeration>
  </set_param>

  <set_param ident="newshell">
    <enumeration>
      <literal>/bin/tcsh</literal>
      <literal>/bin/false</literal>
      <literal>/bin/bash</literal>
    </enumeration>
  </set_param>
</parameters>
```

Figure 5.14 Example XML for enumeration element.

As shown with the *range* element structure (see Section 5.4.2), the *literal* element is used to represent literal values as part of the enumeration dataset. Parameter values can also form part of the enumeration dataset by including *get_param* elements therein. Now that the *enumeration* element has been described, alternate methods for implementation will be discussed.

With both the range and enumeration parameter types, they hold an array of possible values that can be assigned to a parameter. So far, it has been assumed that they will randomly select one possible value to then be set by the *set_param* element. Alternatively, both the *range* and *enumeration* elements could be embedded within another element, *random*, which is responsible for randomly selecting a value from the element data of *range* or *enumeration*. Figure 5.15 shows an example of *range* and *enumeration* elements embedded within a *random* element.

```
<parameters>
  <set_param ident="num1">
    <random>
      <range step="0.5">
        <range_min><literal>0.5</literal></range_min>
        <range_max><literal>5</literal></range_max>
      </range>
    </random>
  </set_param>

  <set_param ident="op1">
    <random>
      <enumeration>
        <literal>+</literal>
        <literal>-</literal>
        <literal>*</literal>
      </enumeration>
    </random>
  </set_param>

  <set_param ident="num2">
    <random>
      <range step="0.5">
        <range_min><literal>0.5</literal></range_min>
        <range_max><literal>5</literal></range_max>
      </range>
    </random>
  </set_param>
</parameters>
```

Figure 5.15 Example of using random element.

The dataset for the range defined for parameters num1 and num2 are the same. Another approach could be to declare the range once given its own *ident*, and then reference the range in the same way as the get and set params elements. The only benefit to this approach is brevity of the resulting XML where it was necessary to select multiple values randomly for parameters from the same dataset (either *range* or *enumeration*). Another example is shown in Figure 5.11 where the question has two parameters randomly selecting values from the same enumerated dataset. Having to track range and enumeration lists in the same way as the actual parameters themselves is merely adding complexity to the specification. Brevity of resulting XML output is of little consequence; therefore, no further consideration of the random element was made.

Both the *range* and *enumeration* elements introduce randomisation into the QTI item. It is also necessary to be able to perform calculations on the values of the parameter themselves which is supported through the *formula* element.

5.4.4 formula element

So far, both the range and enumeration parameter types have dealt with the process of randomly selecting a value from a defined dataset. A formula parameter provides the ability to calculate the value of a parameter using a mathematical expression. Of course, some of the values within the expression may be other parameter values. The formula parameter may often be used in calculating the correct (or incorrect – i.e. distracters) answer to a parameterised question, where the parameter values instantiated in the question form part of the formula parameter expression. Taking the example parameterised question in Figure 5.5 from Section 5.4.2 where two numbers which are parameters are multiplied together, the candidate must determine the result of that multiplication. A formula parameter would then be introduced to actually calculate the multiplication of the two instantiated parameter values which can then be used to compare with the candidate's response to evaluate whether they are correct (see Section 4.5.1). To support this functionality, the *formula* element has been included into the QTIPA as shown in Figure 5.16.

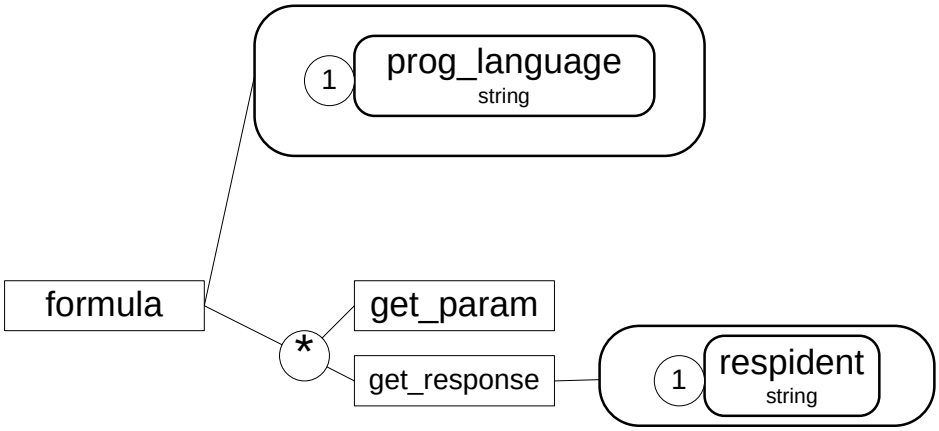


Figure 5.16 formula element structure.

The *formula* element has been implemented using programming language syntax, rather than structured XML. The element has been designed to use mixed content so that the *get_param* elements can be included into the expression. For example, what if a quiz question asked for the radius of a circle given its area? The formula could be written as an expression in C as shown in Figure 5.17.

```
<parameters>
  <set_param ident=”area”>
    <range>
      <range_min><literal>10</literal></range_min>
      <range_max><literal>50</literal></range_max>
    </range>
  </set_param>

  <set_param ident=”answer”>
    <formula prog_language=”c”>
      =sqrt(<get_param ident=”area”/> / 3.14)
    </formula>
  </set_param>
</parameters>
```

Figure 5.17 Example of programming language syntax for formula element.

When the application parses the XML, the element *get_param* will be replaced with the parameter value identified by “area” into the expression. Then the program would simply execute the complete C expression. Furthermore, the *get_response* element is also available as part of the mixed content. The *get_response* element represents a response provided by the candidate in answering the item. This means a response

from the candidate can be included in a mathematical expression as discussed in Section 5.4 where the *parameters* element is located within the *resprocessing* element structure. The response is identified by the *get_response respident* attribute, as is used by the existing QTI *conditionvar* child elements (such as *varequal*) to reference a candidate response.

The formula parameter need not only be for calculating the correct answer to the question. Figure 5.18 shows an example question where a formula parameter is used to calculate an actual parameter for display within the question specification itself.

```
2 to the power of <text box> is equal to the number <param>.
```

Figure 5.18 Example Question using Formula Parameter in stem.

The XML representing this question is illustrated in Figure 5.19.

```
<parameters>
  <set_param ident="answer">
    <range>
      <range_min><literal>1</literal></range_min>
      <range_max><literal>8</literal></range_max>
    </range>
  </set_param>

  <set_param ident="number">
    <formula prog_language="c">
      =pow(2,<get_param ident="answer"/>)
    </formula>
  </set_param>
</parameters>
```

Figure 5.19 Example XML representing Formula Parameter in stem.

The pow C function from math.h will take the base value of 2, and raise it to the power of the value represented by the parameter answer.

Table 5.2 discusses the advantages and disadvantages of using programming syntax for implementation of the *formula* element.

Table 5.2 Advantages & Disadvantages of Programming Syntax for Formula Element

Use of programming syntax for formula element	
Advantages:	Quicker and easier to develop and implement into the specification
	Less parsing of XML structures required
	Easier to interpret the written XML than the other methods
Disadvantages:	Must use a platform-independent language such as Perl or C for the QTI Item to be interoperable
	This approach would not make it possible to perform XML validation of the formula syntax
	Security issues arise due to the fact that code written by the question author will be executed within the quiz system, therefore checks will be necessary to ensure nothing malicious is included.

There are alternate options in implementing the *formula* element. One such alternative could be implementation through the use of the MathML specification (http://www.w3.org/Math/). The MathML specification supports the encoding of mathematical expressions such that they can be rendered, evaluated, and edited (Miner and Schaeffer 2001). Table 5.3 shows the advantages and disadvantages of using MathML for representing the formula parameter type element.

Table 5.3 Advantages & Disadvantages of MathML for the Formula Element

Use of MathML for formula element	
Advantages:	MathML is a very complete specification, which could potentially support a great variety of formulas.
	MathML is already a widely accepted industry specification
Disadvantages:	The complexity of the MathML specification would make its addition to the QTI very bulky and a heavy burden on vendors looking for conformance

Another alternate approach includes the *formula* element being custom designed with a limited subset of mathematical operators, providing the bare essentials. Operators and mathematical functions could include:

Addition	Subtraction	Multiplication
Division	Modulus	Logarithms (base 10 and base e)
Square Root	Tan	Cos
Sin	Absolute	

The advantages and disadvantages to developing a custom designed subset of mathematical operators are described in Table 5.4.

Table 5.4 Advantages & Disadvantages of Custom Designed Mathematical Operators for Formula Element

Use of custom designed mathematical operators for formula element	
Advantages:	Simpler to design and implement
	Significantly less bulk, reducing the complexity for vendors to implement
Disadvantages:	May lack some flexibility for authors trying to write complex expressions
	This approach would be "re-inventing the wheel" given that a specification exists (MathML) for describing mathematical expressions

For example, to represent the mathematical expression:

$(8 \times 12 - 7) - 22 \times \text{Log}_{10}\ 6$

using XML, it could be expressed as shown in Figure 5.20 using the custom subset approach.

```
<set_param ident="complexformula">
  <formula>
    <subtract>
      <divide>
        <subtract>
          <multiply><num>8</num><num>12</num></multiply>
          <num>7</num>
        </subtract>
        <num>11</num>
      </divide>
      <multiply>
        <num>22</num>
        <log10>
          <num>6</num>
        </log10>
      </multiply>
    </subtract>
  </formula>
</set_param>
```

Figure 5.20 Example of custom mathematical operators for formula element expressed in XML.

The custom subset approach has the potential benefit of simplicity in that a limited set of mathematical operators and functions could be implemented, and anything requiring more complex processing could be implemented into an external program (see Section 5.4.6). Furthermore, because everything is explicitly expressed with XML elements as with the MathML solution, there is no certainty of security as the formula is not evaluated as an executable program statement. Yet despite these benefits, this approach would still be "re-inventing the wheel". Alternatively, the custom subset could be derived from MathML, including only the basic mathematical requirements. As a proof of concept, adding support for a multitude of mathematical operators and functions as separate elements either anew or from MathML would contribute very little to the research. As the programming language approach will easily demonstrate the functionality of the formula parameter type, it has been implemented.

The *formula* element allows the evaluation of a mathematical expression in calculating a parameter value. However, sometimes it is not possible to use a formula

to calculate a value and alternate methods are required such as the *program* element (see Section 5.4.6) or the *condition* element.

5.4.5 condition element

A *condition* element is a complex decision structure that allows parameters to be assigned values based on the outcome of a conditional test. A simple example is shown through the question in Figure 5.21.

```
If you have <param ident=”myapples”>1..5</param> <param ident=”plural_yes_no”>apple,
apples</param> and you combine them with your friend’s 5 apples, then how many do you
have all together?
```

Figure 5.21 Simple example question demonstrating need for conditional parameters.

If the parameter myapples has a value of between 2 and 5, then the plural word “apples” needs to be used for the “plural_yes_no” parameter. If the value of myapples is 1, then the singular word “apple” is required.

Another example question necessitating the use of conditional parameters is illustrated in Figure 5.22. In this particular question, the enumerated list of UDP, TCP, or IP are randomly chosen options for the question. Both the UDP and TCP parameters have the answer “Layer 4”, whereas the option IP has the answer “Layer 3”.

```
For the TCP/IP protocol, in which OSI Relational Model Layer does the
<param>tcp,udp,ip</param> protocol belong?

a) Layer 1
b) Layer 2
c) Layer 3
d) Layer 4
e) Layer 5
```

Figure 5.22 Example question requiring the use of conditional parameters.

As this cannot be expressed as a formula or mathematically, a conditional test is required to assign which of the multiple choice options is the correct answer. This is achieved through the *condition* element. The *condition* element supports:

- conditional testing of previously instantiated parameter values,
- the ability to assign values using the *set_param* element based on the outcome of the conditional tests,
- the ability to perform different assignments based on whether the condition is true or false,
- dealing with intersection or overlap of parameter values.

Figure 5.23 provides a diagram, which represents the complex structure of the *condition* element.

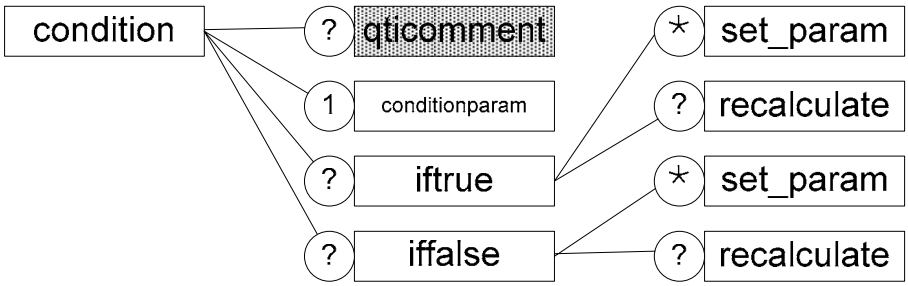


Figure 5.23 condition element structure.

The *condition* element structure is synonymous with the *respcondition* element (refer to Section 3.3.15). It allows an optional *qticomment* element, followed by a *conditionparam* element. The *conditionparam* element structure describes the logical test to be performed on the values of parameters contained therein, in the same way that the *conditionvar* element structure (refer to Section 3.3.16) performs logical tests on the candidate's responses. Figure 5.24 shows the breakdown of the *conditionparam* element structure. One difference between *respcondition* and the

condition element is the *condition* element supports different outcomes for both a true and false result. This is achieved by the implementation of the *iftrue* and *iffalse* elements within the *condition* element. If the result from the *conditionparam* structure is true, then all the structures within the *iftrue* element are processed. Conversely, if the *conditionparam* structure returns false, then *iffalse* is processed. The *conditionparam* element as shown in Figure 5.24 is a child of the *condition* element and describes the logical test to be performed on the values of parameters contained therein, in the same way that the *conditionvar* element structure performs logical tests on the candidate's responses.

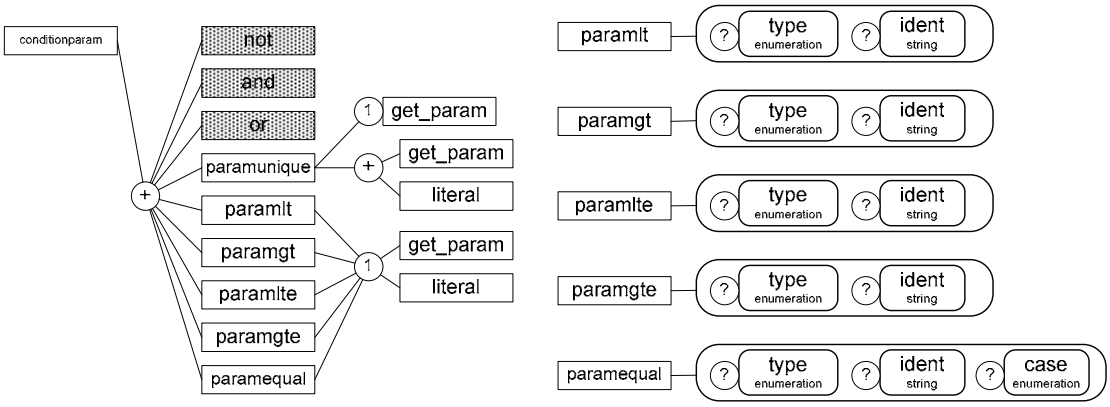


Figure 5.24 conditionparam element structure.

The *not*, *and*, and *or* elements from the existing QTI Specification have been re-used providing the same logical tests as used in the response processing component of the specification. A range of new elements has been implemented to support the evaluation of parameter values, rather than re-using the existing elements as part of the *conditionvar* structure, such as *varequal*. Further discussion on this point can be found later in this section. Table 5.5 shows a mapping between the existing response processing elements, the new parameter processing elements and a brief explanation of their purpose.

Table 5.5 Mapping between response processing, and new parameter processing elements

Response Processing	Parameter Processing	Purpose
not	not	To negate the logical test
and	and	To perform boolean ‘and’ operation between two or more tests
or	or	To perform boolean ‘or’ operation between two or more tests
varequal	paramequal	Test for equality
varlt	paramlt	Less than test
vargt	paramgt	Greater than test
varlte	paramlte	Less than or equal test
vargte	paramgte	Greater than or equal test
	paramunique	Test all values within are unique
	literal	Tags a literal value
	iftrue	Perform parameter assignment if evaluation returns true
	iffalse	Perform parameter assignment if evaluation returns false
	recalculate	Perform a recalculation on all parameter values within parameters section

As illustrated in Table 5.5, the naming convention used with the new elements, mimic the existing names. To see an example using these new elements, refer to Figure 5.25, which illustrates the XML *condition* structure representing the question shown in Figure 5.21.

```
<parameters>
  <set_param ident=”myapples”>
    <range step=”1”>
      <range_min><literal>1</literal></range_min>
      <range_max><literal>5</literal></range_max>
    </range>
  </set_param>

  <condition>
    <conditionparam>
      <paramequal type=”Parameter” ident=”myapples”><literal>1</literal></paramequal>
    </conditionparam>

    <iftrue>
      <set_param ident=”plural_yes_no”><literal>apple</literal></set_param>
    </iftrue>

    <iffalse>
      <set_param ident=”plural_yes_no”><literal>apples</literal></set_param>
    </iffalse>
  </condition>
</parameters>
```

Figure 5.25 Simple Example XML for condition element.

If the parameter myapples has a value of 1, then the parameter plural_yes_no is set to “apple”, otherwise it is set to “apples”. The plural_yes_no parameter is then presented to the candidate immediately after the myapples parameter resulting in correct grammar. The XML structure for *condition* is similar to the *respcondition* structure as described previously. The test is performed using the *paramequal* element and the *ident* attribute refers a parameter rather than a candidate response id. This is denoted by the *type* attribute being set to ”Parameter”. If the *type* attribute is set to “Response”, then the comparison *ident* is with the identifier of a response provided by the candidate. This allows these parameter processing elements to be re-used as part of response processing (see Section 5.6). Both the type and ident attributes are implemented for all of: *paramequal*, *paramgt*, *paramlt*, *paramgte*, and *paramlte* elements. The attribute name *ident* was chosen over *respident* as is used by the *conditionvar* element structure, as *ident* can more generally refer to either the identity of a parameter or the identity of a response.

Another example of the condition parameter type is shown in Figure 5.26, which describes the parameters for the question illustrated in Figure 5.22.

```
<parameters>
  <set_param ident=”protocol”>
    <enumeration>
      <literal>TCP</literal>
      <literal>UDP</literal>
      <literal>IP</literal>
    </enumeration>
  </set_param>

  <condition>
    <conditionparam>
      <or>
        <paramequal type=”Parameter” ident=”protocol”>
          <literal>TCP</literal>
        </paramequal>
        <paramequal type=”Parameter” ident=”protocol”>
          <literal>UDP</literal>
        </paramequal>
      </or>
    </conditionparam>

    <iftrue>
      <set_param ident=”answer”><literal>Layer 4</literal></set_param>
    </iftrue>
    <iffalse>
      <set_param ident=”answer”><literal>Layer 3</literal></set_param>
    </iffalse>
  </condition>
</parameters>
```

Figure 5.26 Example XML for condition element.

If the parameter value selected for protocol is either TCP or UDP, then the parameter answer is set to “Layer 4”, otherwise it is set to “Layer 3”. This example includes the use of the *or* element which allows a logical inclusive “or” to be performed on the tests therein, being the *paramequal* tests.

Overlap or intersection of parameter values within the question is particularly important when generating parameters for multiple choice questions (see Section 4.5.2). This must also be considered. Take the example question shown in Figure 5.11 of Section 5.4.3 asking the candidate to complete a UNIX sed command to replace one string of characters with another for the /etc/passwd file. The parameter oldshell has the same enumerated list of options as the parameter newshell. Therefore, given that both parameters will be randomly set with one of those options there is a 1/3 probability that both oldshell and newshell will have the same parameter

value. For example, if oldshell was set to "/bin/bash" and newshell was set to "/bin/bash", then question would be asking to replace the string "/bin/bash" with "/bin/bash" which would not make a great deal of sense. There are two approaches that have been implemented into the QTIPA to solve this type of problem. They are described in Table 5.6.

Table 5.6 Approaches to deal with parameter value intersection

Avoidance	This approach requires the use of condition tests to prevent a value already assigned to one parameter, being assigned to another.
Detection	This approach requires the use of condition tests to detect when two or more parameter values overlap, and instruct the system to re-calculate the values over again.

By using an avoidance method, conditions need to be implemented for every parameter assignment with the *set_param* element (with exception to the initial *set_param*), where the conditions constrain what possible values could be assigned to the parameter variable based on what parameter values have already been assigned. The advantages and disadvantages of the avoidance method are described in Table 5.7.

Table 5.7 Advantages & Disadvantages of the avoidance intersection method

Avoidance intersection method	
Advantages:	By implementing avoidance of intersection, at the end of parsing and generating all the parameters, it is assured that there is no intersection (assuming the logic of the conditions are correct)
	Parameters need only be parsed once
Disadvantages:	Easy to make an error in the logic used for conditions, particularly where they become quite complex
	Complexity increases significantly with the more possible values to test

Using the question as described in Figure 5.11, an example of this approach is shown in Figure 5.27. As previously described, it is not appropriate to have both oldshell and newshell parameters set to the same value.

