Modelling test-interactions

Educational Technology Expertise Centre OTEC
Open University of the Netherlands

Modelling test-interactions

COLOPHON

Title:	Modelling test-interactions
Author(s):	Henry Hermans, Bert van den Berg, Hubert Vogten, Francis Brouns, Marc Verhooren
Project:	Development Programme WP 2
Project manager:	G. Rodenburg
Project members:	E.J. van den Berg dr.ir. F.M.R. Brouns J.H.M. Daniels drs. H.J.H. Hermans ing. M. Verhooren ing. H.F. Vogten
Consultancy:	J.Berkhout mw.drs. M. Doorten mw.drs. D. Joosten-ten Brinke H.G.H. Martens ing. M. Meex drs. J.H.A.N. Rikers mw.drs. D.M.A. Sluijsmans ir. G.W. van der Vegt
Programme:	Research and Development into Learning Technologies
Programme chair:	Prof.dr. E.J.R. Koper
Programme assistant:	Mieke Haemers
Report type:	Report
Publication date:	2 July 2002
Distribution:	OTEC

Educational Technology Expertise Centre (OTEC)
Open University of the Netherlands

Modelling test-interactions

OTEC report series

The Open University of the Netherlands develops higher distance education and is a central partner in a consortium for the renewal of higher education. Educational technological innovation is one of the main fields of interest. The educational and educational technological expertise of the Open University of the Netherlands are bundled in the Educational technology expertise centre (OTEC). This centre is involved with tasks concerning the development, innovation, research and evaluation of the Open University and its consortium partners. These tasks are performed in close collaboration with directorates and faculties of the Open University and/or its consortium partners.

OTEC publishes a report series. This report is part of that series.

Development Programme reports and some documents can be obtained from:

Open University of the Netherlands
Secretary Development Programme
P.O. Box 2960
6401 DL Heerlen
the Netherlands
Tel. +31 45 5762624
Fax. +31 45 5762800

Internet: http://www.ou.nl/otec

Table of contents

1. Introduction

This report presents the results of work package 2 ('Technology') of the Development programme of the OUNL. The activities reported upon have been carried out from October 2001 until the first week of December 2001 on the specific area of test-interactions. Initially the intended scope was to cover the whole process of creating an UML domain model within the area of test-interactions, validating this model by gathering relevant cases and working out best practices, and at last translating this model into a XML DTD binding. However, reallocation of the available human resources in the beginning of December 2001 forced us to stop the process temporarily. As a result this report does not cover the whole spectrum, but presents the interim results. The main focus in this report is on the process of constructing a coherent domain model on interactions. The WP2 handbook on interactions complements this report.

1.1 Background

One of the major spin-offs from the Development program of the OUNL is EML 1.0 as published in December 2000 on the EML website (http://www.ou.nl/eml). EML or Educational Modelling Language in a narrow sense refers to a specific XML DTD binding for creating educational content. In a broader and preferred sense it refers to a generic model on learning design with a specific implementation in XML (the EML DTD).
The DTD provides two mechanisms to model interactions:

- Predefined modelling of structured interactions
- Unstructured interaction modelling

Both mechanisms will be described below.

The predefined interaction model was primarily derived from the functional specifications of the Vespucci project (see Veldmeijer, 09-1999). However, these models have never been tested extensively. Authors could experiment only with four types of interaction, because only these were available in the authoring environment (Framemaker+SGML 5.5.6) supplied to OUNL authors (see table 1). These were the only four types supported by the EML web player Edubox.

Implemented in player	Not implemented in player
Multiple choice question	Sequence question
Multiple response question	Matching question
Question answer	Short-answer question
True-false question	Prompt

Table 1: Supported predefined structured interactions in Edubox

1.1.1 Predefined structured interactions

1.1.1.1 EML 1.0 Interactions model

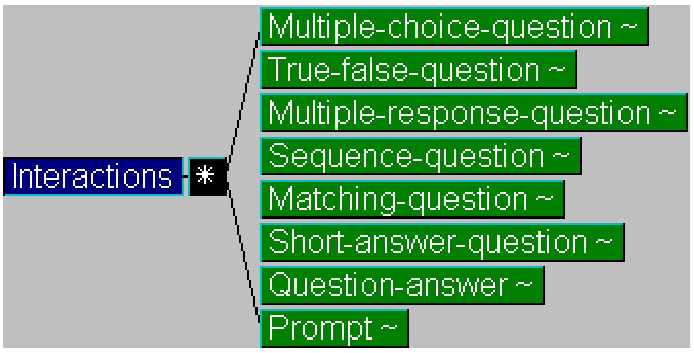


Figure 1: Interactions model

This model exists throughout the EML DTD and is part of the 'extra-p' model. This interactions-model covers eight question-types with a predefined structure (see figure 2 for an example). The application of this model is primarily in the area of intake, monitoring (self-assessment e.g.) or evaluation.