```
<parameters>
  <set_param ident=”oldshell”>
    <enumeration>
      <literal>/bin/bash</literal>
      <literal>/bin/tcsh</literal>
      <literal>/bin/false</literal>
    </enumeration>
  </set_param>

  <condition>
    <conditionparam>
      <paramequal type=”Parameter” ident=”oldshell”>
        <literal>/bin/bash</literal>
      </paramequal>
    </conditionparam>

    <iftrue>
      <set_param ident=”newshell”>
        <enumeration>
          <literal>/bin/tcsh</literal>
          <literal>/bin/false</literal>
        </enumeration>
      </set_param>
    </iftrue>
  </condition>

  <condition>
    <conditionparam>
      <paramequal type=”Parameter” ident=”oldshell”>
        <literal>/bin/tcsh</literal>
      </paramequal>
    </conditionparam>

    <iftrue>
      <set_param ident=”newshell”>
        <enumeration>
          <literal>/bin/bash</literal>
          <literal>/bin/false</literal>
        </enumeration>
      </set_param>
    </iftrue>
  </condition>

  <condition>
    <conditionparam>
      <paramequal type=”Parameter” ident=”oldshell”>
        <literal>/bin/false</literal>
      </paramequal>
    </conditionparam>

    <iftrue>
      <set_param ident=”newshell”>
        <enumeration>
          <literal>/bin/bash</literal>
          <literal>/bin/tcsh</literal>
        </enumeration>
      </set_param>
    </iftrue>
  </condition>
</parameters>
```

Figure 5.27 Example of conditions used for intersection avoidance.

The *condition* statements test each of the possible outcomes for the parameter variable oldshell and then constrain the enumerated list of possible values for newshell so as not to include the value already assigned to oldshell. This approach will assure that there is no intersection of values. However, as demonstrated in Figure 5.27, it can be quite complicated even with only 3 possible values. The complexity of such an approach will grow significantly, the more possible intersectable values that are available, and the number of parameters that are conditional on the values of others.

In contrast, by using the detection method, a single condition needs to be included after all parameter values have been instantiated. This condition would be responsible for detecting any intersection of values for any group of parameters. Once detected, a method of correcting the intersection is used. The advantages and disadvantages of the detection method are described in Table 5.8below.

Table 5.8 Advantages & Disadvantages of the detection intersection method

Detection intersection method	
Advantages:	No matter how many parameters or values are conditional on being unique, the complexity of this method is rather limited as only a single test is required
Disadvantages:	With poorly designed parameters, there is the risk of many (or even infinite) iterations of calculating the parameters will be necessary to result in all parameter values being unique.
	If an intersection is detected, the parameter values must be recalculated again, which adds processing overhead to the quiz system.

To support the detection of intersecting parameter values, an extra element has been introduced above and beyond what is provided by the standard response processing condition tests. The element *paramunique* is provided where by a test will be

performed on each of the parameter values provided, returning true if all of which are unique.

Once again, using the question as described in Figure 5.11, an example of using the detection approach is shown in Figure 5.28.

```
<parameters>
  <set_param ident="oldshell">
    <enumeration>
      <literal>/bin/bash</literal>
      <literal>/bin/tcsh</literal>
      <literal>/bin/false</literal>
    </enumeration>
  </set_param>

  <set_param ident="newshell">
    <enumeration>
      <literal>/bin/bash</literal>
      <literal>/bin/tcsh</literal>
      <literal>/bin/false</literal>
    </enumeration>
  </set_param>

  <condition>
    <conditionparam>
      <paramunique>
        <get_param ident="oldshell"/>
        <get_param ident="newshell"/>
      </paramunique>
    </conditionparam>

    <iffalse>
      <recalculate/>
    </iffalse>
  </condition>
</parameters>
```

Figure 5.28 Example of condition used for intersection detection.

So far, methods for detection of the intersection have been discussed, but not how to deal with an intersection when it occurs. Another element *recalculate* has been introduced which indicates that all the parameters must be recalculated again. The existing structure of the *condition* element already supports intersection avoidance. Given that the complexity of the avoidance approach can become quite cumbersome when there are many possible values and parameters; the detection approach has also been implemented using the *paramunique* and *recalculate* elements.

Another example for using condition parameters within a question is demonstrated in Figure 5.29. The question to which it refers is described in Figure 5.5 from Section 5.4.2 where the candidate is asked to multiply two parameterised numbers together. The constraints on this question are such, that if the parameter num1 is assigned a value greater than 10, then the value assigned to num2 must be less than or equal to 10, thus limiting the difficulty of the question. This would be useful in the situation where the question is for primary school children.

```
<parameters>
  <set_param ident=”num1”>
    <range>
      <range_min><literal>1</literal></range_min>
      <range_max><literal>20</literal></range_max>
    </range>
  </set_param>

  <condition>
    <conditionparam>
      <paramgt type=”Parameter” ident=”num1”><literal>10</literal></paramgt>
    </conditionparam>
    <iftrue>
      <set_param ident=”num2”>
        <range>
          <range_min><literal>1</literal></range_min>
          <range_max><literal>10</literal></range_max>
        </range>
      </set_param>
    </iftrue>
    <iffalse>
      <set_param ident=”num2”>
        <range>
          <range_min><literal>1</literal></range_min>
          <range_max><literal>15</literal></range_max>
        </range>
      </set_param>
    </iffalse>
  </condition>
</parameters>
```

Figure 5.29 Example XML for condition element.

The implementation of the condition parameter has been discussed, including examples of its use. Alternate approaches have also been investigated and are described below.

There are quite a few different approaches that can be used, most of which are based on the typical syntax for conditions of common programming languages. For

example, the “if, then, else” construct, or the case (or switch) statement as illustrated in Figure 5.30 using C syntax.

```
if (<get_param ident=”option1”/> == 20)
  <set_param ident=”option2”><literal>5</literal></set_param>
else
  <set_param ident=”option2”><literal>10</literal></set_param>

//or

switch (get_param ident=”option1”/>)
{
  case 20:  <set_param ident=”option2”><literal>5</literal></set_param> ;
  default:  <set_param ident=”option2”><literal>10</literal></set_param> ;
}
```

Figure 5.30 Example of conditions implemented using C syntax.

Using this approach allows for complex conditions to be performed, using familiar programming constructs. Table 5.9 explains the advantages and disadvantages to using this approach for implementation of conditions into the QTI Parameterisation Addendum.

Table 5.9 Advantages & Disadvantages of Programming Syntax for Condition Element

Use of programming syntax for implementation of conditional parameter types	
Advantages:	Allows for complex conditions to be performed
	Use of the switch statement provides for easy multi-way branching conditions which in turn provides a less verbose output
	Less parsing of XML structures required
	Easier to interpret the written XML than the other methods
Disadvantages:	This approach does not make use of existing QTI element structures that support conditional testing (QTI Response Processing)
	This approach would not make it possible to perform XML validation of the condition syntax
	Must use a platform-independent language such as Perl or Java for the QTI Item to be interoperable
	Security issues arise due to the fact that code written by the question author will be executed within the quiz system, therefore checks would be necessary to ensure nothing malicious is included.

The *respcondition* element as discussed in Section 3.3.15 provides for a conditional test via the *conditionvar* element which if evaluates to true, will allow the manipulation of scoring variables through the multiple *setvar* elements. The *conditionvar* element could be re-used along with all the logical test elements for performing logical tests on parameter values, in the same way they are used to test candidate response values. Therefore, another approach that could be used to support the conditionparam structure is to re-use all the existing elements as per the response processing column in Table 5.5. In other words, re-use all the existing response

processing elements to support conditional testing of parameter values. Table 5.10 shows the advantages and disadvantages of using this approach.

Table 5.10 Advantages & Disadvantages of re-using existing QTI conditions for parameter condition element

Use of existing conditional test elements provided through the conditionvar element for conditional testing of parameter values	
Advantages:	Re-use of existing QTI structures will allow better integration of this functionality, along with more efficient implementation for vendors, due to re-use of coding.
	Re-use of existing QTI structures will also make the addendum quicker and easier to develop and maintain.
	The condition can be more thoroughly validated through the XML DTD
	As the condition is described entirely through the XML, the vendor is free to implement this functionality in anyway, without consideration of programming language.
	Less security issues as the element does not contain explicit code, which would need to be checked for malicious intent.
Disadvantages:	May lack some flexibility for highly complex conditions.
	The condition will be somewhat more difficult to interpret from reading the XML.
	Very difficult to re-use the existing elements without making the elements incompatible with existing QTI documents.

To allow the re-use of the existing elements, changes would be required that may be incompatible with existing QTI documents. Some of the difficulties associated with re-use are:

- the response processing elements provide outcomes only for true evaluations,

- the elements responsible for evaluating candidate responses (i.e. *varequal*, *varlt*) only allow PCDATA values,
- the attribute responsible for identifying which response ident the evaluations should be performed on is called respident, which cannot be re-used to identify parameters.

The existing structure would need to be modified in some way to support assignment of parameter values when evaluations return true, and also for false. Furthermore, the elements responsible for evaluating candidate responses would be required to perform evaluations on parameter values where the *get_param* element would need to be a sub-element (identifying the parameter value to compare with). Yet, because they only allow PCDATA, the elements would need to be changed to mixed content to allow both *get_param* and PCDATA. The use of mixed content seriously reduces the constraints on the content of these elements, as it would be preferred only to have either PCDATA, or *get_param*. However, mixed content does not allow the order or occurrences of the allowed elements and PCDATA to be constrained. A possible solution may be to introduce a third child element named *literal*, which would contain what would normally be simply PCDATA within *varequal* for example. Then phase out the PCDATA for future releases providing time for older QTI documents to be converted to the newer format. This is still less than ideal. Finally, the respident attribute could not be used to identify a parameter ident, given its name (misleading). Also given it is a mandatory attribute, it could not be replaced with an attribute for identifying parameters like *paramident*.

Alternate methods to solve the parameter intersection problem will now be discussed. One such approach would be only to recalculate the intersecting parameters. This approach will involve complexity as shown with the avoidance method.

A simpler method could be introduced where the *set_param* element would have an attribute *unique* which when set to yes would require that the value assigned to that parameter be unique with all other parameters. Of course, this has serious limitations. For example, it could only work on a parameter that is being assigned a random value such as a *range* or *enumeration*; and it may be necessary to have only a subset of parameters assured of being unique. Alternatively, another special type of *condition* type element could have been introduced. If this condition detects parameter value intersection, recalculation would result. By introducing the *recalculate* element, there need only be one "all purpose" *condition* element, which can handle both requirements, with less added complexity.

Of all the alternatives, the chosen approach allows re-use of the design, and assures backward compatibility. It is customised specifically for use with parameters, and still uses consistent logic with other components of the specification. Thus it is more effective.

Thus far all of range, enumeration, formula, and condition parameter types have been examined. What if none of these types can derive a parameter value that is required for a question? This is where the *program* element becomes useful. The next section discusses the use of custom written computer programming for assigning parameter values.

5.4.6 program element

The parameter type will introduce the capability for making calls to external programs or library functions. It will essentially provide unlimited means to calculating a parameter value. This will be particularly useful where the formula and condition parameter types are insufficient for generating a value. It will also be useful for questions that assess content areas associated with computing, for example computer programming. Figure 5.31 shows an example question that would be ideally suited to this parameter type.

```
Given the following C code:

#include <stdio.h>

int main(void)
{
  int a ;
  int b ;
  int c = <param ident=”varc”>1..5</param> ;

  for(a=3;a<10;a++)
    for(b=<param ident=”varb”>5..7</param>;b<10;b+=2)
      c = c + b + a ;

  printf ("c=%i",c) ;
  exit(0) ;
}

The output will be:

c=<text box>

Complete the text box above showing the output from this code.
```

Figure 5.31 Example question suitable for calling external program code.

Although the formula or condition parameter types could be used to determine the output from this C program, a far simpler method would be to actually include the parameter values within the C program itself, and execute it to determine the output.

This functionality is introduced through the program parameter illustrated in Figure 5.32 which:

- supports a wide range of programming languages,
- accepts instantiated parameter values as input,
- generates output to be stored in another question parameter.

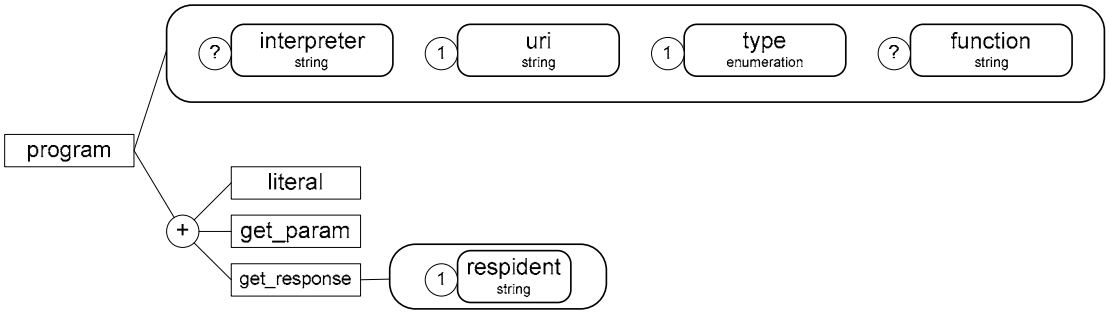


Figure 5.32 program element structure.

The *program* element has an array of attributes, helping to describe the program code and how it is to be executed. For example, the *type* attribute describes whether the program code is via an external file by the value “External”, or whether it is via a library function by the value “Function”. The file which contains the library or external program is represented by the *uri* attribute which is the standard attribute named used within the QTI Specification for referencing an external file/document. If the *type* attribute is “Function”, then the attribute *function* identifies the name of the function to call within the library identified by the *uri* attribute. Finally, if the file identified by the *uri* attribute is a script, which requires execution via an interpreter (such as Perl, or tcl), then the path to the interpreter executable file is provided through the *interpreter* attribute.

The child elements of *program* are used exclusively to pass literal, parameter and candidate response values to the program. This is achieved through the *literal*, *get_param*, and *get_response* elements as described in previous sections. At least one of these elements must be provided, but there can be a mixture. Once these values

have been provided to the program on which the algorithm is to base its processing, the output from the program must then be assigned as a value to a parameter within the question XML definition. This is achieved by using the default output stream of the program to return the resulting parameter value. This is not unlike how a CGI (Common Gateway Interface) program sends output back to the web server, which in turn sends back to the client browser. In the case of a library function, its return value is used to set the resulting parameter value.

An example of the supporting XML is represented in Figure 5.33 for the question described in Figure 5.31.

```
<parameters>
  <set_param ident=”varc”>
    <range step=”1”>
      <range_min><literal>1</literal></range_min>
      <range_max><literal>5</literal></range_max>
    </range>
  </set_param>

  <set_param ident=”varb”>
    <range step=”1”>
      <range_min><literal>5</literal></range_min>
      <range_max><literal>7</literal></range_max>
    </range>
  </set_param>

  <set_param ident=”answer”>
    <program type=”External” uri=”/path/to/program/testnestedloops”>
      <get_param ident=”varc”/><get_param ident=”varb”/>
    </program>
  </set_param>
</parameters>
```

Figure 5.33 Example XML supporting program parameter type.

The parameters varc and varb are set and then the parameter answer is set to the output provided by the /path/to/program/testnestedloops program which is executed with the arguments provided, being parameter values varc and varb. The path to the program is represented by the *uri* attribute, which in this example is the string “/path/to/program/testnestedloops”. The *type* attribute specifies that the program

identified by *uri* will be an “External” application that is called. The program code for testnestedloops is shown in Figure 5.34.

```
 1: #include <stdio.h>
 2:
 3: int main(int argc, char *argv[])
 4: {
 5:   int varc = atoi(argv[1]) ;
 6:   int varb = atoi(argv[2]) ;
 7:   int a ;
 8:   int b ;
 9:   int c = varc ;

10:   for(a=3;a<10;a++)
11:     for(b=varb;b<10;b+=2)
12:       c = c + b + a ;

13:   printf ("%i",c) ;
14:   exit(0) ;
15: }
```

Figure 5.34 Example program code for use with program element.

The first parameter provided as per the XML shown in Figure 5.33 is “varc”, which is then assigned to the same named C variable on line 5. Likewise, the second parameter provided, “varb” is then assigned to the same named C variable on line 6. This is how the question parameter values are aligned with the variables within the program code. Now that the instantiated values assigned to the parameters from the XML binding have been passed to the program, it can execute and generate the output via the printf function on line 13, which is then assigned to the parameter “answer”. Note that the string “c=” has been removed from the printf function as that is not required to be entered by the candidate as represented in the question defined provided in Figure 5.31, and so should not be assigned to the answer parameter for comparison with the candidate’s response.

A more complex example question is shown in Figure 5.11 where the input provided by the candidate will be used by the parameters processing elements to calculate a value, which will be compared with a known correct answer. In other words, the two

text strings the candidate provides as responses to the question will be passed to an external program, rather than just providing instantiated parameters to the program as in the example shown in Figure 5.33. To achieve this, the QTIPA supports instantiation of parameter values within the response processing component of the specification; such that the candidate's responses can be used in calculating parameter values (see Section 5.4). Therefore, there also needs to be support for referencing a response value in the same way that the *get_param* element supports referencing of parameter values. This will allow the passing of the candidate's responses to the external program responsible for executing the sed command (from Figure 5.11) in the same way that a candidate's response can be passed to the *formula* element structure (see Section 5.4.4). Figure 5.35 shows an XML example that demonstrates the instantiation of parameter values and the response processing associated with the question in Figure 5.11. For further details on the response processing elements, refer to Section 5.6.

```
  <item ident="External_Program_Example">
    <parameters>
      <set_param ident="oldshell">
        <enumeration>
          <literal>/bin/bash</literal>
          <literal>/bin/tcsh</literal>
          <literal>/bin/false</literal>
        </enumeration>
      </set_param>
      <set_param ident="newshell">
        <enumeration>
          <literal>/bin/bash</literal>
          <literal>/bin/tcsh</literal>
          <literal>/bin/false</literal>
        </enumeration>
      </set_param>
      <condition>
        <conditionparam>
          <paramunique>
            <get_param ident="oldshell"/>
            <get_param ident="newshell"/>
          </paramunique>
        </conditionparam>
        <iffalse>
          <recalculate/>
        </iffalse>
      </condition>
    </parameters>
    <resprocessing>
      <outcomes>
        <decvar defaultval="0"/>
      </outcomes>
      <parameters>
        <set_param ident="iscorrect">
          <program uri="/usr/local/bin/regularexpression.pl" interpreter="perl"
type="External">
            <get_param ident="oldshell"/>
            <get_param ident="newshell"/>
            <get_response respident="candidateresponse1"/>
            <get_response respident="candidateresponse2"/>
          </program>
        </set_param>
      </parameters>
      <respcondition>
        <conditionvar>
          <paramequal type="Parameter" ident="iscorrect">
            <literal></literal>
            <!--The program will return no output if candidate solution-->
            <!-- is correct.  Otherwise iscorrect parameter will hold  -->
            <!-- the output from the program using the candidate's     -->
            <!-- which could be used as part of the itemfeedback       -->
          </paramequal>
        </conditionvar>
        <setvar action="Add">1</setvar>
      </respcondition>
    </resprocessing>
  </item>
```

Figure 5.35 Example XML supporting the passing of candidate response values to a program.

The *get_response* element has been used to pass the two candidate responses to the Perl program along with the typical *get_param* elements providing the two instantiated values within the question stem. When the program is written using an interpreted language such as Perl or Java, the *interpreter* attribute is used to identify

the name of the interpreter to launch to execute the script file provided by *uri*. In this example, the interpreter attribute is set to "perl". From the four parameter values provided, the Perl script knows how to calculate the output using a correct solution to the problem, and can also use the candidate's answer to calculate a comparison with the known correct solution. If the output matches between the two, then the candidate's solution is correct. Figure 5.36 shows the Perl script, which is called via the program element as demonstrated in the example XML shown in Figure 5.35.

```
#!/usr/bin/perl

my ($oldshell,$newshell,$response1,$response2) = @ARGV ;

#Perform taint checking on candidate input
#by removing any single quote characters
$response1 =~ s/'//g ;
$response2 =~ s/'//g ;

my $candidatesedcommand =<<EOF ;
/bin/sed -e 's/$response1/$response2/g' /etc/passwd
EOF


$oldshell =~ s#/#\\/#g ;
$newshell =~ s#/#\\/#g ;
$oldshell = ":$oldshell\$" ;
$newshell = ":$newshell" ;

my $correctsedcommand =<<EOF ;
/bin/sed -e 's/$oldshell/$newshell/g' /etc/passwd
EOF

my $candidateoutput = `$candidatesedcommand` ;

my $correctoutput = `$correctsedcommand` ;

print $candidateoutput
  if("$candidateoutput" ne "$correctoutput") ;
```

Figure 5.36 Example Perl script supporting use of get_response element via program element.

The algorithm will taint check the responses provided by the candidate to ensure there are no single quotes. If there is a single quote in the candidate's responses, then they can execute other UNIX Shell commands, which would compromise system security. This is because the candidate's responses are enclosed within single quotes ensuring the shell, which executes the sed command, does not parse any special shell characters provided in the candidate's response and executes them. After taint checking, the

script will include the candidate's responses as the arguments to the actual sed command, making it ready to execute via the shell. Then the script will alter the parameters values for the Perl variables oldshell and newshell, such that they themselves are regular expressions that will correctly perform the task asked by the question. Both versions of the sed command are executed and the output from the commands stored in Perl variables. These variables are then compared and if not equal, the output from the candidate's version of the sed program is returned by the Perl program, which will then be stored in the QTIPA parameter "iscorrect". The response processing XML will then check to see if the parameter "iscorrect" is empty, and if so, will add 1 mark to the scoring variable, marking it correct. If the parameter "iscorrect" is not empty, no marks will be given for the question, and the "iscorrect" parameter value could be used as part of the item feedback to the candidate (see Section 3.3.13 for information on the *itemfeedback* element).

There are many advantages to implementation of the *program* element using positional arguments. These advantages are discussed in Table 5.11 along with one disadvantage.

Table 5.11 Advantages & Disadvantages to using positional arguments for passing parameter values

Use of positional arguments for communicating question parameter values to a program	
Advantages:	Positional arguments are a common interface used in modern programming languages
	The positional arguments could be provided to an executable program, or alternatively directly to a library function provided with the question
	No need to tokenise and parse the values as they will be identified in the program code by the position (order) they are provided.
Disadvantages:	Can be difficult to align parameter values in XML to the program, when the number of arguments to pass is variable

This approach has been implemented based on the advantages of using positional arguments, and the fact that the positional arguments can easily be provided to library functions, rather than being limited to external executables. However, alternate approaches were investigated, and will now be examined.

Almost all modern programming languages support file I/O, therefore one alternate method of providing input to the program would be via a file stream. For a UNIX environment, it could simply be a pipe or a socket. The XML for the question item would need to be aligned with the program code, such that the program knows which parameter values it is receiving when reading the input. A simple method could be to name the values using the parameter name and make it line oriented; so the input on the file could be of the format shown in Figure 5.37.