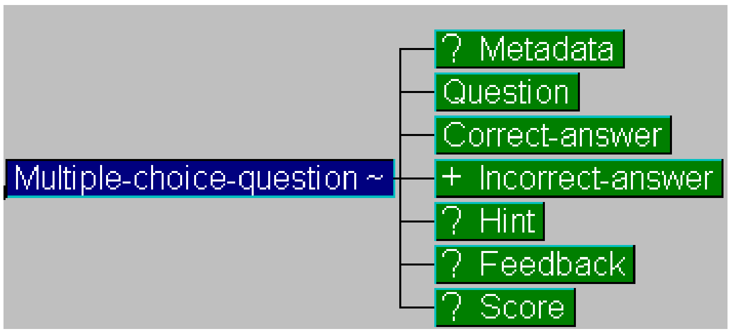


Figure 2: Multiple-choice mode in EML 1.0

1.1.1.2 EML 1.0 Questionnaire object

The questionnaire object in EML 1.0 is in fact an extension of the interactions model. It can be used to group questions and provide additional specifications for the way of presenting and processing. A typical application of the questionnaire object is within the area of forms and self-assessment.

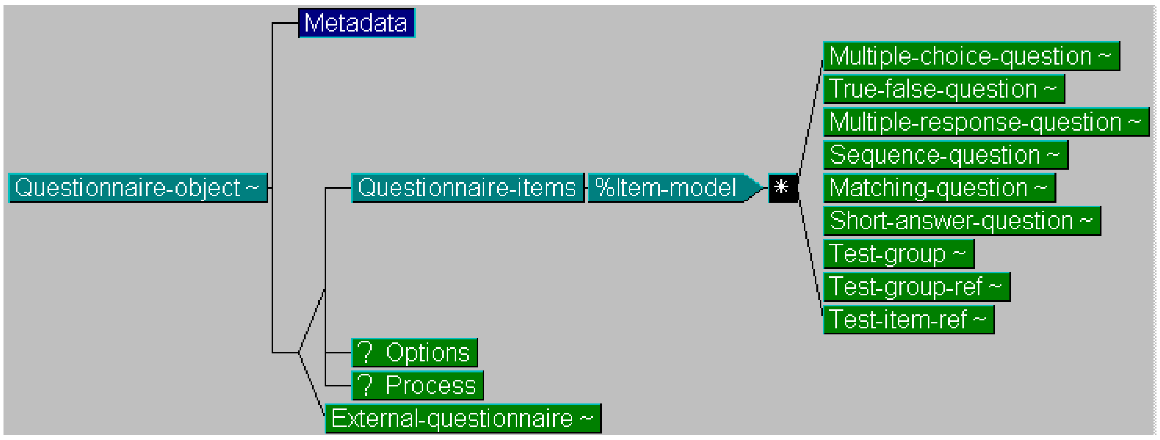


Figure 3: Questionnaire object

1.1.2 Unstructured interaction modelling

In EML interactions can also be modelled without making use of predefined structures. These kinds of interactions are modelled with the use of so-called properties. These properties, which have to be declared within the role-specification of a unit-of-study can take several forms and may contain predefined values. The set-property element can be used to provide the end-user in run-time with possibilities to interact, e.g. to make a choice between given values, fill in a value or string or upload a document. Other EML structures (e.g. method, activity-description) can specify actions of the educational system. These actions can differ according to specific property-values. Property-values can for example influence the study path or the visibility of particular information.

1.2 User experiences and problem definition

During the last two years a large amount of users within the OUNL has become more or less acquainted with EML and the EML authoring process, in particular via the so-called OUNL 'pilot projects' and the HHM-project. In this period many questions were raised on the area of interactions, both on the modelling part (authoring environment) as on the handling and delivery part (player). Authors mentioned issues, which could not be addressed with the current model. It was not possible to specify feedback for specific answer choices. Answers could not be stored (e.g. in properties) or retrieved. Therefore authors were forced to create much server-time consuming work-arounds. There was also a lack of certain types of interaction such as open answer items.
In addition it was not possible to model many aspects of testing like processing of results, score transformation, adaptive testing, etc.

To guarantee a wide use of EML, it might be relevant to look at existing standards such as the IMS QTI and incorporate those, if possible. Another important aspect is user acceptance; meaning that authors needs and expectations should be met. Changes in the interaction model which take these considerations into account might lead to a better acceptance. Through this adaptation a better integration and compatibility with other assessment or test software packages becomes possible.

Thus the ultimate goal of this project was to create an interactions model that is generic and complete and to derive bindings from it, in particular for future versions of EML. It is not likely that such a binding will cover the complete model, but rather only focus on parts of it. However creating such a model will make it easy to understand which components and relationships have to be taken into account when deriving a binding for a particular purpose. Hence one could see this model as the first steps towards a structured approach in defining a reference model in the area of interactions.

1.3 IMS QTI

Before starting the design of the interaction model we discussed the possibility of using the IMS Question and Test Interoperability specification (IMS QTI 1.0) for our purposes. We had several reasons not to adopt this specification at that moment:

- The QTI is a very technical approach towards describing interactions. The QTI dissects an interaction into its basic components like content, response types etc. Although from a systems perspective this seems to be a very powerful approach, allowing numerous types of items, it misses semantics. EML however incorporates a very semantic view that seems to conflict with the technical approach QTI has chosen. An example that will illustrate these differences is a multiple choice item. In EML 1.0 a multiple choice item is modelled using the Multiple-choice-question tag (see figure 2). For a system (user or computer) it is therefore immediately clear what the type of item is, when processing it. When using the QTI standard, the same multiple choice question would be modelled using tags like presentation, material and response. A system (user or computer) processing these tags knows how to behave, but is not aware that it is dealing with a multiple choice question.

- It was very difficult to determine whether all situations can be modelled using QTI, or indeed any modelling specification, due to lacking requirements. We therefore decided that we had to design our interaction model first, so we had a reference model that we could use to evaluate the completeness and effectiveness of any exisiting specification. Indeed we recognised that after having completed the model a re-evaluation of the usefulness of the QTI would be necessary.
- At the moment the interaction model was conceived it was not clear how EML would be positioned in IMS. So we had to make sure that the bulk of our efforts would be useful regardless of the fact if EML would be an integral part of the IMS specifications or not. If so, we could use the reference model to make suggestions for improvements of the QTI standards within IMS if necessary. Otherwise we could derive a proprietary specification from the model.

2. Design method

The Development programme agreed to use the Unified Modelling Language (UML) for object oriented system analysis to streamline communication processes and to support design methods. Although this UML is in itself a language and not a design method it is based on the principles of object oriented software design.

The starting point for object oriented development is domain modelling. The domain model is a map of the reality in which a system functions and consists of concepts, behaviour and relations between objects and classes. In a software lifecycle three phases can be distinguished: analysis, design and implementation. In the timeframe of this project we could only pay attention to the analysis. The results of the project are described in UML class diagrams.

The level of proficiency in UML was different among the project team members. The information technologists were familiar with UML, but for the educational technologists it was a new experience.

The project team consisted of a domain expert in the field of assessment and testing, educational technologists and information technologists. Additional domain experts were invited for the some of the sessions or were consulted on specific issues.

Twelve sessions were organised in all, which are described in the next paragraphs.

2.1 Session overview

The domain model was developed in several sessions. The next paragraphs provide insight in how this process evolved in time. A short description is given of every session. It provides a closer look at the development process and reveals some obstacles and pitfalls. For a more detailed description of the sessions, the Interactions handbook can be consulted.

2.1.1 Session 1 (11-10-01)

Preparation/input

- Each participant received a reader containing the following documents:
  - Veldmeijer, F. (1999). VELO - Functional description version 1.0. (pp 39-49) (confidential report).
  - Rikers, J. H. A. N. (1989). Een classificatie van item vormen. Twente: Toegepaste onderwijskunde.
  - Cap Gemini (1998). Objecten model – Open Universiteit. (pp21-30).
  - IMS – Question and Test Interoperability specification.
  - Hermans, H., Rikers, J. & De Haan, D. (1997). ITEM-TSS Detailontwerp rapport 2: beschrijving module toetsconstructie en –beheer. (OTEC Werkdocument 97/W15.
- Three test experts were invited.
- Test experts were asked to prepare a presentation providing a full overview of the test design and development process.
- Brown papers were put on the walls; a digital camera was available; all kinds of auxiliary aids were available.

Process
At the start of this session the general intentions and approach were explained by the moderator of this session. It was acknowledged that most or all participants were unfamiliar with their specific role within this new UML approach.

After the general introduction a presentation was held by one of the test experts about test development. The following four steps were identified in this process:
1. Identifying the objectives.
2. Test construction.
3. Test delivery.
4. Assessment of test results.

This presentation triggered the scope discussion: what is to be modelled? The following can be stated as the result of this discussion:
- Domain of 'testing' is to be modelled.
- Workflow is to be modelled.
- Not the test design or creation process is to be modelled, but the results of this.
- Actors are not to be included.

After this scope discussion a first attempt to identify relevant classes in the testing domain was made. This resulted in a huge number of potential classes, showing a large overlap. The next step, the attempt to cluster the classes, triggered much discussion, resulting in the second scope discussion focusing on testing interactions versus instructional design.
Major conclusions of the discussion were:
- Test construction (later to be called 'test definition') should be given a crucial position. In the modelling of test construction both EML test-items as well as external available test-items (e.g. from some specific itembank) should be incorporated.
- Also test score processing should belong to the domain.

Results and conclusions
A simple model, containing six classes, along with the first attempt to create a datadictionary, was the output of this session.
The need for a simple model as a start for modelling the whole domain was evident.
Walking an incremental path seems to be a better approach.
More specific preparation is needed to streamline the discussion.

2.1.2 Session 2 (18-10-01)

Preparation/input
- One of the test experts wrote a document on the processes involved in assessment.
- An intermediate meeting in a smaller group resulted in an adjusted domain model, which was handed out at the start of the session.
- Less test experts were invited and present in this session.
- It was also decided that less information technologists should be present to prevent that the focus was too much on implementation aspects.

Process
The domain model was discussed. This model invited the participants to give more precise the definitions of the terms used. The class test was discussed conceptually. One of the outcomes of this discussion was to rename the class in Test-definition. The class Test-part was introduced to allow construction of subtests. Two association-classes with attribute weight were added to the model, which identify the relation between Item, Test-part and Test-definition.

Part of the discussion focused on the class Item. The concept of test-item was analysed, resulting in the following conclusions:

- An item or test-item can be perceived as a stimulus given to a respondent to invoke a response.
- The response contains the output of the respondent.
- This stimulus contains a context, question and response-mode.
- The response-mode can be divided into constructed responses, where the respondent has to create a response, and selected responses, where the respondent can choose from a given set of responses.
- Each item must have (a) learning objective(s).
- Classification of items should be based on the response rather than on the type of stimulus.