```
param1=value
param2=anothervalue
...
```

Figure 5.37 Example of how file input can be used for program code parameter types.

This would be read until end of file was reached at which time the program could begin its algorithm for generating a value for a parameter. Table 5.12 shows the advantages and disadvantages of using this approach.

Table 5.12 Advantages & Disadvantages to using File I/O for passing parameter values

Use of file I/O for communicating question parameter values to a program	
Advantages:	Almost all modern programming languages and operating systems support file I/O ensuring portability
Disadvantages:	Not all operating systems support pipes or sockets, which would be the most convenient methods of implementation
	The file input would need to be tokenised (as per Figure 5.37, meaning special characters such as the = sign and LF need to be parsed and escaped
	Can make the resulting programs somewhat more complex than need be (like passing positional arguments)

Rather than using the format as shown in Figure 5.37 for encoding the parameter values on a file stream, the parameter values could be encoded using XML. This would eliminate the problem with having to parse and tokenise the input as an XML parser could perform this task. The problem with this approach is every program written for calculating parameters would need to include/link an XML parser and then process the parsed input. Potentially, this can add significant overhead in parsing the XML and make a simple algorithm considerably more complex. This is especially true when the objective is simply to pass a list of parameter values to a program.

The parameters declaration structure is responsible for defining and instantiating the values for parameters before presentation to the candidate, and optionally before processing the candidate's response. It supports all of the text media parameter types as shown in Table 4.1 through the range, enumeration, condition, formula, and program element structures. Now that the parameter processing and instantiation has been defined, presentation of the resulting parameter values will be examined in the following section.

5.5 Parameter Presentation

The presentation structure of the QTI Specification (see Section 3.3.8) describes how the presentation of the question stem and optional distracters and key should be displayed to the candidate. With the introduction of parameters to the question definition, provision needs to be made for allowing the presentation of the parameter values seamlessly with the existing question material. Example parameterised questions from Figure 4.5 in Section 4.4 will be used throughout to help illustrate how the parameter values can be presented to the candidate through the *presentation* element structure.

Within the structure of the *presentation* element, the *material* element is responsible for describing/rendering how material is presented to the candidate (see Section 3.3.12). It is within the *material* element that the presentation of parameter values support will be provided, re-using the existing QTI structure. The new structure of the *material* element is illustrated in Figure 5.38.

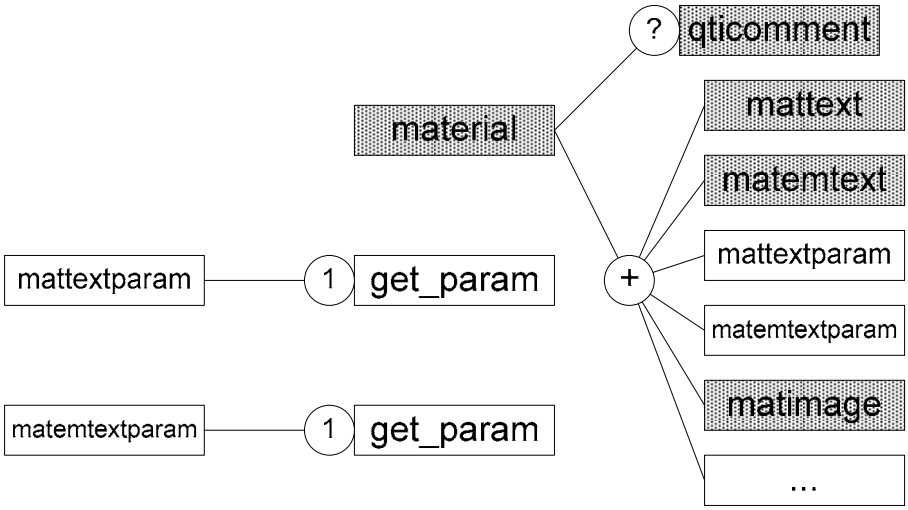


Figure 5.38 Amendments to material element structure.

To support text parameters within the *material* element, the *mattextparam* element has been introduced as shown in Figure 5.38, which will present textual parameter values in the same way as the existing *mattext* element. Likewise, the *matemtextparam* element will present emphasised textual parameter values in the same way as the existing *matemtext* element. Each of these two elements must have one and only one *get_param* element as a child, which provides the *ident* attribute identifying the parameter value is to be rendered. The use of the *mattextparam* element is illustrated in Figure 5.39, for the question shown in Figure 5.22. The *parameters* element declaration for this question has been previously illustrated in Figure 5.26 of Section 5.4.5.

```
    <presentation>
      <flow>
        <material>
          <mattext>For the TCP/IP protocol, in which OSI relational model layer does
the </mattext>
          <mattextparam>
            <get_param ident="protocol"/>
          </mattextparam>
          <mattext> protocol belong?</mattext>
        </material>
        <response_lid ident="MCO1">
          <render_choice shuffle="No">
            <response_label ident="A">
              <flow_mat>
                <material>
                  <mattext>Layer 1</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="B">
              <flow_mat>
                <material>
                  <mattext>Layer 2</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="C">
              <flow_mat>
                <material>
                  <mattext>Layer 3</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="D">
              <flow_mat>
                <material>
                  <mattext>Layer 4</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="E">
              <flow_mat>
                <material>
                  <mattext>Layer 5</mattext>
                </material>
              </flow_mat>
            </response_label>
          </render_choice>
        </response_lid>
      </flow>
    </presentation>
```

Figure 5.39 Example XML showing use of parameters with QTI Presentation structure.

The *mattextparam* element must have only 1 sub-element, being *get_param*, which using this specific combination will display the value of the parameter represented by the *ident* attribute of *get_param*. The rendered version of this question with the parameter values instantiated is shown in Figure 4.7 of Section 4.7.2.

Another example is provided in Figure 5.40 in which parameter values are used within the distracters and key of the question presentation. The question is designed to test the candidate's knowledge on mathematical operator precedence.

```
  <!--A question designed to test candidate's understanding of mathematical operator
precedence.  One of the distracters in this question erroneously changes the order of
precedence for the formula to be evaluated only from left to right.  Another
distracter
erroneously changes the value of the parameter "e" to being its positive counterpart.
That is, it uses the absolute value of "e" rather than negative.


What is the value of "a", given the following expression?

a = c + d - b * e - d / b

parameters
b=1..4
c=1..5
d=3*b
e=-5..0

A)
B)
C)
D) None of the above

-->
  <item ident="Dynamic_MCQ_Example">
    <parameters>
      <set_param ident="varb">
        <range>
          <range_min>
            <literal>1</literal>
          </range_min>
          <range_max>
            <literal>4</literal>
          </range_max>
        </range>
      </set_param>
      <set_param ident="varc">
        <range>
          <range_min>
            <literal>1</literal>
          </range_min>
          <range_max>
            <literal>5</literal>
          </range_max>
        </range>
      </set_param>
      <set_param ident="vard">
        <formula prog_language="perl">=3*<get_param ident="varb"/>
        </formula>
      </set_param>
      <set_param ident="vare">
        <range>
          <range_min>
            <literal>-5</literal>
          </range_min>
          <range_max>
            <literal>0</literal>
          </range_max>
        </range>
      </set_param>
      <set_param ident="optionA">
        <formula prog_language="perl">
          <!-- A=((c + d - b) * e - d) / b -->
=((<get_param ident="varc"/> + <get_param ident="vard"/> - <get_param ident="varb"/>)
* <get_param ident="vare"/> - <get_param ident="vard"/>) / <get_param ident="varb"/>
        </formula>
      </set_param>
      <set_param ident="optionB">
        <formula prog_language="perl">
          <!-- B=c + d - b * abs(e) - d / b -->
=<get_param ident="varc"/> + <get_param ident="vard"/> - <get_param ident="varb"/> *
abs(<get_param ident="vare"/>) - <get_param ident="vard"/> / <get_param ident="varb"/>
        </formula>
      </set_param>
      <set_param ident="optionC">
        <formula prog_language="perl">
```

```
          <!-- CORRECT -->
=<get_param ident="varc"/> + <get_param ident="vard"/> - <get_param ident="varb"/> *
<get_param ident="vare"/> - <get_param ident="vard"/> / <get_param ident="varb"/>
        </formula>
      </set_param>
      <condition>
        <conditionparam>
          <paramunique>
            <get_param ident="optionA"/>
            <get_param ident="optionB"/>
            <get_param ident="optionC"/>
          </paramunique>
        </conditionparam>
        <iffalse>
          <recalculate/>
        </iffalse>
      </condition>
    </parameters>
    <presentation>
      <flow>
        <material>
          <mattext>What is the value of a, given the expression below?</mattext>
        </material>
        <flow>
          <material>
            <mattext>a = </mattext>
            <mattextparam>
              <get_param ident="varc"/>
            </mattextparam>
            <mattext> + </mattext>
            <mattextparam>
              <get_param ident="vard"/>
            </mattextparam>
            <mattext> - </mattext>
            <mattextparam>
              <get_param ident="varb"/>
            </mattextparam>
            <mattext> * </mattext>
            <mattextparam>
              <get_param ident="vare"/>
            </mattextparam>
            <mattext> - </mattext>
            <mattextparam>
              <get_param ident="vard"/>
            </mattextparam>
            <mattext> / </mattext>
            <mattextparam>
              <get_param ident="varb"/>
            </mattextparam>
          </material>
        </flow>
        <response_lid ident="MC">
          <render_choice shuffle="Yes">
            <response_label ident="optionA">
              <flow_mat>
                <material>
                  <mattextparam>
                    <get_param ident="optionA"/>
                  </mattextparam>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="optionB">
              <flow_mat>
                <material>
                  <mattextparam>
                    <get_param ident="optionB"/>
                  </mattextparam>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="optionC">
              <flow_mat>
                <material>
                  <mattextparam>
                    <get_param ident="optionC"/>
                  </mattextparam>
```

```
              </material>
            </flow_mat>
          </response_label>
          <response_label ident="optionD" rshuffle="No">
            <flow_mat>
              <material>
                <mattext>None of the above</mattext>
              </material>
            </flow_mat>
          </response_label>
        </render_choice>
      </response_lid>
    </flow>
  </presentation>
  <resprocessing>
    <outcomes>
      <decvar defaultval="0"/>
    </outcomes>
    <respcondition>
      <conditionvar>
        <varequal respident="MC">optionC</varequal>
      </conditionvar>
      <setvar action="Add">1</setvar>
    </respcondition>
  </resprocessing>
</item>
```

Figure 5.40 Example XML supporting MCQ using parameterised distracters.

The rendered outcome for this question is shown Figure 5.41.

```
What is the value of a, given the expression below?

     a = c + d – b * e – d / b

A) -7.5
B)  6
C)  10
D) None of the above
```

Figure 5.41 Example rendering of question demonstrating parameterised presentation elements.

Figure 5.42 completes the XML example provided in Figure 5.35, which describes a FIB question asking the candidate to complete a UNIX sed command.

```
  <item ident="External_Program_Example">
    <parameters>
      <set_param ident="oldshell">
        <enumeration>
          <literal>/bin/bash</literal>
          <literal>/bin/tcsh</literal>
          <literal>/bin/false</literal>
        </enumeration>
      </set_param>
      <set_param ident="newshell">
        <enumeration>
          <literal>/bin/bash</literal>
          <literal>/bin/tcsh</literal>
          <literal>/bin/false</literal>
        </enumeration>
      </set_param>
      <condition>
        <conditionparam>
          <paramunique>
            <get_param ident="oldshell"/>
            <get_param ident="newshell"/>
          </paramunique>
        </conditionparam>
        <iffalse>
          <recalculate/>
        </iffalse>
      </condition>
    </parameters>
    <presentation>
      <flow>
        <material>
          <mattext>Complete the Regular Expression below, such that it will replace
all occurrences of </mattext>
          <mattextparam>
            <get_param ident="oldshell"/>
          </mattextparam>
          <mattext> with </mattext>
          <mattextparam>
            <get_param ident="newshell"/>
          </mattextparam>
          <mattext> in the shell field of the /etc/passwd file?</mattext>
          <matbreak/>
          <mattext>sed -e 's/</mattext>
        </material>
        <response_str ident="FIB1">
          <render_fib fibtype="String">
            <response_label ident="A"/>
          </render_fib>
        </response_str>
        <material>
          <mattext>/</mattext>
        </material>
        <response_str ident="FIB2">
          <render_fib fibtype="String">
            <response_label ident="B"/>
          </render_fib>
        </response_str>
        <material>
          <mattext>/g' /etc/passwd</mattext>
        </material>
      </flow>
    </presentation>
    <resprocessing>
      <outcomes>
        <decvar defaultval="0"/>
      </outcomes>
      <parameters>
        <set_param ident="iscorrect">
          <program uri="regularexpression.pl" interpreter="perl" type="External">
            <get_param ident="oldshell"/>
            <get_param ident="newshell"/>
            <get_response respident="FIB1"/>
            <get_response respident="FIB2"/>
          </program>
        </set_param>
      </parameters>
```

```
      <respcondition>
        <conditionvar>
          <paramequal type="Parameter" ident="iscorrect">
            <literal/>
            <!--The program will return no output if candidate solution-->
            <!-- is correct.  Otherwise iscorrect parameter will hold  -->
            <!-- the output from the program using the candidate's     -->
            <!-- which could be used as part of the itemfeedback       -->
          </paramequal>
        </conditionvar>
        <setvar action="Add">1</setvar>
        <displayfeedback linkrefid="Correct" feedbacktype="Response"/>
      </respcondition>
      <respcondition>
        <conditionvar>
          <not>
            <paramequal type="Parameter" ident="iscorrect">
              <literal/>
              <!--The program will return no output if candidate solution-->
              <!-- is correct.  Otherwise iscorrect parameter will hold  -->
              <!-- the output from the program using the candidate's     -->
              <!-- which could be used as part of the itemfeedback       -->
            </paramequal>
          </not>
        </conditionvar>
        <setvar action="Set">0</setvar>
        <displayfeedback linkrefid="Incorrect" feedbacktype="Response"/>
        <displayfeedback linkrefid="Incorrect" feedbacktype="Solution"/>
      </respcondition>
    </resprocessing>
    <itemfeedback ident="Correct" view="Candidate">
      <material>
        <mattext>Correct</mattext>
      </material>
    </itemfeedback>
    <itemfeedback ident="Incorrect" view="Candidate">
      <material>
        <mattext>Incorrect</mattext>
      </material>
      <solution>
        <solutionmaterial>
          <flow_mat>
            <material>
              <mattext>Your solution produced the following incorrect
output:</mattext>
            </material>
          </flow_mat>
          <flow_mat>
            <material>
              <mattext><![CDATA[<pre>]]></mattext>
              <mattextparam>
                <get_param ident="iscorrect"/>
              </mattextparam>
              <mattext><![CDATA[</pre>]]></mattext>
            </material>
          </flow_mat>
        </solutionmaterial>
      </solution>
    </itemfeedback>
  </item>
```

Figure 5.42 Example QTIPA XML demonstrating mattextparam element in FIB question.

A rendered example of this question with the parameters instantiated is also illustrated in Figure 5.43.

Complete the Regular Expression below, such that it will replace all occurrences of /bin/tcsh with /bin/false in the shell field of the /etc/passwd file?

sed -e 's/<textbox>/<textbox>/g' /etc/passwd

Figure 5.43 Example rendering of QTIPA question demonstrating mattextparam element in FIB question.

Changes to the presentation component of the QTI Specification have been discussed, with the inclusion of the *mattextparam* and *matemtextparam* elements. However, there are a couple of alternate approaches available in the implementation of the rendering of parameter values.

Within the *material* element, there are quite a few sub-elements which are named so, based on the type of media they present. For example, the *mattext* element is responsible for presenting textual information to the candidate, whereas the *matimage* element is responsible for presenting images to the candidate. New elements supporting parameter values, rather than implemented as sibling elements to the existing elements (such as *mattext* for example) could be implemented as a sub-element to *mattext* or *matimage* where the media type is based on the context in which it is used (i.e. in which of the *material* sub-elements it is contained). Figure 5.44 provides an example illustrating the sub-element approach.

```
<presentation>
  <flow>
    <material>
      <mattext>Complete the Regular Expression below,</mattext>
      <mattext> such that it will replace all occurrences</mattext>
      <mattext> of <get_param ident=”oldshell”/></mattext>
      <mattext> with <get_param ident=”newshell”/></mattext>
      <mattext> in the shell field of the /etc/passwd file?</mattext>
      <matbreak/>
      <mattext>sed -e 's/</mattext>
    </material>
    <response_str ident="FIB1">
      <render_fib fibtype="String">
        <response_label ident="A"/>
      </render_fib>
    </response_str>
    <material>
      <mattext>/</mattext>
    </material>
    <response_str ident="FIB2">
      <render_fib fibtype="String">
        <response_label ident="B"/>
      </render_fib>
    </response_str>
    <material>
      <mattext>/g' /etc/passwd</mattext>
    </material>
  </flow>
</presentation>
```

Figure 5.44 Example of sub-element approach to implementing parameter presentation.

The *mattext* element would be changed to allow mixed content, meaning that PCDATA and elements (*get_param*) can be mixed together in another elements data section. This is illustrated in Figure 5.44 where the *get_param* elements are within two of the existing *mattext* elements, representing the values of oldshell and newshell. Table 5.13 shows the advantages and disadvantages to the sub-element approach.

Table 5.13 Advantages & Disadvantages of implementing parameter element as a sub-element

A parameter element implemented as a sub-element to mattext and others specific to their media type	
Advantages:	There are two elements within material for presenting textual information. One (mattext) will present the text in a normal way, whereas the second (matemtext) will present the text with some emphasis formatting (like bold for example). If the parameter presentation element is provided as a sub-element, then the sub-element in which the parameter element is placed will determine whether it is emphasised or not.
	The type of the parameter is implied. The element in which the parameter element is placed implies what the media type of the parameter element is. This simplifies things as only one parameter element is necessary which can represent any media type, depending in which material sub-element it is placed.
Disadvantages	It may be too inflexible, particularly if the parameterisation is implemented into the other media types, such as images or audio.
	Using this approach necessitates that the existing mattext and matemtext elements be changed from supporting only PCDATA to mixed content, so they may contain either character data or a parameter element. Mixed content is less desirable as it reduces the constraints that can be applied through a DTD.

Instead, Table 5.14 shows the advantages and disadvantages to the implemented sibling elements approach, using the *mattextparam* and *matemtextparam* elements.

Table 5.14 Advantages & Disadvantages of implementing parameter elements as sibling elements

Parameter elements implemented as siblings to mattext and others specific to their media type	
Advantages:	By using this approach, it makes it more flexible as to how the parameters can be introduced into the presentation component of the specification. This is because the design is not constrained by the limitations of the existing elements.
	It is not necessary to change existing elements to use mixed content.
Disadvantages:	A separate parameter presentation element needs to be introduced for each of the media types. Two elements need to be created for the text media type; one for normal text, and another for emphasised text.

The sibling approach has been implemented in providing parameter values within the presentation structure, as it is important that the changes to the specification are still backward compatible with existing QTI documents and minimal use of mixed content is preferred.

The final category to address in implementing parameterised questions into the IMS QTI is the response processing component. This is investigated in the next section.

5.6 Parameter Response Processing

The response processing component of the specification provides instruction on how to evaluate the responses provided by the candidate in answering the test item (see Section 3.3.14). Once parameterisation is implemented, variables are placed within the question stem, and this means their values are required when processing the response. The QTI Parameterisation Addendum does not specify how the instantiated

parameter values are recorded on generation of the question stem, or how they are restored for response processing. All the parameter values previously instantiated before the question was rendered must be available to the response processing structure of the specification identified via their *ident* attribute. Example parameterised questions from Figure 4.5 in Section 4.4 will be used throughout to help illustrate how the parameterisation has been implemented into the specification. Discussion will commence with the conditionvar element structure.

5.6.1 conditionvar element

To support comparison of candidate responses with parameter values, the *conditionvar* element (see Section 3.3.16) needs to be adjusted, without breaking compatibility with existing QTI documents.

Table 5.15 shows a list of elements that have been developed for use within the parameter conditions (see Section 5.4.5) that will be re-used within the *conditionvar* element structure. A brief explanation of their purpose is shown in Table 5.5.

Table 5.15 List of parameter condition elements to be re-used for response processing

paramequal	paramlt	paramgt
	paramlte	paramgte

Rather than develop new elements to support conditions on the candidate's response in conjunction with the parameter values, it is more efficient to re-use those already developed for providing conditions on the parameters themselves. Figure 5.45 shows the new structure for the *conditionvar* element including the new elements as shown in Table 5.15.

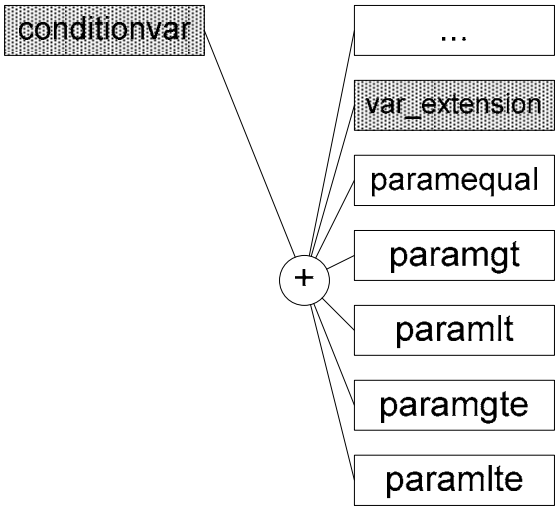


Figure 5.45 Amendments to the conditionvar element structure.

By implementing the elements as shown in Table 5.15, the specification will support response processing where comparisons can be made between 2 parameters, a parameter and a response, a parameter and a literal value, and a response and a literal value as shown in Table 5.16.

Table 5.16 Matrix identifying elements to use for comparison of parameters, responses, or literal values

Compare	Parameter	Literal Value
Parameter	paramequal paramgt paramlt paramgte paramlte	paramequal paramgt paramlt paramgte paramlte
	`<paramequal type="Parameter" ident="param1"> <get_param ident="param2"/> </paramequal>`	`<paramequal type="Parameter" ident="param1"> <literal> 5 </literal> </paramequal>`
Response	paramequal paramgt paramlt paramgte paramlte	varequal vargt varlt vargte varlte ...
	`<paramequal type="response" ident="response1"> <get_param ident="param_ans"/> </paramequal>`	`<varequal respident="option1"> 5 </varequal>`

The “param” type elements are used for all cases where a parameter value is involved, and the “var” type elements are used as before when comparing a literal value with a response. By implementing in this way, the existing specification is preserved without change, and the additional “param” type elements provide all the support for parameterisation comparison. This assures that the specification remains backward compatible.

Figure 3.18 from Section 3.3.17 illustrates the use of these conditional structures within the *respcondition* element, for performing response processing on the question illustrated in Figure 3.19 where comparison is made between the candidate response and a literal value. As can be seen, comparison between a candidate response and a literal value has remained unchanged from the method that was used by the original QTI. However, Figure 5.46 provides an example of response processing where the response from a candidate is compared with parameter values. The question to which it refers is Question 7 from Figure 4.5 in Section 4.4.