Another discussion focused on how to model the respondent score on a test-item. It was concluded that this score could be seen as part of the response rather than part of the test-item. In addition a test-item needs to have a scale, in which this score can be expressed. So scale was added as an attribute to Item, score as an attribute to Response.

Results and conclusions
An adjusted domain model.
Preparation in small group seems to be more effective than a collective cold start.

2.1.3 Session 3 (31-10-01)

Preparation/input
An adjusted domain model and a report of session 2 were handed out to the participants.

Process
The scope discussion started again. It became clear that the process of test construction should not be part of the model. Only the relevant results of this process should be modelled.

Furthermore it was becoming clear that the model should reflect a static as well as a dynamic part. The test definition was being considered as the static part of the model. Based on this test definition, one or more tests can be generated, which belongs to the dynamic part. However, this test was not yet one of instantiated objects of the test definition.

In the dynamic part also the class Test-result was added. This class was needed to store the collected test results. Response is also dynamic, and as such related to the classes Item and Respondent.

A major part of this session focused on scoring:

- Scoring information was added to the test definition. However, there appeared to be different opinions on the meaning and positioning of this scoring information (e.g. caesura).
- It was agreed that normalisation of scores should occur to be able to weigh several possible scores of for example different test parts. Normalisation was conceived as a function and as such was added as an operation to the class.

Results and conclusions
An adjusted domain model.
The understanding of the distinction between a static and dynamic part of the model is essential.
At this moment the scope appeared to be clear to most of the participants.

2.1.4 Session 4 (08-11-01)

Preparation/input
An adjusted domain model and report of session 3 were handed out to the participants.
An intermediate meeting with a test expert took place in order to get more insight in aspects and terms belonging to the scoring process.

Process
The main topic of this session was scoring. A major part of the session was spent on getting a common understanding of the scoring process and sharpening definitions. Terms like raw score, true score and standard score were introduced. However, it was not yet clear whether they should become part of the model.

This session revealed an increasing need for an object model to validate the class model, especially those aspects related to scoring. An existing prior knowledge test was chosen for this object model. To be able to investigate the class model more completely the test specification was modified to some extent. The elaboration of this object model led to several discussions and resulted in modifications of the domain model. There also appeared to be a need to fit correction rules (e.g. for guessing) into the model.

Results and conclusions
An adjusted domain model.
Creating and discussing object models were perceived as suitable means for validating the domain model.

2.1.5 Session 5 (13-11-2001)

Preparation/input
An adjusted domain model and report of session 4 were handed out to the participants.

Process
The session started with a reflection on the object model as developed in the previous session. In addition correction for guessing was applied within this model.
Based on the resulting discussion several changes were made to the domain model. These changes all related to scoring issues.
In addition it was agreed that non-response should also be treated as a type of response and should be modelled the same way.

Results and conclusions
An adjusted domain model.
Again, working out the object model proved its value for modelling the domain.

2.1.6 Session 6 (15-11-2001)

Preparation/input
An adjusted domain model and a report of session 5 were handed out to the participants.
In addition the first draft of the datadictionary was handed out.

Process
The input documents were discussed. The first draft of the datadictionary was welcomed as a necessary addition to the class diagram (domain model). A terminological discussion followed. Terms within the dictionary were compared with those used within the class

diagram. Several adjustments are made and an extra column was added to the datadictionary to indicate related terms.

Scoring issues were subject of discussion again. The problem of initial caesura (in the static part) and definite caesura (in the dynamic part) was handled. The question how to deal with different groups of respondents related to the caesura issue was raised.

In the following part of the session the Item class and its directly related classes were taken into more detailed consideration. This led to several adjustments in the model. In addition the respondent's actual reaction to an item was added as an attribute to the dynamic Response class.

Results and conclusions
Major issues like feedback, adaptive testing and item types were still not addressed.
It was decided to postpone adaptive testing and to schedule remaining major issues.
The datadictionary was a valuable, necessary addition.
Classes should be named more precisely.
Psychometric data related to (test) items should not be modelled here.
The distinction between the dynamic and static parts remained a valuable one.

2.1.7 Session 7 (20-11-2001)

Preparation/input
An adjusted domain model and a report of session 6 were handed out to the participants.

Process
The input documents were discussed and several corrections were made. The main discussion in this session focused on how to classify item types. It was explored whether the classification should be based on type of question or on type of answer. Conclusions were made that the type of answer to be given plays a crucial role in the scoring process. As a result the classification on the basis of answer type seemed to be preferable. The IMS QTI solution was also discussed. This solution was considered to be too general and abstract. Almost all semantics (one of the EML's premises) would be lost with this model.

Results and conclusions
No adjustments were made to the model.
Within the current model no workflow or routing can be modelled. This kind of routing is often used in questionnaires.
Classification of item types should preferably be based on answer types.
IMS QTI is no solution for semantic modelling.
Adaptive testing can probably be modelled within other structures of EML.

2.1.8 Session 8 (27-11-2001)

Preparation/input
A report of session 7 and a preliminary diagram of item types were handed out to the participants.

Process
Discussion of the input document with the preliminary diagram of item types was the main topic of this session. In this document items had been divided in two main categories: ‘closed response’ and ‘open response’. Typical for closed response items is that the respondent either

has to choose between given answer options or has to construct an answer (e.g. make a sequence or a match) by combining a set given building blocks. On the other hand open response refers to situations where the respondent has to construct the answer herself without the help of predefined answers to choose from or build with.

Results and conclusions
Closed response versus open response remained the main distinction between item types. Closed response can be divided into (1) selection, (2) combination and (3) sequence. Open response is divided into (1) fill in, (2) performance (3) short answer and (4) essay.
It was decided to collect use-cases or cases from several test-experts to validate the model obtained so far. Case gathering among colleagues was started.
The class model contained terms in multiple languages. For the moment all terms should be translated into Dutch.

2.1.9 Session 9 (29-11-2001)

Preparation/input
An adjusted domain model (with respect to item types) and report on session 8 were handed out to the participants.

Process
A discussion of the documents led to a refinement of the model with respect to the item classification.
Scoring again was a topic of the discussion. The main issues were correcting for guessing and how to deal in this respect with incorrect answers.
Another major topic in this session was to what extend existing practical issues e.g. in our institution should guide the modelling. At several points there appeared to be conflicts between theory and practice. It was decided that theory should lead.

Results and conclusions
A refined class model.
Not existing practices, but the theory should guide the modelling.
As for scoring, the literature had to be consulted.
The next session should focus on the topic of feedback.

2.1.10 Session 10 (04-12-2001)

Preparation/input
An input document with description of feedback in general.
An adjusted domain model (with respect to item types) and report on session 9.

Process
Some changes were made with respect to the modelling of item types.
The main focus in this session was on how to model feedback, based on the input document.
First the concept of feedback was analysed. It was agreed upon that feedback may contain the following components: (1) judgement, (2) comments and (3) references.
Feedback is always related to a response or set of responses. This response can be correct, incorrect or non-response. As a result also hints could be modelled as a particular type of feedback. Feedback could be given at the level of item, test-part and test.

When trying to elaborate the model some questions remained unanswered, e.g. how to model feedback in sequence items and how to deal with feedback in questions with open answers.

Results and conclusions
Progress was made in modelling feedback.
Hints are treated as a specific type of feedback.
It was still unclear how to deal with feedback on sequencing and open questions.

2.1.11 Session 11 (06-12-2001)

Preparation/input
An adjusted domain model and report on session 10 were handed out to the participants.

Process
Again the topic feedback dominated the session. The presence of other participants in this session led to additional insights concerning how to deal with feedback. One particular issue concerned feedback on open questions. This item-type may contain a standard or default feedback. As a result standard-feedback was added as an attribute to the class Open.
Another unanswered question was at which point of time feedback can be given on tests or test-parts. Four situations were identified:
1. Never.
2. Immediately after a response to an individual item.
3. After finishing a test or test-part.
4. Feedback on item level is only provided when the overall test-score is above the test-caesura.

Results and conclusions
Some changes had been made to the model related to feedback handling.
There was still no satisfying answer to how to model feedback for sequencing questions.
Issues: sequencing of items in a test, (order of) presentation of items in a test, number of responses that can be given to test-items.

2.1.12 Session 12 (13-12-2001)

Preparation/input
An adjusted domain model and report on session 11 were handed out to the participants.

Process
This was a short session focussing on feedback. A way to model feedback on sequences was worked out in this session. However, the solution looked rather complex and had not been tested on its usability.

The hint was discussed again. It was stated that hints do not contain judgements. Furthermore several hints may be provided, sometimes in a fixed, sometimes in a random order. The domain model should take these possibilities also into account.

The second part of the session concerned ‘test-instruction’, in which the conditions are specified under which a test can be taken. However, time-pressure left this issue unresolved.

Results and conclusions
An adjusted domain model with respect to feedback and hints.
This turned out to be the last session, as all human resources were needed for the specification of the new Edubox system. Unfortunately this left the domain model in an

incomplete state. Several issues remained and no validation of the model had taken place. Also the translation of the model into a binding, or XML DTD remained.

3. Domain model

3.1.1 Introduction

The following sections will give an overview description of the UML class models that have been derived from the twelve interactive sessions as described above. The goal is to describe an interaction model containing three major parts:

- static test definition model
- dynamic runtime behaviour model
- static test-item definition model.

Although the class model together with the datadictionary give an in-depth description of the model, the following paragraphs are intended as a walkthrough for the reader who is new to the model. The walkthrough will focus on the main issues of the model and is not intended to cover all details. A complete and detailed description of each class and attribute and association can be found in the data dictionary (Appendix 2). The walkthrough is divided into the three major parts of the model.

All literal class, attribute, method names are represented in italic.

3.1.2 Static test definition model

Test-definition forms the main class of the static test definition model. A *Test-definition* is the container for a number of attributes determining the behaviour of the test:

- *purpose*, a derived attribute describing the purpose of this test (what is tested). The *purpose* is derived from the purpose of all its *Testpart*s.
- *preliminary_caesura*, containing the suggested (initial) value of the cut score.
- *rating_type*, containing the representation of the rating. Typical examples are fail/pass, positive/negative, sufficient/insufficient.

On the basis of a *Test-definition* zero or more *Test*s may be derived. A *Test* is described in the dynamic runtime behaviour model. How such a *Test* should be generated/extracted from the *Test-definition* is not modelled yet.