```
<item ident="FIB_Example2">
  <parameters>
    <set_param ident="area">
      <range>
        <range_min>
          <literal>10</literal>
        </range_min>
        <range_max>
          <literal>50</literal>
        </range_max>
      </range>
    </set_param>
    <set_param ident="answerlower">
      <formula prog_language="perl">
    =int(sqrt(<get_param ident="area"/> / 3.14))
  </formula>
    </set_param>
    <set_param ident="answerupper">
      <formula prog_language="perl">
    =<get_param ident="answerlower"/> + 1
  </formula>
    </set_param>
    <set_param ident="answer">
      <formula prog_language="perl">
      =sqrt(<get_param ident="area"/> / 3.14)
    </formula>
    </set_param>
  </parameters>
  <presentation>
    <flow>
      <material>
        <mattext>What is the radius of a circle if its area is given as </mattext>
        <mattextparam>
          <get_param ident="area"/>
        </mattextparam>
        <mattext>?</mattext>
      </material>
      <response_num ident="FIB1">
        <render_fib fibtype="Decimal">
          <response_label ident="A"/>
        </render_fib>
      </response_num>
    </flow>
  </presentation>
  <resprocessing>
    <outcomes>
      <decvar defaultval="0"/>
    </outcomes>
    <respcondition>
      <conditionvar>
        <and>
          <paramlte type="Response" ident="FIB1">
            <get_param ident="answerupper"/>
          </paramlte>
          <paramgte type="Response" ident="FIB1">
            <get_param ident="answerlower"/>
          </paramgte>
        </and>
      </conditionvar>
      <setvar action="Add">1</setvar>
      <displayfeedback linkrefid="Correct" feedbacktype="Response"/>
      <displayfeedback linkrefid="Correct" feedbacktype="Solution"/>
    </respcondition>
    <respcondition>
      <conditionvar>
        <not>
          <and>
            <paramlte type="Response" ident="FIB1">
              <get_param ident="answerupper"/>
            </paramlte>
            <paramgte type="Response" ident="FIB1">
              <get_param ident="answerlower"/>
            </paramgte>
          </and>
        </not>
      </conditionvar>
```

```
        <displayfeedback linkrefid="Incorrect" feedbacktype="Response"/>
        <displayfeedback linkrefid="Incorrect" feedbacktype="Solution"/>
      </respcondition>
    </resprocessing>
    <itemfeedback ident="Correct" view="All">
      <material>
        <mattext>Correct</mattext>
      </material>
      <solution>
        <solutionmaterial>
          <material>
            <mattextparam>
              <get_param ident="answer"/>
            </mattextparam>
          </material>
        </solutionmaterial>
      </solution>
    </itemfeedback>
    <itemfeedback ident="Incorrect" view="All">
      <material>
        <mattext>Incorrect</mattext>
      </material>
      <solution>
        <solutionmaterial>
          <material>
            <mattext>The correct solution is: </mattext>
            <mattextparam>
              <get_param ident="answer"/>
            </mattextparam>
            <mattext>.  However a solution between </mattext>
            <mattextparam>
              <get_param ident="answerlower"/>
            </mattextparam>
            <mattext> and </mattext>
            <mattextparam>
              <get_param ident="answerupper"/>
            </mattextparam>
            <mattext> would have been acceptable</mattext>
          </material>
        </solutionmaterial>
      </solution>
    </itemfeedback>
  </item>
```

Figure 5.46 Example response processing question comparing the candidate response with a parameter value.

The candidate must provide a numeric value, which represents the radius of a circle. The value of the response provided by the candidate represented by the ident “FIB1” is compared with the values of the parameters represented by both “answerupper” and “answerlower”. The logic of the response processing is to compare the numeric value provided by the candidate to see whether it is lower than one parameter value, and higher than another, thus ensuring the answer provided is within a suitable range to be considered correct. The *displayfeedback* within that particular *respcondition* structure indicates that the *itemfeedback* material identified by the *ident* attribute value “Correct” should be displayed if the *conditionvar* returns true. Another *respcondition*

structure is provided which negates the result of the comparison with a *not* element. It will detect when the candidate's solution does not fall within the acceptable range, and likewise will allocate an *itemfeedback ident* through the *displayfeedback* element, except the *ident* value shall be set to "Incorrect". In this case, the *itemfeedback* provides a response to the candidate indicating their solution was incorrect.

What about the case of a parameter being compared with a literal (static) value? A FIB example was provided in Section 5.4.3 through Figure 5.11 in which the candidate was required to complete a UNIX sed command, such that it will perform a find and replace of a particular file's contents. The Perl script (Figure 5.36) which the *program* element calls will take both the instantiated parameter values from the question, plus the candidate's solution. It will then use the instantiated parameter values to generate a correct solution and execute it, storing the resulting output in a Perl variable, before then executing the solution provided by the candidate and storing likewise. With the two Perl variables, it will perform a comparison and if they are identical, it will return nothing from the script. If they are not identical, it will return the output from the candidate's solution so it can be presented back to the candidate as feedback. The response processing of this question is demonstrated in Figure 5.35 in which the parameter holding the output of the candidate's solution is compared with an empty *literal* element, which is testing for an empty string. If this returns true, then the candidate's solution was correct. Next is an example of QTIPA XML, representing the case where two parameters are compared in processing the response from a candidate.

Figure 5.47 represents an alternate approach supported by the QTIPA to that used in the previous example (Figure 5.35) in performing the response processing of the question illustrated in Figure 5.11.

```
...
    <resprocessing>
      <parameters>
        <set_param ident="candidate_iscorrect">
          <program uri="/usr/local/bin/regularexpression.pl" interpreter="perl"
type="External">
            <get_param ident="oldshell"/>
            <get_param ident="newshell"/>
          </program>
        </set_param>
        <set_param ident=”solution”>
          <program uri="/usr/local/bin/regularexpression.pl" interpreter="perl"
type="External">
            <get_response respident="candidateresponse1"/>
            <get_response respident="candidateresponse2"/>
          </program>
        </set_param>
      </parameters>
      <respcondition>
        <conditionvar>
          <paramequal type="Parameter" ident="solution">
            <get_param ident=”candidate_iscorrect”/>
          </paramequal>
        </conditionvar>
        <setvar action="Add">1</setvar>
      </respcondition>
    </resprocessing>
...
```

Figure 5.47 Example response processing question comparing two parameter values.

Instead of the Perl script represented in Figure 5.36 returning an empty string if the output of the candidate’s sed command matches the output of the correct command, the Perl script is executed twice once providing the candidate’s solution and returning with the output from that solution, the other time with the correct solution and its output. Then, the comparison of the output between each is made through the *paramequal* element structure. If they do not match, then the candidate’s solution is still incorrect, and the output from the candidate’s solution is still available within a parameter variable, which can be used in *itemfeedback* if necessary.

As demonstrated with the previous example (Figure 5.47), to support the ability to generate and instantiate parameter values within the response processing component

of the specification, the parameters element structure must also be implemented into the resprocessing element structure. This implementation is reported in the next section.

5.6.2 Re-use of Parameters Element within Response Processing

As discussed in Section 5.4, the QTIPA Specification will allow the *parameters* element to be placed both under the *item* element, and under the *resprocessing* element. By implementing the *parameters* element in both places, it allows parameters to be instantiated when the question is initially parsed before the question stem is processed for presentation to the candidate. It also allows parameters to be instantiated when the candidate has submitted their response. This provides the benefit of being able to calculate the correct answer after the candidate has submitted their solution. This is significant, because some questions (as illustrated in Figure 5.47) may need to be implemented in such a way that the candidate's response forms part of the processing through the parameters elements to determine a marking outcome.

An example (Figure 5.47) was provided in the previous section (Section 5.6.1) where the candidate's responses are passed to a program, which uses them to determine if they are correct. To support this functionality it is necessary to include the ability to instantiate parameter values within the resprocessing element structure. Furthermore, it is necessary for the *formula* and *program* elements to be able to use responses provided by the candidate (see sections 5.4.4 and 5.4.6). Therefore, in the spirit of re-use, the *parameters* element structure will also be introduced into the *resprocessing* element as a child. The *get_response* element has been included to support the ability

access the candidate's responses within the *formula* and *program* elements. Refer to Section 5.4.6 which demonstrates implementation of the *get_response* element. The QTIPA version of the *resprocessing* element is illustrated in Figure 5.3 from Section 5.4.

By re-introducing the parameters elements into another section of the QTI item, certain issues need to be considered. For example where possible, it would be beneficial to include all parameter processing in the parameters element structure provided as a child to *item*, and limit those to be included within the *resprocessing* child *parameters* element. Any question that utilises the parameters structure within *resprocessing* is precluded from having the question converted into a discrete set of multiple questions, representing all possible outcomes of the question (refer to the "Future Work" Section of Chapter 7 for further explanation of this concept). Another problem to consider is the namespace of the two different parameters structures.

The QTIPA Specification mandates that all the parameters instantiated at presentation time of the item should be restored and completely available at response processing time. If there is a second parameters declaration that is instantiated at the response processing time, there are two sets of parameter values. Essentially, the second set should directly manipulate the restored parameter namespace, thus ensuring there is always only one namespace for parameters. For example, Figure 5.42 in Section 5.5 provides an example QTIPA XML implementation with two parameters declarations implemented. Referring to this figure, if there was a parameter assigned the name "iscorrect" from the parameters structure declared at presentation time of the question, then its value would be overwritten by the value instantiated in the second parameters

structure under *resprocessing*. In other words, there should be only one parameter namespace and the parameters instantiated before presentation are loaded first.

5.7 Global design considerations

This section will discuss some of the high level design considerations of implementing parameterisation, and how it affected the implemented XML structure. To best integrate support for parameterised questions, it was necessary to consider the development approach of the IMS QTI authors, and maintain consistency as such. This section will discuss many such considerations, which affected how this added support was implemented.

The existing QTI Specification supports proprietary extensions within certain element structures. Parameters could have been implemented via these proprietary extensions where a new namespace is introduced to the specification. This would not disturb the existing specification and provide a modular approach to implementing parameterisation. There are some problems associated with this approach. Anything that is implemented through the extension elements would not be interoperable with other IMS QTI compliant systems. This negates many of the benefits associated with an industry specification. Furthermore, IMS have indicated that it is likely that modification will be required to questions which make use of QTI Version 1 extensions, to be compliant with their future version 2 release (IMS 2003d). Therefore, implementation of parameters has been provided directly within the core specification, such that the full benefits of an interoperable specification are realised.

Although not necessary to demonstrate the concept of this research, it is recognised that to maintain consistency with the existing QTI Specification, it would be appropriate to also include proprietary extension elements for both parameter declaration and response processing. For example, a new element *set_param_extension* could be included within the *set_param* element structure, and would allow the setting of a parameter variable via some alternate proprietary means. Similarly, a new element *param_extension* could also be included within the *conditionparam* element structure such that proprietary methods can be implemented in the comparison of parameter values.

The QTI XML Specification has been designed such to allow linking of data components externally via the application, rather than internally using XML syntax. The typical method to identify a particular data component is to refer to it via a uniquely assigned attribute value using common attribute names such as *ident*. Other elements within the specification refer to one another via these *ident* attributes, rather than using the IDREF functionality provided by XML. The reason for this may be due to many existing systems already having their own internal representation of questions and tests, and simply use the QTI Specification as an interchange format. By designing the QTI in this way, it provides total freedom to the developer to implement the specification in whatever way they choose. Therefore, to keep in-line with the existing QTI philosophy, the QTIPA has not incorporated the use of IDREF attributes and uses similarly named attributes for linking values.

5.8 Conclusions

Chapter 5 has provided an in depth discussion of the new QTIPA Specification, and background into the decisions surrounding the details of its implementation. Table 5.17 provides a summary of the newly implemented XML elements, along with the existing elements, which have been altered or re-used to support parameterised questions.

Table 5.17 Summary of new and amended XML elements in QTIPA

New Elements	Amended Elements
condition	and
conditionparam	not
enumeration	item
formula	or
get_param	resprocessing
get_response	material
iffalse	conditionvar
iftrue	
literal	
matemtextparam	
mattextparam	
paramequal	
parameters	
paramgt	
paramgte	
paramlt	
paramlte	
paramunique	
program	
range	
range_max	
range_min	
recalculate	
set_param	

Complete examples of the QTIPA XML supporting each of the questions illustrated in Figure 4.5 of Section 4.4 are available on the attached Supplementary CDROM. Refer to Appendix C for further information regarding the Supplementary CDROM. The next chapter discusses a prototype quiz system, which demonstrates the implementation of the parameterised quiz questions as per Figure 4.5.

6 Implementing a QTIPA Quiz System

To demonstrate the successful application of the QTIPA Specification, a prototype quiz system "QTIPA Quiz System" has been developed. This chapter will cover topics related to the QTIPA Quiz System, such as:

- the supported elements (Section 6.1),
- an overview of the software design including programming language choice for development (Section 6.2),
- rendered screen shots with XML excerpts supporting the rendered output (Section 6.3).

6.1 Element Support

QTIPA Quiz has been designed to support sufficiently enough of the existing IMS QTI Specification's item element structure, plus all of the QTIPA enhancements to demonstrate the successful implementation of parameterised quiz questions. Table 6.1 shows two alphabetical lists of the original IMS QTI elements and the enhanced QTIPA elements that are supported by QTIPA Quiz.

Table 6.1 QTIPA Quiz Supported QTI & QTIPA Elements

Original IMS QTI Elements	QTIPA Enhanced Elements
and	condition
conditionvar	conditionparam
decvar	enumeration
displayfeedback	formula
flow	get_param
flow_label	get_response
flow_mat	iffalse
hint	iftrue
hintmaterial	literal
itemfeedback	matemtextparam
matbreak	mattextparam
matemtext	paramequal
material	parameters
mattext	paramgt
not	paramgte
or	paramlt
outcomes	paramlte
render_choice	paramunique
render_fib	program
respcondition	range
response_label	range_max
response_lid	range_min
response_num	recalculate
response_str	set_param
setvar	
solution	
solutionmaterial	
unanswered	
varequal	
vargt	
vargte	
varlt	
varlte	

The focus of this research is aimed at parameterisation of the two most contrasting question types: fill in the blank and multiple choice questions. Therefore, elements supported from the *item* element structure of the original IMS QTI Specification were selected such that QTIPA Quiz can support:

- FIB, MCQ, or a combination of both question rendering types,

- all of the QTIPA parameterisation elements,
- all questions (including those with parameterisation) such that:
    - they can be presented to the candidate,
    - the candidate response(s) can be processed,
    - feedback can be presented to the candidate based on the response processing.

The next section provides an overview of the QTIPA Quiz code design. The interested reader is referred to the attached Supplementary CDROM, which provides a complete listing of the QTIPA Quiz System source code. The contents of the Supplementary CDROM are detailed in Appendix C.

6.2 Overview of QTIPA Quiz

QTIPA Quiz has been implemented as a collection of Perl (Practical Extraction and Reporting Language) object-oriented classes, which are driven by two Perl CGI (Common Gateway Interface) scripts. Other choices for programming languages included Java and C, however Perl was selected for the following reasons (Muldner and Currie 1999):

- quicker development time than C and Java,
- quicker testing time than C and Java,
- platform independent (portable interpreter),
- excellent text processing support,
- flexible programming syntax,
- supports a variety of interfaces such DBI (Database Interface), RPC (Remote Procedure Calls), and Agents.

The SCORM Sharable Content Objects (SCOs) are implemented into SCORM compliant LMS via an API adapter using the javascript language (Rustici 2004). This means that active client processing is performed in the candidate's browser. SCORM repackage the IMS QTI Specification into their suite, however it would not be advisable to implement the parameterisation addendum using Javascript due to potential security issues (Hay and Nance 2002). Furthermore, the author is quite familiar with web based development using the Perl language which ensured development time of the prototype was not stalled as a result of learning a new programming environment. The system was implemented through the Apache Web Server Software, on a Fedora Core Linux Operating System. Due to the portability of the software used for this project, there should be no reason to prevent the system from operating on other web servers or operating systems, thus maintaining the portability aspect of the project.

6.2.1 Perl XML Parser

There are many Application Programming Interfaces (APIs) available in Perl for accessing and manipulating XML structures. QTIPA Quiz makes use of the XML::DOM module originally by Enno Derksen, and as of this writing maintained by T. J. Mather. This module provides the standard DOM (Document Object Model) API for accessing and manipulating XML structures in memory. Given that XML is a recursive object based data structure, QTIPA Quiz has been implemented to use DOM to parse the XML and generate a hierarchy of Perl DOM objects, matching the structure of the item XML. After parsing, the QTIPA Quiz system will recursively walk the DOM structure passing the element objects and their attributes to like named methods, processing the item XML for the quiz. Further information on the DOM

walker implementation is included with discussion of the *Item* class in the next section.

6.2.2 QTIPA Quiz Perl Classes

The QTIPA system is comprised of a collection of Perl object-oriented classes, which drive each of the three categories of the parameterisation framework (see Section 5.2), being Parameter Declaration, Presentation and Response Processing. Table 6.2 provides a list of all the classes as used by QTIPA Quiz, which categories the class supports, and a brief description of its purpose.

The *Item* class has a dual purpose. *Item* is inherited by other classes using the traditional object-oriented programming model. This allows methods to be shared between classes without duplication. For example, the *and*, *or*, and *not* elements are shared between the parameters element structure, and also the resprocessing element structure in the QTIPA Specification. In the QTIPA Quiz System, they are implemented as the private methods *_and*, *_or*, and *_not* in the *Parameters* and *Resprocessing* classes. Rather than implementing two copies of the same code into two different classes, the code for all three of these private methods is stored in the *Item* class, and both the *Parameters* and *Resprocessing* classes inherit those methods.

Table 6.2 QTIPA Quiz Perl Classes

Class Name	Category	Description
Item	Parameter Declaration Presentation Response Processing	This class is inherited by the other classes and is used to process all aspects of the question item.
ItemFeedback	Presentation	Each instance of this class is used to parse and declare possible feedback material to the candidate.
ItemInstance	Parameter Declaration Presentation	This class is used to save state information for an item when it is presented, such as its parameter values, and the ResponseLids mapping, for retrieval when candidate responses are received.
Parameters	Parameter Declaration	This class is used to parse, generate, store, and access parameter values for an item.
Presentation	Presentation	This class is used to parse, and generate HTML from the presentation component of the item, for presentation to candidate.
PresentationFeedback1	Presentation	An instance of this class is passed to the item class to present one particular format for feedback to the candidate after the response processing.
ResponseLids	Presentation Response Processing	This class stores the mapping between the original response_label ident, and the label value generated into the presentation HTML (from presentation class).
Responses	Response Processing	This class stores the responses provided by the candidate for a particular item.
Resprocessing	Response Processing	This class is used to parse, and process the candidate's responses using the Responses object.
ScoringVar	Response Processing	Each instance of this class will track the scoring variables associated with an item.

As previously discussed, the DOM API manipulates the XML structures within the Perl classes, and a recursive walking algorithm processes the DOM objects. To implement this, an associative array declared in the *Item* class maps the elements names to corresponding object methods names. Another shared method *_callMethod*, provided in the *Item* class, when passed a DOM object of type "ELEMENT_NODE" (an XML element object) will look up the associative array and call the matching method name, passing the DOM object element for processing. An excerpt from the declaration of the associative array, and the source for the *_callMethod* method are illustrated in Figure 6.1.

```
my %subs = (

#Parameters
"parameters" => "_parameters",
"set_param" => "_set_param",
"condition" => "_condition",
"conditionparam" => "_conditions",
"paramequal" => "_paramMatch",
"paramgt" => "_paramMatch",
"paramgte" => "_paramMatch",
"paramlt" => "_paramMatch",
"paramlte" => "_paramMatch",
"paramunique" => "_paramunique",
"recalculate" => "_recalculate",
"iftrue" => "_ifelement",
"iffalse" => "_ifelement",
"enumeration" => "_enumeration",
...
) ;
...
sub _callMethod
{
  my $self = shift ;

  my $dom = shift ;

confess "undefined object to callmethod" unless(defined $dom) ;
confess "object is not of type element" unless($dom->getNodeTypeName
         eq "ELEMENT_NODE") ;

  my $tagName = $dom->getTagName ;

  my $method = $subs{"$tagName"}
      or confess "Invalid element or element not supported: $tagName" ;

  #Call the method passing all remaining parameters as well
  return $self->$method ($dom,@_) ;
}
```

Figure 6.1 Associate array for elements to method names and source for _callMethod method (Item.pm).

Figure 6.2 lists the source code of the *_and* method, which is designed to perform a logical and of the return values for each of its child DOM element objects, as per the *and* element of the QTIPA Specification. The *_and* method gathers a list of all the child element DOM objects, then iterates through each child DOM object calling the *_callMethod* method passing the object, and saving the return value. If all the return values are true, then the *_and* method returns true. Otherwise, the *_and* method returns false. All methods that manipulate a DOM element object that potentially has children must pass each of its children to the *_callMethod* method to have them processed.

```
sub _and
{
  my $self = shift ;

  my $andDom = shift ;

  my $retval = 1 ;

  my $children = $self->_getChildElementsNodeList($andDom) ;

  my $n = $children->getLength ;
  my $i = 0;
  do
  {
    my $result = $self->_callMethod($children->item($i++)) ;

    $retval = ($retval && $result) ;
  } while ($retval && $i < $n) ;

  return $retval ;
}
```

Figure 6.2 source for _and method (Item.pm)

Figure 6.3 reveals a small excerpt of the response processing of a question, which makes use of the *and* element, and thus the *_and* method. So if the method representing the *paramlte* element returns true (where the FIB1 response is less than or equal to the parameter value answerupper) and the *paramgte* method also returns true, then the *_and* method shall return true to its parent calling method. In this case it is *conditionvar*.

```
<conditionvar>
  <and>
    <paramlte type="Response" ident="FIB1">
      <get_param ident="answerupper"/>
    </paramlte>
    <paramgte type="Response" ident="FIB1">
      <get_param ident="answerlower"/>
    </paramgte>
  </and>
</conditionvar>
```

Figure 6.3 Example QTIPA XML performing logical "and" of candidate responses.

Implementation of the *_callMethod* method reduces the complexity of implementing further elements into the system. Only a new method needs to be developed and an entry in the associative array to provide additional support. This approach does incur overheads in stack processing, due to its recursive properties. Given the processing performance of computer hardware, the resulting latency observed does not prevent acceptable system response. The *Item* class does have other purposes and is not just limited to being a container for shared methods.

Instances of the *Item* class when provided an XML DOM object representing an entire item element structure are also responsible for providing public methods, to drive all the processes associated with a quiz item. When the *Item* constructor is called, instances of many of the other classes are created within the *Item* object to perform the work of the quiz system, as a black box. This work is performed by calling the public methods of the *Item* class, which can:

- generate parameter values and generate presentation HTML code,
- retrieve presentation HTML code,
- retrieve state information such as parameter values such that it can be stored for when the candidate's responses are to be processed,
- pass in responses from the candidate, to perform response processing,

- based on outcomes of response processing, retrieve the scoring variable values and the candidate feedback.

Other less obvious classes within the system as described in Table 6.2 include *ItemInstance*, *PresentationFeedback1*, and *ResponseLids*. For the remaining classes, the interested reader is referred to the attached Supplementary CDROM, which provides a complete listing of the QTIPA Quiz System source code. The contents of the Supplementary CDROM are detailed in Appendix C. When processing MCQs, the *ResponseLids* object is used by the *Presentation* object. The *Presentation* object will use the *ident* attribute of the *response_lid* element as the name attribute of all the corresponding HTML INPUT tags for presentation of each of the options to the candidate. It will then use the *ident* attribute of the *response_label* elements, which are children to the *response_lid* element to set the value attribute of each of the INPUT tags, uniquely identifying the chosen response from the candidate. In some circumstances, the same *ident* value for the *response_lid* element will always identify the key to the question. Therefore, to prevent students from analysing the HTML source of multiple instances of the same question, the *Presentation* object will create a random value and associate it with the *ident* attribute value of each of the *response_lid* elements, and only present the randomised value in the HTML. Therefore, no consistencies of the HTML source occur between sittings of the same question. This is illustrated in Figure 6.4, which demonstrates the use of random values for INPUT tags. The XML representing this output is listed in Figure 5.40 of Section 5.5.