Important for a test is the context surrounding it, as this will determine the environment in which the test is taken. Therefore a *Test-definition* contains this context via an association with the *Context* class. Next to this context there are two associations with *Feedback*, one describing the feedback in case the caesura is matched, the other relates to the *Feedback* when the caesura is not met.

A *Test-definition* is constructed of smaller parts that are represented in the model by the class *Testpart*. *Testpart* also has its own derived *purpose* attribute. The association between *Test-definition* and *Testpart* has the class *Testdefinition.testpart* associated with it containing the *weight* attribute. The *weight* attribute determines the relative weight of each *Testpart* in the whole *Test-definition*. The weight is not part of the *Testpart* itself because the same *Testpart*s can be reused in different *Test-definition*s and therefore can have a different weight in each *Test-definition*. A *Testpart* may contain an association with one ore more *Test-definition*s, meaning that a *Test-definition* can act as a *Testpart* in another *Test-definition*. A

Testpart may consist of or one or more *Item*s. A mixed association relationship of *Item*s and *Test-definition*s is not allowed.

An item is the instrument used to measure a learning objective. *Item* contains a *Context* similar to *Testpart* and *Testdefinition*. In addition *Item* has three attributes:

- *purpose*, defining the purpose of this item (e.g. what is tested by it).
- *item_scale*, defining the scale the score to the item is expressed.
- *stimulus*, the presented stimulus (e.g. a question).

The association class *Testpart.item* determines the association between *Item* and *Testpart.* This association class has a similar function as *Testdefinition.testpart*. Again the weight attribute defines the relative weight of each *Item* in the *Testpart*.
An *Item* may have one *Hint* associated with it, being the default hint for that *Item*. *Hint* is a specialisation of *Feedback*.
Finally an *Item* has a *Responsemode* associated with it. The *Reponsemode* defines how an *Item* is represented and in what form the reaction to it will be. This part of the model is further described in the static test-item definition model.

3.1.3 Dynamic runtime behaviour model

A *Test* forms the basis for the dynamic runtime behaviour model. Much of the actual behaviour of the *Test* is already modelled in the *Test-definition*. A *Test* will be generated when a test is taken by *Respondent*s. How such a *Test* is derived from the *Test-definition* is not yet part of the model presented. One could think of a *Test* as being an instance of the *Test-definition*, or in a more complex situation a subset of such *Test-definition*.

The test contains an algorithm for correcting the normalised scale for guessing. Users or groups of users may take a *Test*. For this purpose *Test* has an association with *Respondentgroup*. An instance of *Respondentgroup* can be made up of a single *Respondent* or a group of *Respondent*s. For each *Respondentgroup* there is a *final_caesura* which is based on the *initial_caesura* of the *Test-definition*.

A *Test* will have zero or more *Testresult*s associated with it. A *Testresult* can be thought of as being the container for storing the end result of the *Test* taken by a *Respondent*. Furthermore a *Test* has an association with *Item*. This relation makes up the actual *Test*, in the sense that it determines which *Item*s are selected from the associated *Test-definition*. How the *Item*s are structured into *Testpart*s can be derived form the *Test-definition*.

A *Testresult* is always associated with one and only one *Test*. Furthermore, each *Testresult* is also associated with one and only one *Respondent*. A *Respondent* however may or may not have a *Testresult*. Several calculations may be performed on the *Testresult*:

- *grade*
- *normalise_score*
- *score_corrected_for_guess_chance*
- *testscale_score*.

The *Respondent* has an association with all the *Item*s that have been presented to him in any *Test*. This association is important when tracking the *Item* history for each individual *Respondent*. Finally, each *Respondent* has zero or more *Response*s.

A *Response* is associated with the *Item* it is a response to. The *Response* keeps track of the *Respondent*'s *itemscore* for that *Item* and the *reaction* (e.g. answer) of the *Respondent* on the *Item*. Furthermore, it is possible to calculate the normalised score of the item for the

Respondent. The weight factors for calculating the normalised score can be derived from the *Test-definition* via the *Test*.

Each response may lead to a number of *Hint*s. These *Hint*s are not defined in the *Test-definition* because they are not pre-determined when designing the *Test-definition*, but are created in reaction to that *Response* at runtime. One could think of a reaction of a tutor to a response. Besides the association with *Hint*s, there is also an association with a *Verdict*.

A *Response* may have one association with a special form of *Feedback*, called *Verdict*. The *Verdict* encompasses a *judgement* about the *Response* being correct or incorrect. Typically such *Verdict*s are given in situations where there is no standard method of determining a verdict, like e.g. oral examinations.

3.1.4 Static test-item definition model

The static test-item definition model focuses on the responsemode for different types of items. The model presented is in its early design stage and far from complete.
The main categories of *Responsemode* are *Closed* and *Open*. Items with a closed responsemode have only a limited or fixed number of answer options. This characteristic makes these types of items very suitable for automated processing. With some of the closed responsemodes a user has the possibility of guessing the answer. The chance the answer is guessed correctly depends on the number of possible answers. Therefore, the attribute *chance_guess_correct* contains the chance the correct answer is obtained by guessing. Responses to items with an open responsemode are products created by the respondent that cannot be selected from a limited set of answers. It is very hard to process these types of items in an automated manner. The attribute *scoring_procedure* contains a prescription (e.g. template) how such items should be scored. Furthermore, an *Open* responsemode item may have a *default_feedback,* which will be given regardless of the response to the item.

Closed responsemode items can be divided into three main types:

- *Selection*: at least 2 *Choice*s are presented to the *Respondent* to choose from. A *Choice* may be correct or not. This is represented by the *correct* attribute of the *Choice* class. Because more than one choice can be correct, *Selection* has an attribute *required_number_of_correct_responses* to indicate how many correct responses have to be provided to have a 100% correct score. A *Choice* can have at most two specialisations of *Feedback* associated with it, the first being a *Hint* and the second being a *Verdict*. Selection is divided into two categories:
  - *True-False*: the responses on the stimulus situation may be generated from one or more *Statement*s. All possible combination of the statements and their negations form the *Choice*s. E.g. when there are two statements the result would be:
    - Statement 1 is true and Statement 2 is false.
    - Statement 1 is true and Statement 2 is true.
    - Statement 1 is false and Statement 2 is true.
    - Statement 1 is false and Statement 2 is false.
  - *Multiple-choice*: a multiple-choice item has a number of *Choice*s associated with it of which some may be correct and others may be incorrect. If the *required_number_correct_responses* is greater than one, we are effectively dealing with a multiple response item.
- *Combination*: the answers are constructed of matched pairs or classification(s). This is achieved by having an association with two or more *Topic*s and one or more *Category*, thus forming a potential matrix of *Topic*s and *Category*(ies) with a minimum size of 2 by 1. The *type* attribute of *Combination* indicates if the expected *Responsemode* is a match or classification. The major differences between the two types are the dimensions of the

matrix and uniqueness of the association between *Topic* and *Category*. The association class *Pair* maps the *Topic* and *Category*(ies) which each other. The attribute *true* indicates if this combination is 'correct' or not. If the *type* of *Combination* is match, there should be one and only one *Category* for each *Topic* and vice versa. If the *type* of *Combination* is classify then each *Topic* has one ore more *Category*(ies). A *Feedback* may be associated with each *Pair*.

- *Sequence*: the answers are constructed of re-ordered lists of the answer options presented. For this purpose a *Sequence* is made up out of 2 or more *Element*s. Each *Element* is aware of its successor if there is one. Only the last Element does not have a successor. Furthermore a *Sequence* has one or more *Series* associated with it. A *Series* itself has a two or more *Element*s associated with it, forming a subset of all *Element*s associated with *Sequence*. The *Element*s themselves are aware of their *Sequence* using the *next* association. In principle a *Series* is a correct sequencing of the *Item*. A *Feedback* may be associated with a *Series*.

Open responsemode items are divided into four categories:

- *Fill-in-the-blanks*: for this item class the respondent has to complete a phrase by entering the missing words or numbers. The attribute *number_of_blanks* contains the number of blanks in the Item;
- *Performance*: for this item part of the *Response* is the process how products like journals, essays, etc. are produced;
- *Short-answer*: the *Response* only consists out of a short answer (max. 10 words or so);
- Essay: the *Response* consists of an essay of multiple paragraphs.

4. Conclusions and discussion

4.1 Designing using UML

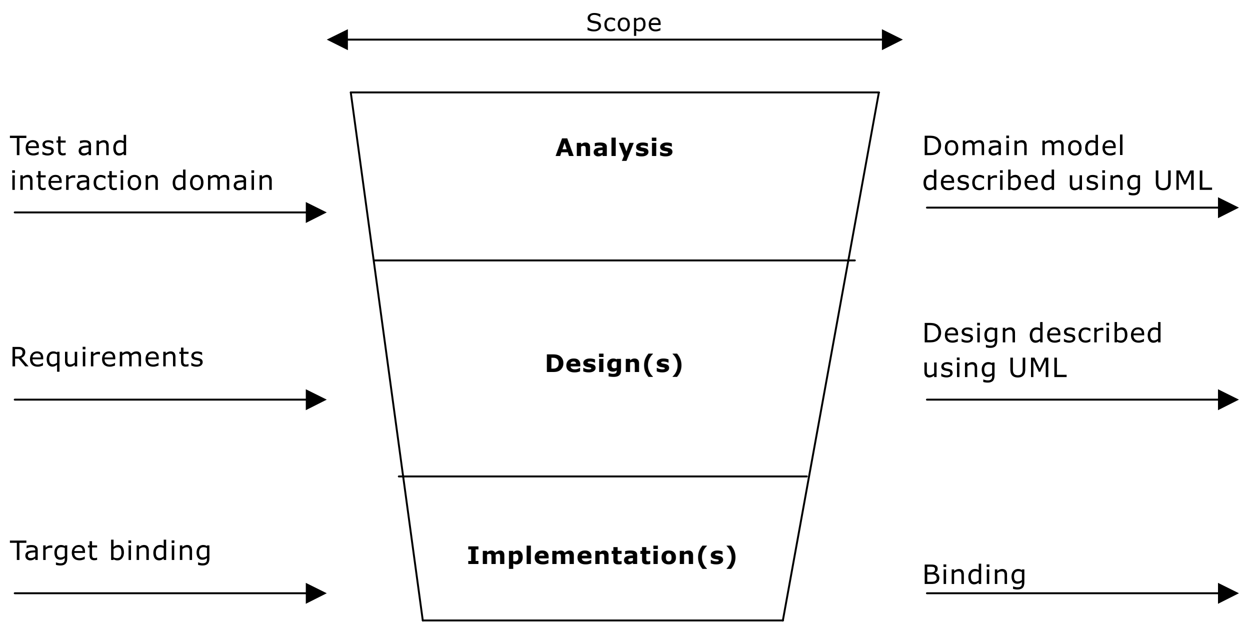
Reflecting on the sessions as described in chapter 2 of this report, the process resulting in the domain model as described in chapter 3 appears to consist of three main parts:

1 The first three sessions were used to *define the scope* of the model, to produce a basic model for interactions and to determine definitions.