```
...
<input type="RADIO" name="RESPONSES_Complex_Parameter_Conditions_Example|MC"
value="eAYFPFqT">5</input>
</p>
<p>
<input type="RADIO" name="RESPONSES_Complex_Parameter_Conditions_Example|MC"
value="lDXKXMb5">21</input>
</p>
<p>
<input type="RADIO" name="RESPONSES_Complex_Parameter_Conditions_Example|MC"
value="nuKRgzh1">-9</input>
</p>
<p>
<input type="RADIO" name="RESPONSES_Complex_Parameter_Conditions_Example|MC"
value="RzCgl4hW">None of the above
</input>
</p>
```

Figure 6.4 Sample HTML output demonstrating use of randomised value attributes for INPUT tag.

The mapping of the original and randomised identifiers is stored along with the instantiated parameter values of the question in the *ItemInstance* object. The *ItemInstance* is passed the *Parameters* object with the parameter values assigned to the question and the *ResponseLids* object with the mapping of original to randomised *response_label* values. *ItemInstance* can then generate custom XML to store the information away. Figure 6.5 illustrates the XML generated by the *ItemInstance* object for storing the parameter values and *response_lid* mappings for the question represented by the HTML in Figure 6.4. The structure of the XML for storing the parameter values is the same as is used for representing the parameter declaration in the QTIPA Specification. The *response_label ident* mappings use a simple custom structure. This information could be stored in whatever way the implementer of the specification chooses, although using XML is flexible and re-uses some existing code. When the candidate's response is processed, the *ItemInstance* object will retrieve the stored XML, so that the *ResponseLids* and *Parameters* objects can load back their previously stored values.

```
<iteminstance>
  <parameters>
    <set_param ident="correct">
      <literal>optionC</literal>
    </set_param>
    <set_param ident="optionB">
      <literal>5</literal>
    </set_param>
    <set_param ident="optionA">
      <literal>-9</literal>
    </set_param>
    <set_param ident="varc">
      <literal>4</literal>
    </set_param>
    <set_param ident="vare">
      <literal>-2</literal>
    </set_param>
    <set_param ident="varb">
      <literal>4</literal>
    </set_param>
    <set_param ident="correctFeedback">
      <literal>21</literal>
    </set_param>
    <set_param ident="randomNOTA">
      <literal>80</literal>
    </set_param>
    <set_param ident="optionC">
      <literal>21</literal>
    </set_param>
    <set_param ident="vard">
      <literal>12</literal>
    </set_param>
  </parameters>
  <responselids>
    <responselid ident="MC">
      <labelvalue ident="RzCgl4hW"><literal>optionD</literal></labelvalue>
      <labelvalue ident="eAYFPFqT"><literal>optionB</literal></labelvalue>
      <labelvalue ident="nuKRgzh1"><literal>optionA</literal></labelvalue>
      <labelvalue ident="lDXKXMb5"><literal>optionC</literal></labelvalue>
    </responselid>
  </responselids>
</iteminstance>
```

Figure 6.5 XML generated by ItemInstance object to save parameter values and response_label ident mappings.

The *presentationFeedback1* class is one possible class that can be used to process and generate HTML for the candidate's feedback, after response processing. When performing the response processing through the *Item* object, it is possible to pass in a *PresentationFeedback* type object which the *Item* object will use to generate the HTML for presenting feedback to the candidate. By creating different *PresentationFeedback* classes, which implement the required methods, custom feedback HTML can be created that is abstracted from the implementation of the *Item* class itself. Refer to Figure 6.8 and Figure 6.12 from Section 6.3 for sample output from the *PresentationFeedback1* class. The only methods that must be publicly

available through the *PresentationFeedback* type classes are: *add* (to add responses to the object from the candidate), *parse* (to generate the HTML to present back to the candidate), and *getHtml* (to retrieve the HTML source from the object).

Now that the QTIPA Quiz system has been examined, the following section will provide some sample output from the system and its matching XML source.

6.3 Example QTIPA Quiz Rendered Questions

The QTIPA Quiz system is capable of supporting both the FIB and MCQ parameterised question types. Therefore, rendering examples and XML source of each type are provided to demonstrate the working of the QTIPA Quiz system. A demonstration of the QTIPA Quiz System is available from http://cq-pan.cqu.edu.au/qtipa. This includes each of the ten parameterised questions as illustrated in Figure 4.5 of Section 4.4.

Figure 6.6 shows the rendered output as seen by the candidate when attempting Question 9 from Figure 4.5 of Section 4.4.

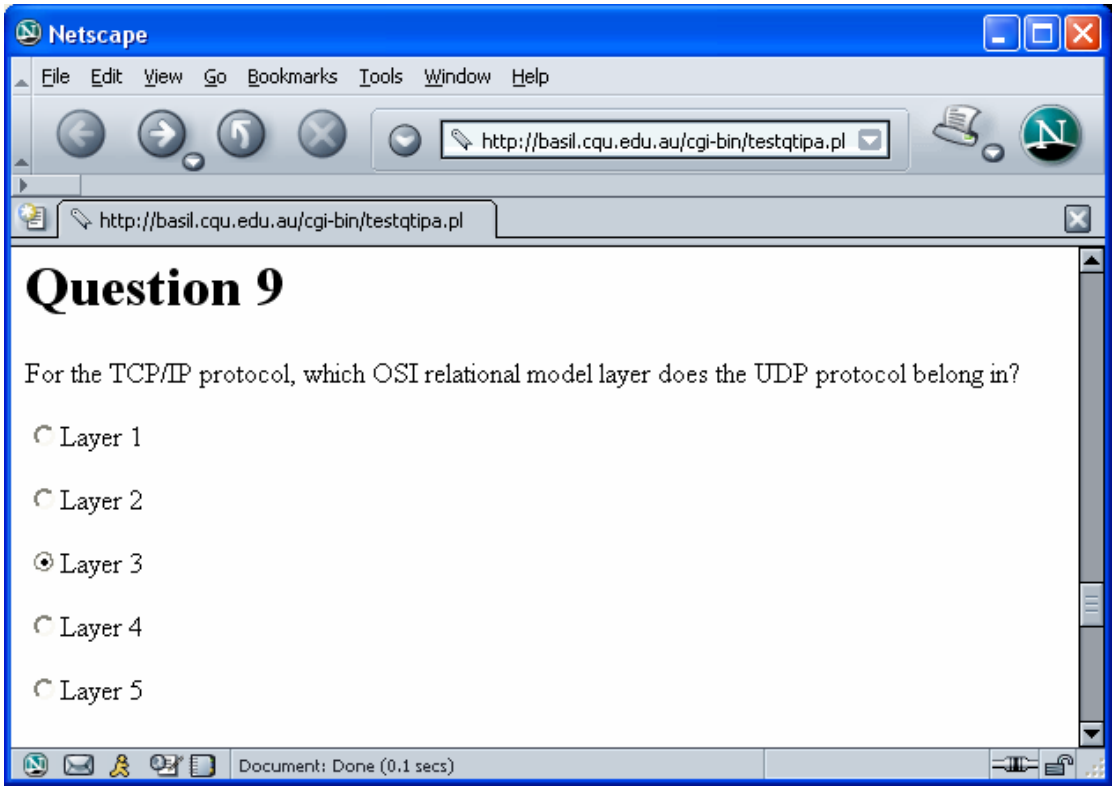


Figure 6.6 Example rendered MCQ produced by QTIPA Quiz System.

The XML, which drives the parameter instantiation and question presentation, is shown in Figure 6.7.

```
  <item ident="Static_MCQ_Example">
    <parameters>
      <set_param ident="protocol">
        <enumeration>
          <literal>TCP</literal>
          <literal>UDP</literal>
          <literal>IP</literal>
        </enumeration>
      </set_param>
      <condition>
        <conditionparam>
          <or>
            <paramequal type="Parameter" ident="protocol">
              <literal>TCP</literal>
            </paramequal>
            <paramequal type="Parameter" ident="protocol">
              <literal>UDP</literal>
            </paramequal>
          </or>
        </conditionparam>
        <iftrue>
          <set_param ident="answer">
            <literal>D</literal>
          </set_param>
          <set_param ident="answerDisplay">
            <literal>Layer 4</literal>
          </set_param>
        </iftrue>
        <iffalse>
          <set_param ident="answer">
            <literal>C</literal>
          </set_param>
          <set_param ident="answerDisplay">
            <literal>Layer 3</literal>
          </set_param>
        </iffalse>
      </condition>
    </parameters>
    <presentation>
      <flow>
        <material>
          <mattext>For the TCP/IP protocol, in which OSI relational model layer does
the </mattext>
          <mattextparam>
            <get_param ident="protocol"/>
          </mattextparam>
          <mattext> protocol belong?</mattext>
        </material>
        <response_lid ident="MC1">
          <render_choice shuffle="No">
            <response_label ident="A">
              <flow_mat>
                <material>
                  <mattext>Layer 1</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="B">
              <flow_mat>
                <material>
                  <mattext>Layer 2</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="C">
              <flow_mat>
                <material>
                  <mattext>Layer 3</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="D">
              <flow_mat>
                <material>
                  <mattext>Layer 4</mattext>
                </material>
```

```
            </flow_mat>
          </response_label>
          <response_label ident="E">
            <flow_mat>
              <material>
                <mattext>Layer 5</mattext>
              </material>
            </flow_mat>
          </response_label>
        </render_choice>
      </response_lid>
    </flow>
  </presentation>
```

Figure 6.7 QTIPA XML supporting parameter instantiation and presentation for MCQ.

If the candidate selected the response “Layer 3”, Figure 6.8 shows the rendering of the resulting response feedback as generated by the QTIPA Quiz System. The question is presented again with the same formatting as used when the question was initially shown, except the selected response from the candidate is red and underlined. The table following provides the actual feedback to the candidate. In this case, it indicates that the candidate’s solution was “Incorrect” and that the solution for the question was in fact “Layer 4”. It also provides a score for the question, which is zero.

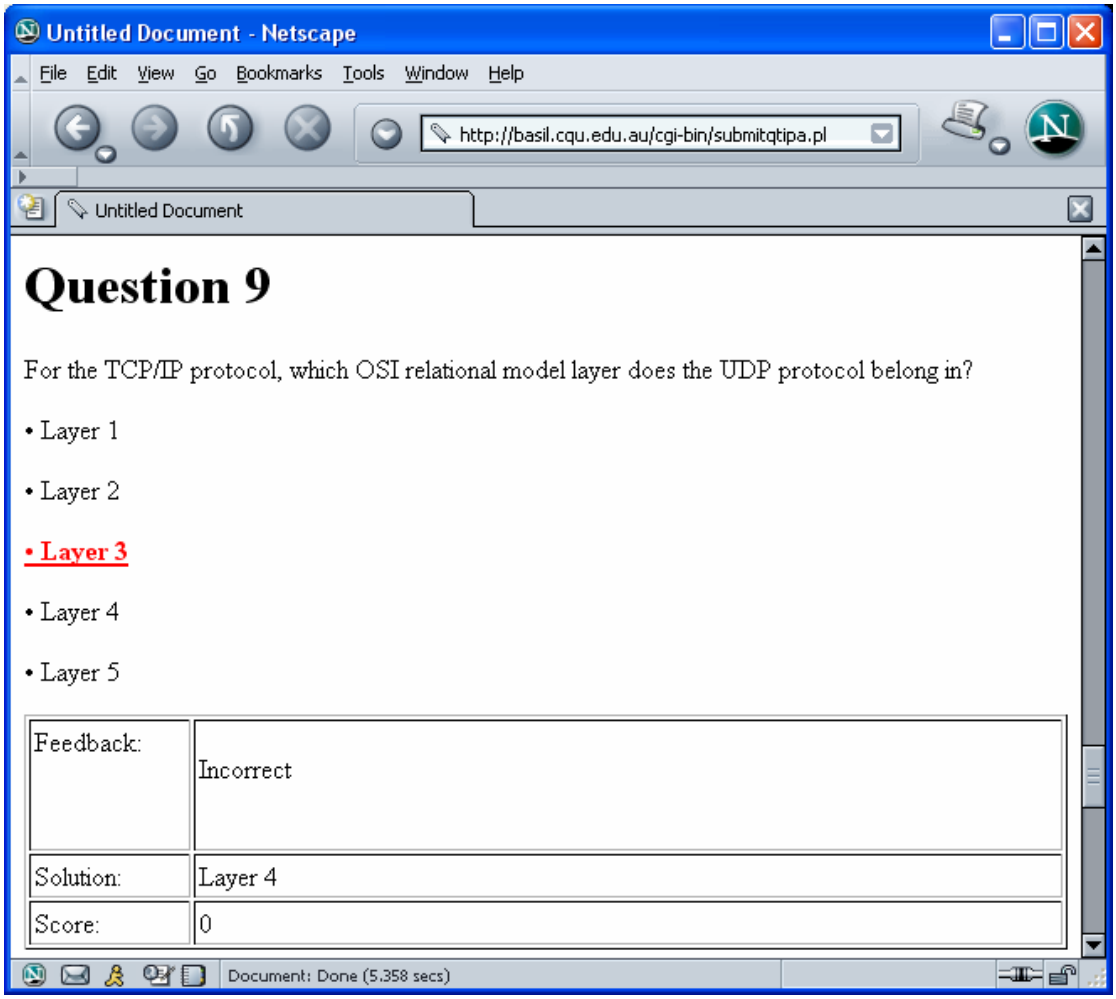


Figure 6.8 Example rendered candidate response feedback for MCQ produced by QTIPA Quiz System.

The XML, which drives the response processing and feedback presentation, is shown in Figure 6.9.

```
<resprocessing>
  <outcomes>
    <decvar defaultval="0"/>
  </outcomes>
  <respcondition>
    <conditionvar>
      <paramequal type="Response" ident="MC1">
        <get_param ident="answer"/>
      </paramequal>
    </conditionvar>
    <setvar action="Add">1</setvar>
    <displayfeedback linkrefid="Correct" feedbacktype="Response"/>
  </respcondition>
  <respcondition>
    <conditionvar>
      <not>
        <paramequal type="Response" ident="MC1">
          <get_param ident="answer"/>
        </paramequal>
      </not>
    </conditionvar>
    <displayfeedback linkrefid="Incorrect" feedbacktype="Response"/>
    <displayfeedback linkrefid="Incorrect" feedbacktype="Solution"/>
  </respcondition>
</resprocessing>
<itemfeedback view="All" ident="Correct">
  <flow_mat>
    <material>
      <mattext>Correct</mattext>
    </material>
  </flow_mat>
  <solution>
    <solutionmaterial>
      <material>
        <mattextparam><get_param ident="answerDisplay"/></mattextparam>
      </material>
    </solutionmaterial>
  </solution>
</itemfeedback>
<itemfeedback view="All" ident="Incorrect">
  <flow_mat>
    <material>
      <mattext>Incorrect</mattext>
    </material>
  </flow_mat>
  <solution>
    <solutionmaterial>
      <material>
        <mattextparam><get_param ident="answerDisplay"/></mattextparam>
      </material>
    </solutionmaterial>
  </solution>
</itemfeedback>
```

Figure 6.9 QTIPA XML supporting response processing and item feedback for MCQ.

Figure 6.10 shows the rendered output as seen by the candidate when attempting Question 7 from Figure 4.5 of Section 4.4.

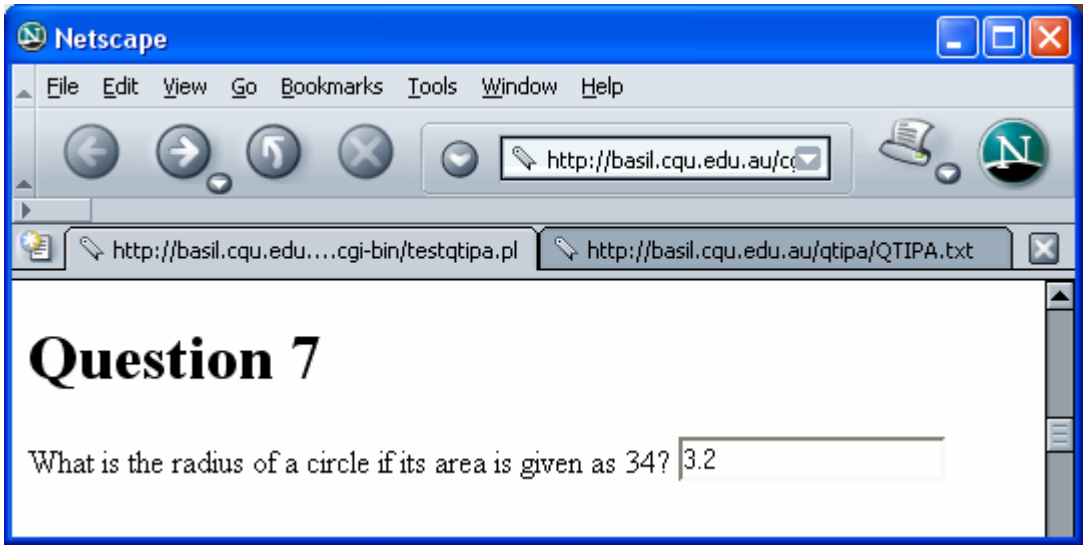


Figure 6.10 Example rendered FIB Question produced by QTIPA Quiz System.

The XML, which drives the parameter instantiation and question presentation, is shown in Figure 6.11.

```
<parameters>
  <set_param ident="area">
    <range>
      <range_min>
        <literal>10</literal>
      </range_min>
      <range_max>
        <literal>50</literal>
      </range_max>
    </range>
  </set_param>
  <set_param ident="answerlower">
    <formula prog_language="perl">
  =int(sqrt(<get_param ident="area"/> / 3.14))
</formula>
  </set_param>
  <set_param ident="answerupper">
    <formula prog_language="perl">
  =<get_param ident="answerlower"/> + 1
</formula>
  </set_param>
<set_param ident="answer">
  <formula prog_language="perl">
    =sqrt(<get_param ident="area"/> / 3.14)
  </formula>
</set_param>
</parameters>
<presentation>
  <flow>
    <material>
      <mattext>What is the radius of a circle if its area is given as </mattext>
      <mattextparam>
        <get_param ident="area"/>
      </mattextparam>
      <mattext>?</mattext>
    </material>
    <response_num ident="FIB1">
      <render_fib fibtype="Decimal">
        <response_label ident="A"/>
      </render_fib>
    </response_num>
  </flow>
</presentation>
```

Figure 6.11 QTIPA XML supporting parameter instantiation and presentation of FIB Question.

After the candidate submits their response to the question, a feedback screen is returned where the question is presented again except that the candidate's response is red and underlined, as was illustrated with the previous MCQ example. For the FIB question, this is illustrated in Figure 6.12. Once again, feedback is provided and in this example, the solution provided was correct within the precision boundaries specified within the response processing (refer to Figure 6.13), based on the parameter names "answerlower" and "answerupper" (refer to Figure 6.11). The system calculated solution is provided along with the score for this question. Had the

candidate's solution been incorrect, the feedback would indicate so, and the score given as zero.

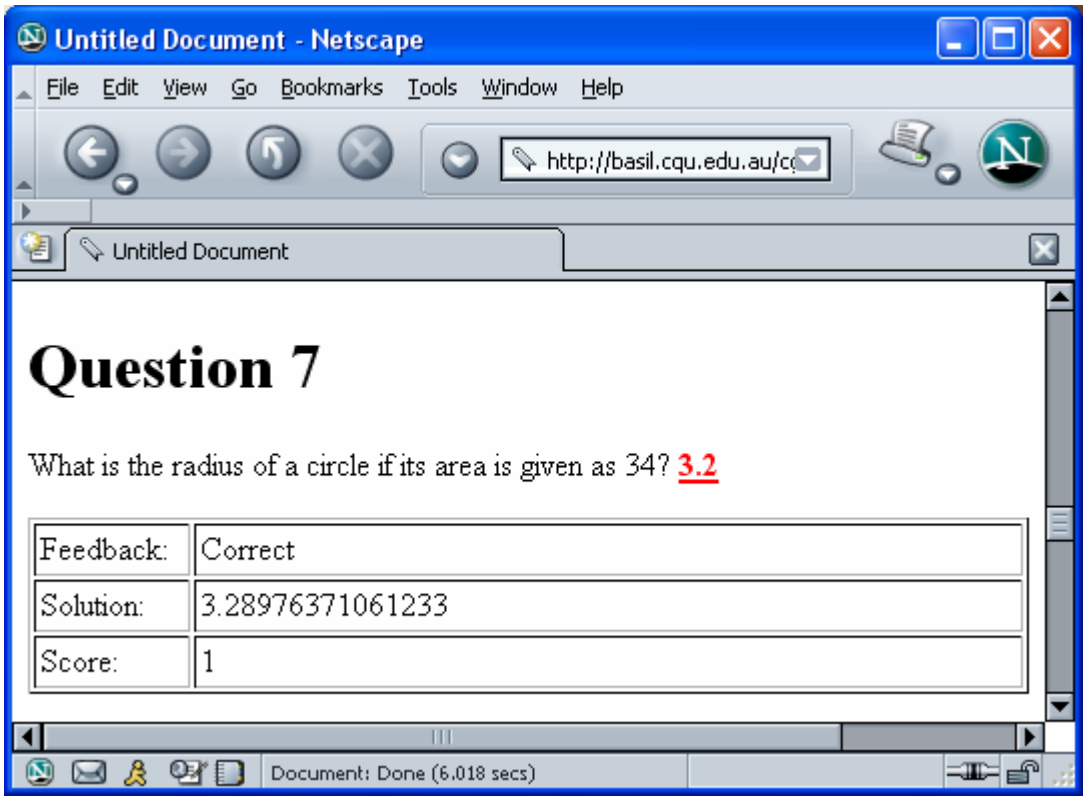


Figure 6.12 Example rendered candidate response feedback for FIB Question produced by QTIPA Quiz System.

The XML, which drives the response processing and feedback presentation, is shown in Figure 6.13. As can be seen, the response processing will check that the candidate's solution provided is greater than or equal to the parameter value "answerlower", and less than or equal to the parameter value "answerupper".

```
<resprocessing>
  <outcomes>
    <decvar defaultval="0"/>
  </outcomes>
  <respcondition>
    <conditionvar>
      <and>
        <paramlte type="Response" ident="FIB1">
          <get_param ident="answerupper"/>
        </paramlte>
        <paramgte type="Response" ident="FIB1">
          <get_param ident="answerlower"/>
        </paramgte>
      </and>
    </conditionvar>
    <setvar action="Add">1</setvar>
    <displayfeedback linkrefid="Correct" feedbacktype="Response"/>
    <displayfeedback linkrefid="Correct" feedbacktype="Solution"/>
  </respcondition>
  <respcondition>
    <conditionvar>
      <not>
        <and>
          <paramlte type="Response" ident="FIB1">
            <get_param ident="answerupper"/>
          </paramlte>
          <paramgte type="Response" ident="FIB1">
            <get_param ident="answerlower"/>
          </paramgte>
        </and>
      </not>
    </conditionvar>
    <displayfeedback linkrefid="Incorrect" feedbacktype="Response"/>
    <displayfeedback linkrefid="Incorrect" feedbacktype="Solution"/>
  </respcondition>
</resprocessing>
<itemfeedback ident="Correct" view="All">
  <material>
    <mattext>Correct</mattext>
  </material>
  <solution>
    <solutionmaterial>
      <material>
        <mattextparam><get_param ident="answer"/></mattextparam>
      </material>
    </solutionmaterial>
  </solution>
</itemfeedback>
<itemfeedback ident="Incorrect" view="All">
  <material>
    <mattext>Incorrect</mattext>
  </material>
  <solution>
    <solutionmaterial>
      <material>
        <mattext>The correct solution is: </mattext>
        <mattextparam><get_param ident="answer"/></mattextparam>
        <mattext>.  However a solution between </mattext>
        <mattextparam><get_param ident="answerlower"/></mattextparam>
        <mattext> and </mattext>
        <mattextparam><get_param ident="answerupper"/></mattextparam>
        <mattext> would have been acceptable</mattext>
      </material>
    </solutionmaterial>
  </solution>
</itemfeedback>
```

Figure 6.13 QTIPA XML supporting response processing and item feedback for FIB Question.

6.4 Summary

Chapter 6 has discussed the implementation of the QTIPA Quiz Prototype, which implements the QTIPA Specification sufficiently to demonstrate the successful implementation of parameterised quiz questions. The system supports both MCQ and FIB questions, and can perform the tasks associated with: parameter instantiation, question presentation, candidate response processing, and presentation of candidate feedback. QTIPA Quiz is a collection of Perl object-oriented classes, driven by two CGI Perl scripts through web server software.

The next chapter provides a summary of the material covered within this document, and future work for this research.

7 Conclusions

This chapter reviews the focus of this research and the problems solved. It also identifies possible future research building on the work completed.

7.1 Research achievements

The focus of this research was to investigate the development of an enhanced version of the IMS QTI Specification, to provide interoperable support for recording parameterised quiz questions. The research has achieved the following goals in supporting parameterised questions:

- integration with the IMS QTI Specification, ensuring backward compatibility with existing QTI XML material,
- implementation of parameterised textual media (as discussed in Section 4.3), including the ability to generate the following parameter types:
    - Range,
    - Enumeration,
    - Formula,
    - Condition,
    - External Program.
- support for two common question types: FIB and MCQ,
- methods to avoid parameter value intersection or more specifically MCQ distracter overlap (as discussed in Section 4.5.2).

Finally, development of a prototype online quiz system has been achieved, which demonstrates the working of the new QTIPA Specification through the example questions as described in Figure 4.5 of Section 4.4. A demonstration of the QTIPA Quiz System is available from http://cq-pan.cqu.edu.au/qtipa. The prototype system

also demonstrates the ability to record the instantiated parameters of a particular quiz attempt by a student, such that the question as was seen originally by the student can be reproduced at a later time.

7.2 Future work

The focus of this research has been on the ASI Sub-specification of the QTI; however consideration has not been made of other components of the QTI such as Selection & Ordering (refer to "Selection and Ordering" in Section 3.2.1), or Results Reporting (discussed in Section 3.2.3). For example, the Selection & Ordering Sub-specification could draw in special meta-data for QTI items that describe the parameterisation in the question, and use this meta-data in selection and ordering decisions. Similarly, the Results Reporting Specification could also be amended to support parameterisation. Given that each attempt of a particular question will have a different outcome, the context in which the candidate submitted their answer must be recorded. Furthermore, given the dynamic and random nature of such a system, audit trails would also need to be implemented within the results reporting specification, to verify system integrity. Therefore, another consideration for implementing parameterisation into the QTI is to ensure that the question stem is recorded along with the candidate's response to the question, such that what the candidate was presented with and their response, can be reproduced at a later time. Implementation through the Results Reporting Specification would be best to ensure interoperable storage of parameterised quiz results.

Furthermore, the Internal Interchange Data Language Representation as per the Results Reporting Specification would need to be aware of parameters within an item,

to facilitate inter-process communication with other related sub-systems. The interested reader is referred to the document: IMS Question & Test Interoperability: Results Reporting Information Model (IMS 2002f) for discussion on the Internal Interchange Data Language and the Results Reporting Specification. There are still other components of the QTI item that could be further explored with parameterisation.

This research has focused on two common question types being MCQ and FIB questions. There are other rendering types made available through the QTI Specification, which could include support for parameterisation. Examples include image hot-spot through the *response_xy* QTI element and drag and drop objects for assessing sequencing knowledge through the *response_grp* element. A hot-spot question could have the co-ordinates of the image parameterised, such that the question asks the candidate to identify different objects within an image where the actual co-ordinates of the object are parameterised. Or for a drag and drop ordering question, the objects themselves (either words or images) could be randomly generated. The specification of how the objects must be arranged may also be parameterised, such as ascending or descending order. Of course, the QTI does support alternate media besides text and images.

Other media types supported within the QTI could also be parameterised. Table 4.1 of Section 4.3 represents different possible media types that could be parameterised. These media types are also supported through the QTI Specification. Examples of image media parameterisation have already been discussed previously, but not audio. An audio question could be parameterised to assess a student's spelling ability. The

word to be spelt could be randomly generated through a parameter, selected through an electronic dictionary based on constraints of difficulty such as word length, and silent letters. Then this word can be presented to the candidate aurally via the *mataudio* element of the QTI, using the current advances in text to speech technology. The candidate after listening to the word would then be expected to type the correct spelling.

The external program type parameter is common to all media types possible for parameterisation. This parameter type provides almost endless possibilities and flexibility in generation of parameter values. Yet, the *program* element as per the QTIPA Specification needs to be as portable as possible, ensuring that the question XML will still work after moving to a new platform. Using platform-independent languages such as C, Perl and Java would be good languages of choice for interoperability. In certain situations though, they may be inappropriate. For example, a programming question could be required in a similar fashion to that provided in Figure 5.31 of Section 5.4.6 where a student is asked to enter the output that would be provided by a certain segment of programming code. The language being tested may not be ported to the platform in which the quiz system is running, for example, Microsoft's Visual Basic language through a quiz system implemented on a UNIX platform. To deal with such cases, the quiz system can communicate with another service by various methods, executing on the appropriate platform necessary to produce the data. For example, the XML-RPC protocol could be used to execute the external programs on an alternate system across the network for generation of the parameter values of the question. The connection would be encrypted and access to the XML-RPC server would be restricted to the quiz system only, thus not giving

away the answers to anyone who submits queries. This would then enable a UNIX based quiz system to execute a Visual Basic program on a Microsoft Windows operating system from the network. This functionality is currently supported within the QTIPA Specification through the *uri* attribute of the program element, by providing a network address.

Finally, not all quiz systems support parameterisation, which can add significant complexity. An alternate approach to providing parameterisation support for a quiz system is to develop a translation tool, which can generate all possible instances of a parameterised question, as a list of separate distinct questions. The format of the resulting question instances could be that of the traditional QTI, or of a proprietary format for a particular system. These questions could then be imported into the quiz system and categorised based on the initial parameterised question from which they were derived. The selection of questions within the quiz can be such that one question is selected from each category of questions derived from a particular parameterised question. This approach effectively allows the delivery of parameterised questions via quiz systems that do not natively support parameterisation.

Appendix A – QTIPA DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Generated by Turbo XML 2.3.1.100.-->
<!-- ************************************************      -->
<!--                                          -->
<!-- TITLE:    ims_qtiasiv1p2p1PA.dtd          -->
<!-- TYPE:     IMS Question and Test Interoperability-->
<!--         Assessment, Section, Item structure  and-->
<!--         Objects-bank with Parameterisation Addendum.    -->
<!--                                          -->
<!-- REVISION HISTORY:                        -->
<!-- Date         Author                           -->
<!-- ====         ======                        -->
<!-- 14th Feb 2003  Colin Smythe                   -->
<!--  7th Mar 2004    Damien Clark (Parameterisation Addendum)-->
<!--                                          -->
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- **********************************************
  -->
<!--           ROOT DEFINITION                -->
<!-- **********************************************
  -->
<!ELEMENT questestinterop (qticomment?, (objectbank | assessment |
(section | item)+))>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- **********************************************
  -->
<!--           ENTITY DEFINITIONS             -->
<!-- **********************************************
  -->
<!ENTITY % I_Testoperator " testoperator  (EQ | NEQ | LT | LTE | GT |
GTE )  #REQUIRED">
<!ENTITY % I_Pname " pname CDATA  #REQUIRED">
<!ENTITY % I_Class " class CDATA  'Block'">
<!ENTITY % I_Mdoperator " mdoperator  (EQ | NEQ | LT | LTE | GT | GTE
)  #REQUIRED">
<!ENTITY % I_Mdname " mdname CDATA  #REQUIRED">
<!ENTITY % I_Title " title CDATA  #IMPLIED">
<!ENTITY % I_Label " label CDATA  #IMPLIED">
<!ENTITY % I_Ident " ident CDATA  #REQUIRED">
<!ENTITY % I_View " view  (All |
          Administrator |
          AdminAuthority |
          Assessor |
          Author |
          Candidate |
          InvigilatorProctor |
          Psychometrician |
          Scorer |
          Tutor )  'All'">
<!ENTITY % I_FeedbackSwitch " feedbackswitch  (Yes | No )  'Yes'">
<!ENTITY % I_HintSwitch " hintswitch  (Yes | No )  'Yes'">
<!ENTITY % I_SolutionSwitch " solutionswitch  (Yes | No )  'Yes'">
<!ENTITY % I_Rcardinality " rcardinality  (Single | Multiple |
Ordered )  'Single'">
<!ENTITY % I_Rtiming " rtiming  (Yes | No )  'No'">
<!ENTITY % I_Uri " uri CDATA  #IMPLIED">
<!ENTITY % I_X0 " x0 CDATA  #IMPLIED">
<!ENTITY % I_Y0 " y0 CDATA  #IMPLIED">
```

```
<!ENTITY % I_Height " height CDATA  #IMPLIED">
<!ENTITY % I_Width " width CDATA  #IMPLIED">
<!ENTITY % I_Embedded " embedded CDATA  'base64'">
<!ENTITY % I_LinkRefId " linkrefid CDATA  #REQUIRED">
<!ENTITY % I_VarName " varname CDATA  'SCORE'">
<!ENTITY % I_RespIdent " respident CDATA  #REQUIRED">
<!ENTITY % I_Continue " continue  (Yes | No )  'No'">
<!ENTITY % I_CharSet " charset CDATA  'ascii-us'">
<!ENTITY % I_ScoreModel " scoremodel CDATA  #IMPLIED">
<!ENTITY % I_MinNumber " minnumber CDATA  #IMPLIED">
<!ENTITY % I_MaxNumber " maxnumber CDATA  #IMPLIED">
<!ENTITY % I_FeedbackStyle " feedbackstyle  (Complete | Incremental |
Multilevel | Proprietary )  'Complete'">
<!ENTITY % I_Case " case  (Yes | No )  'No'">
<!ENTITY % I_EntityRef " entityref ENTITY  #IMPLIED">
<!ENTITY % I_Index " index CDATA  #IMPLIED">
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- ************************************************
   -->
<!--          PARAMETERISATION ENTITIES                 -->
<!-- ************************************************
   -->
<!ENTITY % I_Type "type (Parameter | Response) #REQUIRED">
<!-- ____________________________ -->
<!ELEMENT qmd_computerscored (#PCDATA)>
<!ELEMENT qmd_feedbackpermitted (#PCDATA)>
<!ELEMENT qmd_hintspermitted (#PCDATA)>
<!ELEMENT qmd_itemtype (#PCDATA)>
<!ELEMENT qmd_maximumscore (#PCDATA)>
<!ELEMENT qmd_renderingtype (#PCDATA)>
<!ELEMENT qmd_responsetype (#PCDATA)>
<!ELEMENT qmd_scoringpermitted (#PCDATA)>
<!ELEMENT qmd_solutionspermitted (#PCDATA)>
<!ELEMENT qmd_status (#PCDATA)>
<!ELEMENT qmd_timedependence (#PCDATA)>
<!ELEMENT qmd_timelimit (#PCDATA)>
<!ELEMENT qmd_toolvendor (#PCDATA)>
<!ELEMENT qmd_topic (#PCDATA)>
<!ELEMENT qmd_material (#PCDATA)>
<!ELEMENT qmd_typeofsolution (#PCDATA)>
<!ELEMENT qmd_levelofdifficulty (#PCDATA)>
<!ELEMENT qmd_weighting (#PCDATA)>
<!ELEMENT qtimetadata (vocabulary?, qtimetadatafield+)>
<!ELEMENT vocabulary (#PCDATA)>
<!ATTLIST vocabulary
  %I_Uri;
  %I_EntityRef;
  vocab_type CDATA #IMPLIED
>
<!ELEMENT qtimetadatafield (fieldlabel, fieldentry)>
<!ATTLIST qtimetadatafield
  xml:lang CDATA #IMPLIED
>
<!ELEMENT fieldlabel (#PCDATA)>
<!ELEMENT fieldentry (#PCDATA)>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- ************************************************
   -->
<!--         COMMON OBJECT DEFINITIONS        -->
<!-- ************************************************
   -->
```

```
<!ELEMENT qticomment (#PCDATA)>
<!ATTLIST qticomment
  xml:lang CDATA #IMPLIED
>
<!-- Parameterisation addendum - material has extra optional elements
'mattextparam and matemtextparam' -->
<!ELEMENT material (qticomment?, (mattext | matemtext | mattextparam
| matemtextparam | matimage | mataudio | matvideo | matapplet |
matapplication | matref | matbreak | mat_extension)+, altmaterial*)>
<!ATTLIST material
  %I_Label;
  xml:lang CDATA #IMPLIED
>
<!ELEMENT mattext (#PCDATA)>
<!ATTLIST mattext
  texttype CDATA "text/plain"
  %I_Label;
  %I_CharSet;
  %I_Uri;
  xml:space (preserve | default) "default"
  xml:lang CDATA #IMPLIED
  %I_EntityRef;
  %I_Width;
  %I_Height;
  %I_Y0;
  %I_X0;
>
<!ELEMENT matemtext (#PCDATA)>
<!ATTLIST matemtext
  texttype CDATA "text/plain"
  %I_Label;
  %I_CharSet;
  %I_Uri;
  xml:space (preserve | default) "default"
  xml:lang CDATA #IMPLIED
  %I_EntityRef;
  %I_Width;
  %I_Height;
  %I_Y0;
  %I_X0;
>
<!ELEMENT matimage (#PCDATA)>
<!ATTLIST matimage
  imagtype CDATA "image/jpeg"
  %I_Label;
  %I_Height;
  %I_Uri;
  %I_Embedded;
  %I_Width;
  %I_Y0;
  %I_X0;
  %I_EntityRef;
>
<!ELEMENT mataudio (#PCDATA)>
<!ATTLIST mataudio
  audiotype CDATA "audio/base"
  %I_Label;
  %I_Uri;
  %I_Embedded;
  %I_EntityRef;
>
```

```
<!ELEMENT matvideo (#PCDATA)>
<!ATTLIST matvideo
  videotype CDATA "video/avi"
  %I_Label;
  %I_Uri;
  %I_Width;
  %I_Height;
  %I_Y0;
  %I_X0;
  %I_Embedded;
  %I_EntityRef;
>
<!ELEMENT matapplet (#PCDATA)>
<!ATTLIST matapplet
  %I_Label;
  %I_Uri;
  %I_Y0;
  %I_Height;
  %I_Width;
  %I_X0;
  %I_Embedded;
  %I_EntityRef;
>
<!ELEMENT matapplication (#PCDATA)>
<!ATTLIST matapplication
  apptype CDATA #IMPLIED
  %I_Label;
  %I_Uri;
  %I_Embedded;
  %I_EntityRef;
>
<!ELEMENT matbreak EMPTY>
<!ELEMENT matref EMPTY>
<!ATTLIST matref
  %I_LinkRefId;
>
<!ELEMENT material_ref EMPTY>
<!ATTLIST material_ref
  %I_LinkRefId;
>
<!ELEMENT altmaterial (qticomment?, (mattext | matemtext | matimage |
mataudio | matvideo | matapplet | matapplication | matref | matbreak
| mat_extension)+)>
<!ATTLIST altmaterial
  xml:lang CDATA #IMPLIED
>
<!ELEMENT decvar (#PCDATA)>
<!ATTLIST decvar
  %I_VarName;
  vartype (Integer | String | Decimal | Scientific | Boolean |
Enumerated | Set) "Integer"
  defaultval CDATA #IMPLIED
  minvalue CDATA #IMPLIED
  maxvalue CDATA #IMPLIED
  members CDATA #IMPLIED
  cutvalue CDATA #IMPLIED
>
<!ELEMENT setvar (#PCDATA)>
<!ATTLIST setvar
  %I_VarName;
  action (Set | Add | Subtract | Multiply | Divide) "Set"
```

```
>
<!ELEMENT interpretvar (material | material_ref)>
<!ATTLIST interpretvar
  %I_View;
  %I_VarName;
>
<!-- Parameterisation addendum - conditionvar has extra optional
elements 'paramequal,parmgt,paramlt,paramgte,paramlte' -->
<!ELEMENT conditionvar (not | and | or | unanswered | other |
varequal | varlt | varlte | vargt | vargte | varsubset | varinside |
varsubstring | durequal | durlt | durlte | durgt | durgte |
var_extension | paramequal | paramgt | paramlt | paramgte |
paramlte)+>
<!-- Parameterisation addendum - not has extra optional elements
'paramequal,parmgt,paramlt,paramgte,paramlte,paramunique' -->
<!ELEMENT not (and | or | not | unanswered | other | varequal | varlt
| varlte | vargt | vargte | varsubset | varinside | varsubstring |
durequal | durlt | durlte | durgt | durgte | paramequal | paramgt |
paramlt | paramgte | paramlte | paramunique)>
<!-- Parameterisation addendum - and has extra optional elements
'paramequal,parmgt,paramlt,paramgte,paramlte,paramunique' -->
<!ELEMENT and (not | and | or | unanswered | other | varequal | varlt
| varlte | vargt | vargte | varsubset | varinside | varsubstring |
durequal | durlt | durlte | durgt | durgte | paramequal | paramgt |
paramlt | paramgte | paramlte | paramunique)+>
<!-- Parameterisation addendum - or has extra optional elements
'paramequal,parmgt,paramlt,paramgte,paramlte,paramunique' -->
<!ELEMENT or (not | and | or | unanswered | other | varequal | varlt
| varlte | vargt | vargte | varsubset | varinside | varsubstring |
durequal | durlt | durlte | durgt | durgte | paramequal | paramgt |
paramlt | paramgte | paramlte | paramunique)+>
<!ELEMENT varequal (#PCDATA)>
<!ATTLIST varequal
  %I_Case;
  %I_RespIdent;
  %I_Index;
>
<!ELEMENT varlt (#PCDATA)>
<!ATTLIST varlt
  %I_RespIdent;
  %I_Index;
>
<!ELEMENT varlte (#PCDATA)>
<!ATTLIST varlte
  %I_RespIdent;
  %I_Index;
>
<!ELEMENT vargt (#PCDATA)>
<!ATTLIST vargt
  %I_RespIdent;
  %I_Index;
>
<!ELEMENT vargte (#PCDATA)>
<!ATTLIST vargte
  %I_RespIdent;
  %I_Index;
>
<!ELEMENT varsubset (#PCDATA)>
<!ATTLIST varsubset
  %I_RespIdent;
  setmatch (Exact | Partial) "Exact"
```

```
  %I_Index;
>
<!ELEMENT varinside (#PCDATA)>
<!ATTLIST varinside
  areatype (Ellipse | Rectangle | Bounded) #REQUIRED
  %I_RespIdent;
  %I_Index;
>
<!ELEMENT varsubstring (#PCDATA)>
<!ATTLIST varsubstring
  %I_Index;
  %I_RespIdent;
  %I_Case;
>
<!ELEMENT durequal (#PCDATA)>
<!ATTLIST durequal
  %I_Index;
  %I_RespIdent;
>
<!ELEMENT durlt (#PCDATA)>
<!ATTLIST durlt
  %I_Index;
  %I_RespIdent;
>
<!ELEMENT durlte (#PCDATA)>
<!ATTLIST durlte
  %I_Index;
  %I_RespIdent;
>
<!ELEMENT durgt (#PCDATA)>
<!ATTLIST durgt
  %I_Index;
  %I_RespIdent;
>
<!ELEMENT durgte (#PCDATA)>
<!ATTLIST durgte
  %I_Index;
  %I_RespIdent;
>
<!ELEMENT unanswered (#PCDATA)>
<!ATTLIST unanswered
  %I_RespIdent;
>
<!ELEMENT other (#PCDATA)>
<!ELEMENT duration (#PCDATA)>
<!ELEMENT displayfeedback (#PCDATA)>
<!ATTLIST displayfeedback
  feedbacktype (Response | Solution | Hint) "Response"
  %I_LinkRefId;
>
<!ELEMENT objectives (qticomment?, (material+ | flow_mat+))>
<!ATTLIST objectives
  %I_View;
>
<!ELEMENT rubric (qticomment?, (material+ | flow_mat+))>
<!ATTLIST rubric
  %I_View;
>
<!ELEMENT flow_mat (qticomment?, (flow_mat | material |
material_ref)+)>
<!ATTLIST flow_mat
```

```
  %I_Class;
>
<!ELEMENT presentation_material (qticomment?, flow_mat+)>
<!ELEMENT reference (qticomment?, (material | mattext | matemtext |
matimage | mataudio | matvideo | matapplet | matapplication |
matbreak | mat_extension)+)>
<!ELEMENT selection_ordering (qticomment?, sequence_parameter*,
selection*, order?)>
<!ATTLIST selection_ordering
  sequence_type CDATA #IMPLIED
>
<!ELEMENT outcomes_processing (qticomment?, outcomes,
objects_condition*, processing_parameter*, map_output*,
outcomes_feedback_test*)>
<!ATTLIST outcomes_processing
  %I_ScoreModel;
>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- ******************************************************
   -->
<!--           EXTENSION DEFINITIONS         -->
<!-- ******************************************************
   -->
<!ELEMENT mat_extension ANY>
<!ELEMENT var_extension ANY>
<!ELEMENT response_extension ANY>
<!ELEMENT render_extension ANY>
<!ELEMENT assessproc_extension ANY>
<!ELEMENT sectionproc_extension ANY>
<!ELEMENT itemproc_extension ANY>
<!ELEMENT respcond_extension ANY>
<!ELEMENT selection_extension ANY>
<!ELEMENT objectscond_extension (#PCDATA)>
<!ELEMENT order_extension ANY>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- ******************************************************
   -->
<!--         OBJECT-BANK OBJECT DEFINITIONS   -->
<!-- ******************************************************
   -->
<!ELEMENT objectbank (qticomment?, qtimetadata*, (section | item)+)>
<!ATTLIST objectbank
  %I_Ident;
>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- ******************************************************
   -->
<!--         ASSESSMENT OBJECT DEFINITIONS -->
<!-- ******************************************************
   -->
<!ELEMENT assessment (qticomment?, duration?, qtimetadata*,
objectives*, assessmentcontrol*, rubric*, presentation_material?,
outcomes_processing*, assessproc_extension?, assessfeedback*,
selection_ordering?, reference?, (sectionref | section)+)>
<!ATTLIST assessment
  %I_Ident;
  %I_Title;
  xml:lang CDATA #IMPLIED
>
<!ELEMENT assessmentcontrol (qticomment?)>
<!ATTLIST assessmentcontrol
```

```
  %I_HintSwitch;
  %I_SolutionSwitch;
  %I_View;
  %I_FeedbackSwitch;
>
<!ELEMENT assessfeedback (qticomment?, (material+ | flow_mat+))>
<!ATTLIST assessfeedback
  %I_View;
  %I_Ident;
  %I_Title;
>
<!ELEMENT sectionref (#PCDATA)>
<!ATTLIST sectionref
  %I_LinkRefId;
>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- ************************************************
   -->
<!--         SECTION OBJECT DEFINITIONS       -->
<!-- ************************************************
   -->
<!ELEMENT section (qticomment?, duration?, qtimetadata*, objectives*,
sectioncontrol*, sectionprecondition*, sectionpostcondition*,
rubric*, presentation_material?, outcomes_processing*,
sectionproc_extension?, sectionfeedback*, selection_ordering?,
reference?, (itemref | item | sectionref | section)*)>
<!ATTLIST section
  %I_Ident;
  %I_Title;
  xml:lang CDATA #IMPLIED
>
<!ELEMENT sectionprecondition (#PCDATA)>
<!ELEMENT sectionpostcondition (#PCDATA)>
<!ELEMENT sectioncontrol (qticomment?)>
<!ATTLIST sectioncontrol
  %I_FeedbackSwitch;
  %I_HintSwitch;
  %I_SolutionSwitch;
  %I_View;
>
<!ELEMENT itemref (#PCDATA)>
<!ATTLIST itemref
  %I_LinkRefId;
>
<!ELEMENT sectionfeedback (qticomment?, (material+ | flow_mat+))>
<!ATTLIST sectionfeedback
  %I_View;
  %I_Ident;
  %I_Title;
>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- ************************************************
   -->
<!--           ITEM OBJECT DEFINITIONS       -->
<!-- ************************************************
   -->
<!-- Parameterisation addendum - item has extra optional element
'parameters' -->
<!ELEMENT item (qticomment?, parameters?, duration?, itemmetadata?,
objectives*, itemcontrol*, itemprecondition*, itempostcondition*,
```

```
(itemrubric | rubric)*, presentation?, resprocessing*,
itemproc_extension?, itemfeedback*, reference?)>
<!ATTLIST item
  maxattempts CDATA #IMPLIED
  %I_Label;
  %I_Ident;
  %I_Title;
  xml:lang CDATA #IMPLIED
>
<!ELEMENT itemmetadata (qtimetadata*, qmd_computerscored?,
qmd_feedbackpermitted?, qmd_hintspermitted?, qmd_itemtype?,
qmd_levelofdifficulty?, qmd_maximumscore?, qmd_renderingtype*,
qmd_responsetype*, qmd_scoringpermitted?, qmd_solutionspermitted?,
qmd_status?, qmd_timedependence?, qmd_timelimit?, qmd_toolvendor?,
qmd_topic?, qmd_weighting?, qmd_material*, qmd_typeofsolution?)>
<!ELEMENT itemcontrol (qticomment?)>
<!ATTLIST itemcontrol
  %I_FeedbackSwitch;
  %I_HintSwitch;
  %I_SolutionSwitch;
  %I_View;
>
<!ELEMENT itemprecondition (#PCDATA)>
<!ELEMENT itempostcondition (#PCDATA)>
<!ELEMENT itemrubric (material)>
<!ATTLIST itemrubric
  %I_View;
>
<!ELEMENT presentation (qticomment?, (flow | (material | response_lid
| response_xy | response_str | response_num | response_grp |
response_extension)+))>
<!ATTLIST presentation
  %I_Label;
  xml:lang CDATA #IMPLIED
  %I_Y0;
  %I_X0;
  %I_Width;
  %I_Height;
>
<!ELEMENT flow (qticomment?, (flow | material | material_ref |
response_lid | response_xy | response_str | response_num |
response_grp | response_extension)+)>
<!ATTLIST flow
  %I_Class;
>
<!ELEMENT response_lid ((material | material_ref)?, (render_choice |
render_hotspot | render_slider | render_fib | render_extension),
(material | material_ref)?)>
<!ATTLIST response_lid
  %I_Rcardinality;
  %I_Rtiming;
  %I_Ident;
>
<!ELEMENT response_xy ((material | material_ref)?, (render_choice |
render_hotspot | render_slider | render_fib | render_extension),
(material | material_ref)?)>
<!ATTLIST response_xy
  %I_Rcardinality;
  %I_Rtiming;
  %I_Ident;
>
```

```
<!ELEMENT response_str ((material | material_ref)?, (render_choice |
render_hotspot | render_slider | render_fib | render_extension),
(material | material_ref)?)>
<!ATTLIST response_str
  %I_Rcardinality;
  %I_Ident;
  %I_Rtiming;
>
<!ELEMENT response_num ((material | material_ref)?, (render_choice |
render_hotspot | render_slider | render_fib | render_extension),
(material | material_ref)?)>
<!ATTLIST response_num
  numtype (Integer | Decimal | Scientific) "Integer"
  %I_Rcardinality;
  %I_Ident;
  %I_Rtiming;
>
<!ELEMENT response_grp ((material | material_ref)?, (render_choice |
render_hotspot | render_slider | render_fib | render_extension),
(material | material_ref)?)>
<!ATTLIST response_grp
  %I_Rcardinality;
  %I_Ident;
  %I_Rtiming;
>
<!ELEMENT response_label (#PCDATA | qticomment | material |
material_ref | flow_mat)*>
<!ATTLIST response_label
  rshuffle (Yes | No) "Yes"
  rarea (Ellipse | Rectangle | Bounded) "Ellipse"
  rrange (Exact | Range) "Exact"
  labelrefid CDATA #IMPLIED
  %I_Ident;
  match_group CDATA #IMPLIED
  match_max CDATA #IMPLIED
>
<!ELEMENT flow_label (qticomment?, (flow_label | response_label)+)>
<!ATTLIST flow_label
  %I_Class;
>
<!ELEMENT response_na ANY>
<!ELEMENT render_choice ((material | material_ref | response_label |
flow_label)*, response_na?)>
<!ATTLIST render_choice
  shuffle (Yes | No) "No"
  %I_MinNumber;
  %I_MaxNumber;
>
<!ELEMENT render_hotspot ((material | material_ref | response_label |
flow_label)*, response_na?)>
<!ATTLIST render_hotspot
  %I_MaxNumber;
  %I_MinNumber;
  showdraw (Yes | No) "No"
>
<!ELEMENT render_slider ((material | material_ref | response_label |
flow_label)*, response_na?)>
<!ATTLIST render_slider
  orientation (Horizontal | Vertical) "Horizontal"
  lowerbound CDATA #REQUIRED
  upperbound CDATA #REQUIRED
```

```
  step CDATA #IMPLIED
  startval CDATA #IMPLIED
  steplabel (Yes | No) "No"
  %I_MaxNumber;
  %I_MinNumber;
>
<!ELEMENT render_fib ((material | material_ref | response_label |
flow_label)*, response_na?)>
<!ATTLIST render_fib
  encoding CDATA "UTF_8"
  fibtype (String | Integer | Decimal | Scientific) "String"
  rows CDATA #IMPLIED
  maxchars CDATA #IMPLIED
  prompt (Box | Dashline | Asterisk | Underline) #IMPLIED
  columns CDATA #IMPLIED
  %I_CharSet;
  %I_MaxNumber;
  %I_MinNumber;
>
<!-- Parameterisation addendum - resprocessing has extra optional
element 'parameters' -->
<!ELEMENT resprocessing (qticomment?, outcomes, parameters?,
(respcondition | itemproc_extension)+)>
<!ATTLIST resprocessing
  %I_ScoreModel;
>
<!ELEMENT outcomes (qticomment?, (decvar, interpretvar*)+)>
<!ELEMENT respcondition (qticomment?, conditionvar, setvar*,
displayfeedback*, respcond_extension?)>
<!ATTLIST respcondition
  %I_Continue;
  %I_Title;
>
<!ELEMENT itemfeedback ((flow_mat | material) | solution | hint)+>
<!ATTLIST itemfeedback
  %I_View;
  %I_Ident;
  %I_Title;
>
<!ELEMENT solution (qticomment?, solutionmaterial+)>
<!ATTLIST solution
  %I_FeedbackStyle;
>
<!ELEMENT solutionmaterial (material+ | flow_mat+)>
<!ELEMENT hint (qticomment?, hintmaterial+)>
<!ATTLIST hint
  %I_FeedbackStyle;
>
<!ELEMENT hintmaterial (material+ | flow_mat+)>
<!--    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
<!--    ********************************************************
-->
<!--    PARAMETERISATION OBJECT DEFINITIONS          -->
<!--    ********************************************************
-->
<!ELEMENT parameters (qticomment?, (set_param | condition)+)>
<!ELEMENT set_param (qticomment?, (range | literal | enumeration |
formula | program | get_param))>
<!ATTLIST set_param
  %I_Ident;
>
```

```
<!ELEMENT mattextparam (get_param)>
<!ATTLIST mattextparam
  texttype CDATA "text/plain"
  %I_CharSet;
  xml:space (preserve | default) "default"
  xml:lang CDATA #IMPLIED
  %I_Width;
  %I_Height;
  %I_Y0;
  %I_X0;
>
<!ELEMENT matemtextparam (get_param)>
<!ATTLIST matemtextparam
  texttype CDATA "text/plain"
  %I_CharSet;
  xml:space (preserve | default) "default"
  xml:lang CDATA #IMPLIED
  %I_Width;
  %I_Height;
  %I_Y0;
  %I_X0;
>
<!ELEMENT get_param EMPTY>
<!ATTLIST get_param
  %I_Ident;
>
<!ELEMENT get_response EMPTY>
<!ATTLIST get_response
  %I_RespIdent;
>
<!ELEMENT literal (#PCDATA)>
<!ELEMENT range (range_min, range_max)>
<!ATTLIST range
  step CDATA "1"
>
<!ELEMENT range_min (literal | get_param)>
<!ELEMENT range_max (literal | get_param)>
<!ELEMENT enumeration (literal | get_param)+>
<!ELEMENT formula (#PCDATA | get_param | get_response)*>
<!ATTLIST formula
  prog_language CDATA #REQUIRED
>
<!ELEMENT program (literal | get_param | get_response)+>
<!ATTLIST program
  interpreter CDATA #IMPLIED
  uri CDATA #REQUIRED
  type (Function | External) "External"
  function CDATA #IMPLIED
>
<!ELEMENT condition (qticomment?, (conditionparam, iftrue?,
iffalse?))>
<!ELEMENT iftrue (set_param*, recalculate?)>
<!ELEMENT iffalse (set_param*, recalculate?)>
<!ELEMENT recalculate EMPTY>
<!ELEMENT conditionparam (not | and | or | paramequal | paramlt |
paramgt | paramlte | paramgte | paramunique)+>
<!ELEMENT paramequal (literal | get_param)>
<!ATTLIST paramequal
  %I_Type;
  %I_Ident;
  %I_Case;
```

```
>
<!ELEMENT paramlt (literal | get_param)>
<!ATTLIST paramlt
  %I_Type;
  %I_Ident;
>
<!ELEMENT paramgt (literal | get_param)>
<!ATTLIST paramgt
  %I_Type;
  %I_Ident;
>
<!ELEMENT paramlte (literal | get_param)>
<!ATTLIST paramlte
  %I_Type;
  %I_Ident;
>
<!ELEMENT paramgte (literal | get_param)>
<!ATTLIST paramgte
  %I_Type;
  %I_Ident;
>
<!ELEMENT paramunique (get_param, (literal | get_param)+)>
<!--    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- ****************************************************
   -->
<!-- SELECTION AND ORDERING OBJECT DEFINITIONS-->
<!-- ****************************************************
   -->
<!ELEMENT selection (sourcebank_ref?, selection_number?,
selection_metadata?, (and_selection | or_selection | not_selection |
selection_extension)?)>
<!ELEMENT order (order_extension?)>
<!ATTLIST order
  order_type CDATA #REQUIRED
>
<!ELEMENT selection_number (#PCDATA)>
<!ELEMENT selection_metadata (#PCDATA)>
<!ATTLIST selection_metadata
  %I_Mdname;
  %I_Mdoperator;
>
<!ELEMENT sequence_parameter (#PCDATA)>
<!ATTLIST sequence_parameter
  %I_Pname;
>
<!ELEMENT sourcebank_ref (#PCDATA)>
<!ELEMENT and_selection (selection_metadata | and_selection |
or_selection | not_selection)+>
<!ELEMENT or_selection (selection_metadata | and_selection |
or_selection | not_selection)+>
<!ELEMENT not_selection (selection_metadata | and_selection |
or_selection | not_selection)>
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-->
<!-- ****************************************************
   -->
<!--      OUTCOMES PREOCESSING OBJECT DEFINITIONS-->
<!-- ****************************************************
   -->
<!ELEMENT objects_condition (qticomment?, (outcomes_metadata |
and_objects | or_objects | not_objects)?, objects_parameter*,
map_input*, objectscond_extension?)>
```

```
<!ELEMENT map_output (#PCDATA)>
<!ATTLIST map_output
  %I_VarName;
>
<!ELEMENT map_input (#PCDATA)>
<!ATTLIST map_input
  %I_VarName;
>
<!ELEMENT outcomes_feedback_test (test_variable, displayfeedback+)>
<!ATTLIST outcomes_feedback_test
  %I_Title;
>
<!ELEMENT outcomes_metadata (#PCDATA)>
<!ATTLIST outcomes_metadata
  %I_Mdname;
  %I_Mdoperator;
>
<!ELEMENT and_objects (outcomes_metadata | and_objects | or_objects |
not_objects)+>
<!ELEMENT or_objects (outcomes_metadata | and_objects | or_objects |
not_objects)+>
<!ELEMENT not_objects (outcomes_metadata | and_objects | or_objects |
not_objects)>
<!ELEMENT test_variable (variable_test | and_test | or_test |
not_test)>
<!ELEMENT processing_parameter (#PCDATA)>
<!ATTLIST processing_parameter
  %I_Pname;
>
<!ELEMENT and_test (variable_test | and_test | or_test | not_test)+>
<!ELEMENT or_test (variable_test | and_test | or_test | not_test)+>
<!ELEMENT not_test (variable_test | and_test | or_test | not_test)>
<!ELEMENT variable_test (#PCDATA)>
<!ATTLIST variable_test
  %I_VarName;
  %I_Testoperator;
>
<!ELEMENT objects_parameter (#PCDATA)>
<!ATTLIST objects_parameter
  %I_Pname;
>
```

Appendix B – XML Primer

The definition of the QTI Specification has been achieved through the use of XML.

XML or eXtensible Mark-up Language has many useful properties including:

- a hierarchical structure,
- a text based syntax, making it portable across many operating systems,
- an open industry standard developed by the World Wide Web Consortium, and is derived from SGML (Standard Generalised Mark-up Language).

Due to the aforementioned advantages and growing use, XML has been selected as the technology used to define the QTI, to promote the widest possible adoption (IMS 2002c). An example of XML is illustrated in Figure 7.1.

```
<Contact_Book>
  <Contact Type=”Friend”>
    <Name Title=”Mr”>Fred Nerk</Name>
    <Phone Type=”Mobile”>0123 456 789</Phone>
  </Contact>
</Contact_Book>
```

Figure 7.1 Example XML Elements.

The fundamental component of an XML document is an element. An element is an object container that stores data, whether it is character data, or further elements. It may also have multiple attributes, which define the properties of the object. Angle brackets “< >”are used to tag element names and attribute assignments. Following these tagged element names is the data for that element, which may be either further elements or character data. To denote the end of scope for an element, a matching element name enclosed in angle brackets is used, with the exception that the name starts with a forward slash “/”. As previously mentioned, XML is a hierarchical structure. Figure 7.2 represents the XML as shown in Figure 7.1 as a tree diagram.

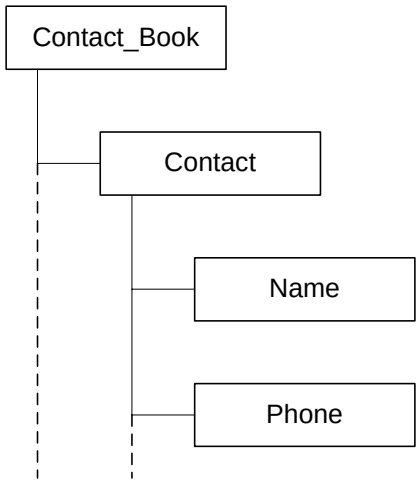


Figure 7.2 Tree diagram representing XML document.

The hierarchical nature of XML is implemented through the encapsulation of elements within other elements. This is demonstrated in Figure 7.3, which also represents the XML document as shown in Figure 7.1.

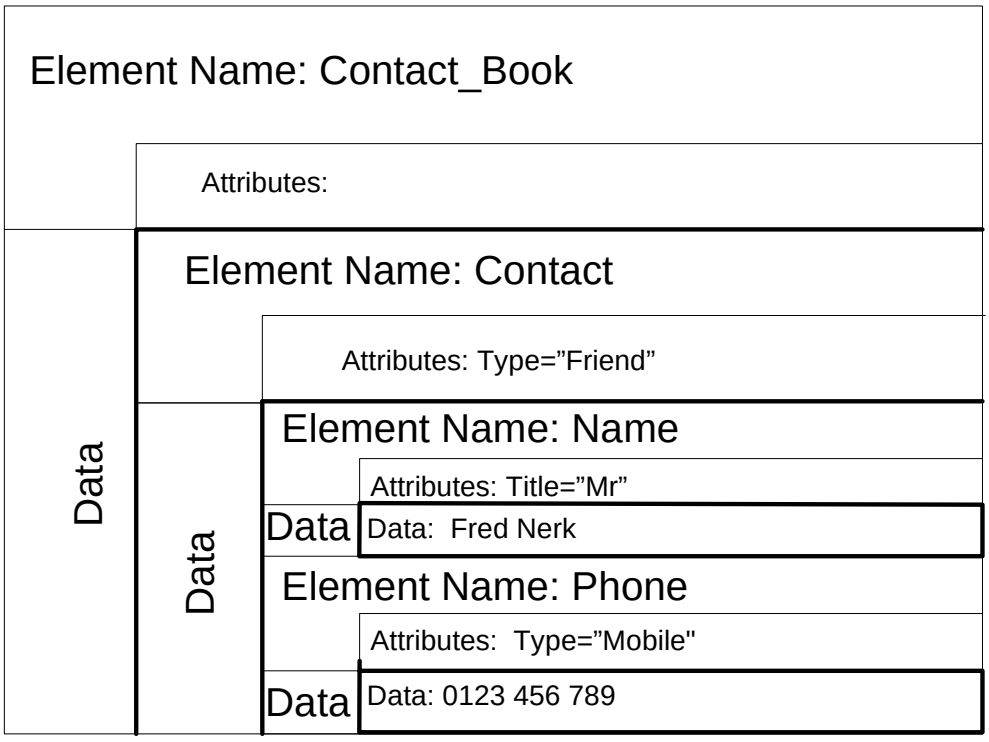


Figure 7.3 Basic structure of XML elements.

The element *Contact_Book*, has a data component which comprises of another element, *Contact*. In this scenario, *Contact* is considered a child element of

Contact_Book. The *Contact* element then contains two child elements, *Name* and *Phone*. It also has an attribute named *Type*. This attribute *Type* describes the contact as a "Friend". The *Name* element also has an attribute *Title*, describing the title of the name given, which in this case is "Mr". Furthermore, the *Name* element also contains data, but rather than being another element, instead is merely character data, which in the example is "Fred Nerk". The second element contained within *Contact* is *Phone*, and has an attribute *Type* to describe the type of phone number. In this case it is a "Mobile" phone. Finally, it is also holding character data, which is "0123 456 789". Therefore, in the example XML used, the contact book contains one contact entry, which lists a name and mobile phone number.

For an implementation of a contact book, it may also be necessary to store additional phone numbers for a contact, and of course additional contacts. Yet, how is it determined exactly what elements can be used and how they can be arranged to describe the data? One of the major benefits of XML is the ability to constrain the structure of an XML document using a Document Type Definition (DTD). Figure 7.4 provides a graphical representation of the DTD that describes this contact book.

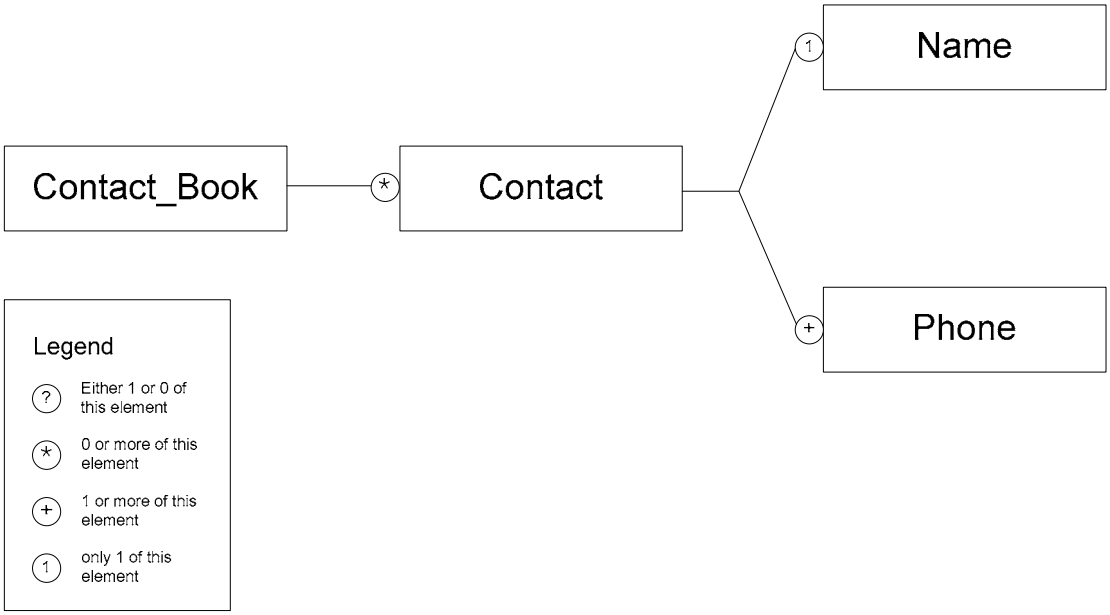


Figure 7.4 Structure of XML Example.

As illustrated and using the legend, the *Contact_Book* element may contain zero or more *Contact* elements. The *Contact* element must contain one and only one *Name* element, followed by one or more *Phone* elements. Not shown in this example, however the use of the question mark "?" requires that the element is optional meaning there can be only either one occurrence or no occurrences of the given element. The lines connecting the parent and child elements specify the structure and groupings of elements in conjunction with the cardinality symbols (i.e. *, +, ?).

By developing a DTD, a standard can be introduced which describes how an XML document should be structured for storing of a specific data set. This provides a consistent format for sharing data between different applications. The DTD can validate any XML document, to ensure that it adheres to the DTD constraints set forth for its use. An example DTD is shown in Figure 7.5, which implements the structure as illustrated in Figure 7.4 for the contact book.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!—edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Damien Clark (CQU) -->
<!ELEMENT Phone (#PCDATA)>
<!ATTLIST Phone
  Type (Mobile | Home | Office) “Office”
>
<!ELEMENT Name (#PCDATA)>
<!ATTLIST Name
  Title (Mr | Miss | Mrs | Ms | Dr) #REQUIRED
>
<!ELEMENT Contact (Name,Phone+)>
<!ATTLIST Contact
  Type CDATA #REQUIRED
>
<!ELEMENT Contact_Book (Contact+)>
```

Figure 7.5 An example DTD.

The structure of an XML document using a DTD can be constrained based on:

- the name of elements,
- the name of element attributes,
- the placement of elements within other elements (including order),
- the cardinality of elements within other elements (how often they may occur),
- whether elements may contain parsed character data (PCDATA), other elements, or both (mixed content),
- the type of the element attributes (CDATA, enumerated list of option, string, or IDREF).

Further information on the XML as used by the QTI Specification can be found in the document: IMS Question & Test Interoperability: ASI XML Binding Specification (IMS 2002b). Other useful material on XML is available from the following web sites:

- http://www.xml.com/
- http://xml.coverpages.org/xml.html
- http://www.w3.org/XML/
- http://www.xml.org/

Appendix C – Supplementary CDROM Contents

The Supplementary CDROM contains the QTIPA DTD file (ims_qtiasiv1p2p1_PA.dtd) as shown in Appendix A. It also contains a file (QTIPA.xml), which is a validated QTIPA XML quiz and describes each of the ten questions as illustrated in Figure 4.5 of Section 4.4 . Finally, the software directory on the CDROM contains the source code for the QTIPA Quiz system. Refer to the README file for installation instructions. A demonstration of the QTIPA Quiz System is available from http://cq-pan.cqu.edu.au/qtipa. Following is a manifest of all of the files contained on the Supplementary CDROM.

```
ims_qtiasiv1p2p1_PA.dtd
QTIPA.xml
README
software/tests/testharness.sh
software/tests/ScoringVar.t
software/tests/Resprocessing.t
software/tests/Responses.t
software/tests/ResponseLids.t
software/tests/Presentation.t
software/tests/Parameters.t
software/tests/ItemInstance.t
software/tests/ItemFeedback.t
software/tests/IMS.t
software/cgi-bin/submitqtipa.pl
software/cgi-bin/testqtipa.pl
software/IMS/QTIPA/ASI/Item/Resources/testnestedloops.c
software/IMS/QTIPA/ASI/Item/Resources/passwd
software/IMS/QTIPA/ASI/Item/ProgramFunctions/testnestedloops
software/IMS/QTIPA/ASI/Item/ProgramFunctions/regularexpression.pl
software/IMS/QTIPA/ASI/Item/ScoringVar.pm
software/IMS/QTIPA/ASI/Item/Resprocessing.pm
software/IMS/QTIPA/ASI/Item/Responses.pm
software/IMS/QTIPA/ASI/Item/ResponseLids.pm
software/IMS/QTIPA/ASI/Item/Presentation.pm
software/IMS/QTIPA/ASI/Item/PresentationFeedback1.pm
software/IMS/QTIPA/ASI/Item/Parameters.pm
software/IMS/QTIPA/ASI/Item/ItemInstance.pm
software/IMS/QTIPA/ASI/Item/ItemFeedback.pm
software/IMS/QTIPA/ASI/Item.pm
```

Appendix D – Glossary

ACME	Automated Courseware Management Environment
ADL:	Advanced Distributed Learning
AI	Artificial Intelligence
API:	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASI:	Assessments, Sections and Items Sub-specification
Assess:	The process of assessing a student's abilities and knowledge through the use of an assessment task.
Assessment:	Assessment is a tool used to determine a student's knowledge and abilities in a content area. An assessment is often in the form of an examination or a task assignment.
CAA:	Computer Aided Assessment
CADAL	Computer Aided Dynamic Assessment and Learning System
CALM	Computer Aided Learning in Mathematics
Candidate:	In the context of this document, a candidate is synonymous with student.
CCS	Ceilidh Courseware System
CCS:	Ceilidh Courseware System
CDATA:	Character Data
CGI:	Common Gateway Interface
DBI	Perl Database Interface

Distracter:	Distracters are selectable options of a Multiple Choice Question that are incorrect answers as asked in the stem, and are provided to sway a student from the key.
DOM:	Document Object Model
DTD:	Document Type Definition
FIB:	Fill in the blank question
Formative:	Formative assessment permits students to undertake self-assessment to identify any weaknesses in knowledge or skill.
GPL:	General Public Licence
Grade, Mark or Score:	Each of these terms refers to a measure of a student's performance as a result of assessment tasks.
HTML	Hypertext Mark-up Language
IEEE:	Institute of Electrical and Electronic Engineers
IIDL:	Internal Interchange Data Language
IMS:	IMS Global Learning Consortium
ISO:	International Organisation for Standardisation
Item:	An item in the context of this document is the equivalent of an assessment question. In the IMS QTI Specification, an item is the smallest exchangeable object that defines a complete assessment question.
JME:	Judged Mathematical Expression
Key:	The key is a selectable option of a Multiple Choice Question that is the actual correct answer as asked in the stem.
Lecturer:	A lecturer can be considered a teacher or a facilitator of knowledge, and is responsible for instructing, and assessing students.
LID:	Logical Identifier
LMS:	Learning Management System

LTSC:	IEEE Learning Technology Standards Committee
MathML:	Mark-up Language derived from XML for describing mathematics
MCQ:	Multiple Choice Questions
OASM:	Online Assignment Submission Management
OMR:	Optical Mark Reader
OSF:	Open Software Foundation
OWL	Online Web-based Learning
PCDATA:	Parsed Character Data
Perl:	Practical Extraction and Reporting Language
QTI:	Question & Test Interoperability Specification
QTIPA:	Question & Test Interoperability Parameterisation Addendum
Regular Expression	Regular expressions use a collection of meta-characters to match certain patterns within a text strings.
RPC	Remote Procedure Call
Rubric:	A rubric is a set of rules that govern the assessment of a student's performance.
SCO	Sharable Content Object as part of the SCORM project
SCORM	Sharable Content Object Reference Model
SGML	Standard Generalised Mark-up Language
Stem:	A stem is the question definition component of a Multiple Choice Question.

Student:	A student is defined in the context of this research as someone who completes assessment as part of a course of study.
Summative:	Summative assessment facilitates the determination of student's final grade for a course.
Term:	The time period in which classes are defined for students to enroll and complete a course of study.
Text box:	A text box is a screen object used by modern graphical computer systems to facilitate the input of textual information, such as a string of characters.
TML:	Tutorial Mark-up Language
TRIADS	Tripartite Interactive Assessment Delivery System
TTS	Text-to-speech
VBA:	Visual Basic for Applications
WTML:	Webtest Mark-up Language
XML:	Extensible Mark-up Language

References

Angseesing, J. (2002). Computer-marked tests in geography and geology. Computer Based Assessment (volume 2): Case studies in Science and Computing. D. Charman and A. Elmes**:** 9-14.

Anido-Rifon, L., M. J. Fernandez-Iglesias, et al. (2001). "A component model for standardized web-based education." Journal on Educational Resources in Computing (JERIC) **1**(2es).

Anzai, H., T. Hirahara, et al. (1999). WebQP: Web-based teaching assistant quizzes provider. World Conference on the WWW and Internet.

Arnow, D. and O. Barshay (1999). On-line programming examinations using WebToTeach. Conference on integrating technology into computer science education (ITiCSE), Crocow, Poland.

Ashton, H. S. and C. E. Beevers (2002). Extending flexibility in an existing online assessment system. Computer Aided Assessment (CAA'2002).

Baillie-de Byl, P. (2004). "An online assistant for remote, distributed critiquing of electronically submitted assessment." Educational technology and society **7**(1): 29-41.

Benford, S. D., E. K. Burke, et al. (1994). A courseware system for the assessment and administration of computer programming courses in higher education. Complex Learning in Computer Environments (CLCE'94).

Bigdeli, A., J. Boys, et al. (2002). OASIS-F: Development of a fuzzy online assessment system. Computer Aided Assessment (CAA'2002).

Brusilovsky, P. and P. Miller (1999). Web-based testing for distance education. WebNet'99, Honolulu, Hawaii.

Burguillo, J. C., J. V. Benlloch, et al. (2001). X-Quest: An open tool to support evaluation in distance learning. World Conference on Educational Multimedia, Hypermedia and Telecommunications.

Creative-Technology (2003). Program Features, Creative Technology. **2003**.

Crisp, G. T. (2001). Interactivity in on-line assessment in science. Computer Aided Assessment (CAA'2001), Leicestershire, UK.

Crofts, A. (1999). Enabling reuse of CAA by design. Computer Aided Assessment (CAA'99), Leicestershire, UK.

Daly, C. and J. Waldron (2002a). Introductory programming problem solving and computer assisted assessment. Computer Aided Assessment (CAA'2002).

Daly, J. (2002b). An XML-based question bank using microsoft office. Computer Aided Assessment (CAA'2002).

Dalziel, J. (2000). Integrating CAA with textbooks and question banks: Options for enhancing learning. Computer Aided Assessment (CAA'2000), Leicestershire, UK.

Dalziel, J. (2001). Enhancing web-based learning with CAA: Pedagogical and technical considerations. Computer Aided Assessment (CAA'2001), Leicestershire, UK.

Dalziel, J. and S. Gazzard (1999). Next generation computer assisted assessment software: The design and implementation of WebMCQ. Computer Aided Assessment (CAA'99), Leicestershire, UK.

Darbyshire, P. (2000). Distributed web-based assignment management. Web based learning and teaching technologies: Opportunities and challenges. A. Aggarwal. London, Idea Group Publishing**:** 198-215.
Davies, P. (2001). Computer aided assessment must be more than multiple-choice tests for it to be academically credible? Computer Aided Assessment (CAA'2001), Leicestershire, UK.
Davies, P. (2002). There's no confidence in multiple-choice testing, ... Computer Aided Assessment (CAA'2002).
Dodds, P. (2003). The sharable content object reference model (SCORM), Advanced Distributed Learning. **2003**.
English, J. and P. Siviter (2000). Experience with an automatically assessed course. Conference on integrating technology into computer science education (ITiCSE), Helsinki, Finland.
Farah, H. (2002). iQUIZ: A tool making internet quizzes easy to develop and use. World Conference on E-Learning in Corp., Govt., Health., and Higher Ed.
Ferrandez, T. (1998). Development and testing of a standardized format for distributed learning assessment and evaluation using XML. Department of Electrical and Computer Engineering. Orlando, Florida, University of Central Florida.
Gayo, J. E. L., J. M. M. Gil, et al. (2003). A generic e-learning multiparadigm programming language system: IDEFIX project. Technical Symposium on Computer Science Education, Reno, USA, ACM Press.
Hay, B. and K. Nance (2002). Using javascript for client-side online testing. World Conference on Educational Multimedia, Hypermedia and Telecommunications.
Hill, T. G. (2003). "MEAGER: Microsoft Excel automated grader." The Journal of Computing in Small Colleges **18**(6): 151-164.
Hubler, A. W. and A. M. Assad (1995). CyberProf: An intelligent human-computer interface for asynchronous wide area training and breakching. 4th International World Wide Web Conference.
Huizinga, D. (2001). Identifying topics for instructional improvement through on-line tracking of programming assessment. Conference on integrating technology into computer science education (ITiCSE), Canterbury, United Kingdom.
IEEE (2003). IEEE learning technology standards committee (LTSC), IEEE. **2003**.
IMS (2000). IMS question & test interoperability specification: A review, IMS Global Learning Consortium, Inc. **2003**.
IMS (2002a). IMS question & test interoperability: ASI best practice & implementation guide, IMS Global Learning consortium, Inc. **2003**.
IMS (2002b). IMS question & test interoperability: ASI XML binding specification, IMS Global Learning consortium, Inc. **2003**.
IMS (2002c). IMS question & test interoperability: ASI selection & ordering. **2003**.
IMS (2002d). IMS question & test interoperability: ASI outcomes processing. **2003**.
IMS (2002e). IMS question & test interoperability QTILite Specification. **2003**.
IMS (2002f). IMS question & test interoperability: Results reporting information model. **2003**.
IMS (2002g). IMS question & test interoperability: ASI information model, IMS Global Learning consortium, Inc. **2003**.
IMS (2002h). E-learning specifications: Why do you need them?, IMS. **2003**.
IMS (2003a). Instructional management systems global learning, IMS. **2003**.

IMS (2003b). Directory of products and organisations supporting IMS specifications, IMS Global. **2003**.
IMS (2003c). IMS question and test interoperability addendum - version 1.2.1, IMS Global Learning consortium, Inc. **2003**.
IMS (2003d). Question and Test interoperability - Frequently Asked Questions, IMS. **2003**.
IMS (2004a). IMS Learner Information Package Specification, IMS. **2004**.
IMS (2004b). IMS Content packaging specification, IMS. **2004**.
IMS (2004c). IMS Simple Sequencing Specification, IMS. **2004**.
ISO (2003). ISO-IEC JTC1 SC36, ISO. **2003**.
Jackson, D. (2000). A semi-automated approach to online assessment. Conference on integrating technology into computer science education (ITiCSE), Helsinki, Finland.
Jacobsen, M. and R. Kremer (2000). Online testing and grading using WebCT in computer science. World Conference on the WWW and Internet.
Jefferies, P., I. Constable, et al. (2000). Computer aided assessment using WebCT. Computer Aided Assessment (CAA'2000), Leicestershire, UK.
Johnson, M. (2001). Attributes versus elements: The never-ending choice. IT World.com, IT World.com. **2003**.
Jones, D. and S. Behrens (2003). Online assignment management: An evolutionary tale. Hawaii International Conference on System Sciences, Waikoloa Village, Hawaii.
Jones, D. and B. Jamieson (1997). Three generations of online assignment management. Australian Society for Computers in Learning in Tertiary Education Conference, Perth, Australia.
Joy, M. and M. Luck (1998). Effective electronic marking for on-line assessment. Conference on integrating technology into computer science education (ITiCSE), Dublin, Ireland.
Kashy, E., Y. Thoennessen, et al. (1997). Using networked tools to enhance student success rates in large classes. Frontiers in Education Conference, Stipes Publishing L.L.C.
Kimber, W. E. (1997). Re: Designing a DTD: Elements or attributes? Newsgroup: comp.text.sgml. **2003**.
Li, R. (2003). Using Macromedia Flash MX to create interactive online quizzes. World Conference on Educational Multimedia, Hypermedia and Telecommunications.
Lister, R. (2000). On blooming first year programming and its blooming assessment. ACE'2000, Melbourne, Australia.
Long, S. (2000). Lessons in the development and deployment of automated IT skills acreditation. Computer Aided Assessment (CAA'2000), Leicestershire, UK.
Lundquist, R. (2001). Quiz collaboration -- cheating or a learning opportunity? World Conference on Educational Multimedia, Hypermedia and Telecommunications.
MacKay, B. R. (2000). Web-based assessment of writing skills. Computer Aided Assessment (CAA'2000, Leicestershire, UK.
Mackenzie, D. (2000). Production and delivery of TRIADS assessments on a university-wide basis. Computer Aided Assessment (CAA'2000), Leicestershire, UK.

Mason, D. V. and D. M. Woit (1999). Providing mark-up and feedback to students with online marking. SIGCSE Technical Symposium on Computer Science Education, New Orleans, USA.

Matsuno, R., Y. Tsutsumi, et al. (1999). Interactive courseware quiz creator: An adaptive zero-programming content development tool for students learning and testing via Intra/Internet. World Conference on the WWW and Internet.

Megginson, D. (1997). Element content vs. element attribute. xml-dev mailing list archives. **2003**.

Merat, F. L. and D. Chung (1997). World wide web approach to teaching microprocessors. Frontiers in Education Conference, Stipes Publishing L.L.C.

Mine, T., T. Shoudai, et al. (2000). Automatic exercise generator with tagged documents considering learner's performance. World Conference on the WWW and Internet.

Miner, R. and J. Schaeffer (2001). A gentle introduction to MathML. Design Science. **2003**.

Muldner, M. and D. Currie (1999). Techniques to implement high speed scalable dynamic on-line systems. World Conference on the WWW and Internet.

Pain, D. and J. L. Heron (2003). "WebCT and online assessment: The best thing since SOAP?" Journal of International Forum of Educational Technology & Society **6**(2): 62-71.

Palmer, J., R. Williams, et al. (2002). Automated essay grading system applied to a first year university subject. Informing Science.

Paterson, J. (2002). What's in a name? - A new hierarchy for question types. Computer Aided Assessment (CAA'2002).

Pathak, S. and P. Brusilovsky (2002). Assessing student programming knowledge with web-based dynamic parameterized quizzes. EdMedia'2002, Denver, Collarado.

Peat, M., S. Franklin, et al. (2001). A review of the use of online self-assessment modules to enhance student learning outcomes: Are they worth the effort of production. ASCILITE'2001, Melbourne, Australia.

Price, B. and M. Petre (1997). Teaching programming through paperless assignments: An empirical evaluation of instructor feedback. Conference on integrating technology into computer science education (ITiCSE), Uppsala, Sweden.

Ratna, A. A. P., P. Raymonth, et al. (2003). Distance E-learning implmentation and analysis on Jarkom-Online evaluation system. World Conference on Educational Multimedia, Hypermedia and Telecommunications.

Richmon, S., P. O'Shea, et al. (2002). QIICC - Using technology to involve students in assessment. World Conference on Educational Multimedia, Hypermedia and Telecommunications.

Roantree, M. and T. E. Keyes (1998). Automated collection of coursework using the web. Conference on integrating technology into computer science education (ITiCSE), Dublin, Ireland.

Rustici, M. (2004). Two minute SCORM overview for developers. **2004**.

Sclater, N. and K. Howie (2000). The Scottish computer assisted assessment network. Computer Aided Assessment (CAA'2000), Leicestershire, UK.

Sheard, J. and A. Carbone (2000). Providing support for self-managed learning? World Conference on the WWW and Internet 2000.

Sjoer, E. and S. Dopper (2002). Are the promises of online assessment being provided in practice? A case study into what conditions should be met in order to use online assessment successfully. EdMedia'2002, Denver, Collarado.

Smythe, C. and P. Roberts (2000). Overview of the IMS question & test interoperability specification. Computer Aided Assessment (CAA'2000), Leicestershire, UK.

Summons, P., J. Coldwell, et al. (1997). Automatic assessment and marking of spreadsheet concepts. Australiasian Computer Science Education Conference (ACSE), Melbourne, Australia.

Thomas, P. (2000). Reducing the distance in distance education. Computer Aided Assessment (CAA'2000), Leicestershire, UK.

Thomas, P., B. Price, et al. (2001). Experiments with electronic examinations over the Internet. Computer Aided Assessment (CAA'2001), Leicestershire, UK.

Trivedi, A., D. C. Kar, et al. (2003). "Automatic assignment management and peer evaluation." The Journal of Computing in Small Colleges **18**(4): 30-37.

White, J. (2000). Online testing: The dog sat on my keyboard. International Conference on Technology in Collegiate Mathematics, Atlanta, Georgia.

Williams, J., P. Browning, et al. (1997). "The NetQuest Project: Question delivery over the web using TML." Computers in Higher Education Economics Review **11**(2).

Winslow, J. (2002). Cheating an online test: Methods and reduction strategies. World Conference on E-Learning in Corp., Govt., Health., and Higher Ed.

Woit, D. and D. Mason (1998). Lessons from On-Line Programming Examinations. Conference on integrating technology into computer science education (ITiCSE), Dublin, Ireland.

Woit, D. and D. Mason (2000). Enhancing student learning through online quizzes. SIGCSE Technical Symposium on Computer Science Education, Austin, USA.

Woodbury, J., M. Ratcliffe, et al. (2001). Building and deploying an extensible CAA system: From theory to practice. Computer Aided Assessment (CAA'2001), Leicestershire, UK.

Woolf, B., R. Day, et al. (1999). OWL: An integrated web-based learning environment. International Conference on Mathematics/Science Education and Technology.