2 In de following sessions (4 to 6) this *basic model was refined*, the *definitions became more or less fixed* and described in a data dictionary and the model was tested (checked) with a case study.

3 In the last sessions the *test-items were modelled* and a start was made with the modelling of *feedback*.

One of the problems encountered very early on in the process was determining the scope. It proved to be very difficult to state the exact boundaries of the relevant domain. This was probably strengthened by the fact that the team had a secondary goal in mind. They wanted to have a kind of reference model as well as an outcome of this process, which eventually could be used to analyse emerging standards in a structured and controlled manner. Using this reference domain model would make it easy to determine for example which parts of the domain are covered by e.g. IMS-QTI and which parts are not. Depending on the needs identified, this could lead e.g. to improvement suggestions or to a new binding.


Scope
Test and interaction domain
Analysis
Domain model described using UML
Requirements
Design(s)
Design described using UML
Target binding
Implementation(s)
Binding

The figure above shows this approach. From top to bottom there are three phases being analysis, design and finally implementation. The width of each parallelepipidum represents the scope covered by each layer. On the left hand side one can see the input needed to come to the output at the right hand side. In order to use the analysis for verification and creation of multiple designs, the scope must be wider than that of the individual design. The same applies to the design scope itself when deriving multiple implementations (bindings) for it, each with their own limitations.

The described approach has the drawback that the scope determination of the domain analyses is not very strict and this became obvious during the initial sessions. The advantages however are clear as well. These means that:

- The analysis can be used for several purposes, which in the longer term will be economic.
- Discussions in each design phase will be cleaner due to the fact that the issues regarding the domain, design and implementation are clearly separated.
- Adoptions in standards and bindings will require less work because the analysis and the design will not be influenced by these and can be reused.

In conclusion, it is fair enough to say that UML is a powerful tool to support communication. The fact that we used it for the first time has had its drawback on the project. Things took more time because of this. This is not caused by the difficulty of the language itself but by the fact that UML forces one to work in a very structured and systematic way. It forces one to be very explicit about what is meant. The sessions revealed that several times experts did not agree and there was confusion about definitions and differences in opinion sometimes leading to unnecessary long conversations and spending too much time on details. Uncertainty about the scope of the project in the starting phase created the opportunity to talk about everything relevant in the domain.

4.2 The current domain model and steps to be taken

As stated in the introduction section the current domain model has not been finished yet and, seen in the perspective of the overall aims of the project, is an interim product. What remains to be done is:

1. Finish the domain model.
2. Gather use cases.
3. Validate the models by creating object models based on the cases to be gathered.
4. Refine the model based on the results.
5. Externally validate the model (CITO).
6. Translate the model into an XML-binding (DTD and/or schema).
7. Dry-test the binding and if necessary adjust it.
8. Implement the binding in an authoring environment.

Appendix 1: Explanation of UML terms

Object
An object is an independent unit that can exhibit certain behaviour (functions, operations) and represent a certain state (static properties or data). An instance of a class.
Notation: rectangle with one or two compartments. Top one contains name of object and class, all underlined. Second compartment contains attributes and values, type may be omitted.

Responsibility
Every object has its own responsibility. It should perform all actions required by that responsibility without fault. The actions that an object can perform are an indication of its responsibility

Class
A class is a description of objects with the same features, i.e. all properties and operations for every object in the class are the same.
Notation: box with 3 compartments. Top contains name of class, it can contain a property list {} and a stereotype. Second compartment contains attributes, third contains operations. Name of class starts with capital, is centred and bold.

Object identity
Every object has a unique identity. Two instances of the same class with the same state are still different identifiable objects.

Attribute
An attribute is a named property of an object. Anything that is worthwhile to keep up to date could be an attribute. Attributes represent the state of the object. The values are a precise representation of the state of an object at a certain moment.
Notation: visibility name : type-expression = initial-value{property-string} Specified in second compartment. Class attributes are underlined. Name starts with lowercase, left justified. Initial value and property-string may be omitted.

Methods and operations
An operation is a service provided by an object.
A method is the implementation of an operation.
The difference between both is that the requesting object needs to know the operations it can call, but does not need to know the method that is executed to perform the requested operation. An operation usually gives information on the state or transforms the state.

Operation notation
visibility name (parameter-list) : return-type-expression {property-string}

Class attributes and operations
Class attributes are attributes whose value is common to a class rather than to a particular instance(e.g. defaults and constants).
Class operations are operation on a class (e.g. create a new instance).
Notation: underlined.

Association
An association is a structural relationship between the instances of two or more classes.

A link is a relationship between objects. Is an instance of an association.
Notation: solid line from one class to another. May have a name.

Association role
Indicates the role played by the class attached to the end of the path.

Association class
An association can be so meaningful that it can be represented as a class. Connected to the association via dashed line.

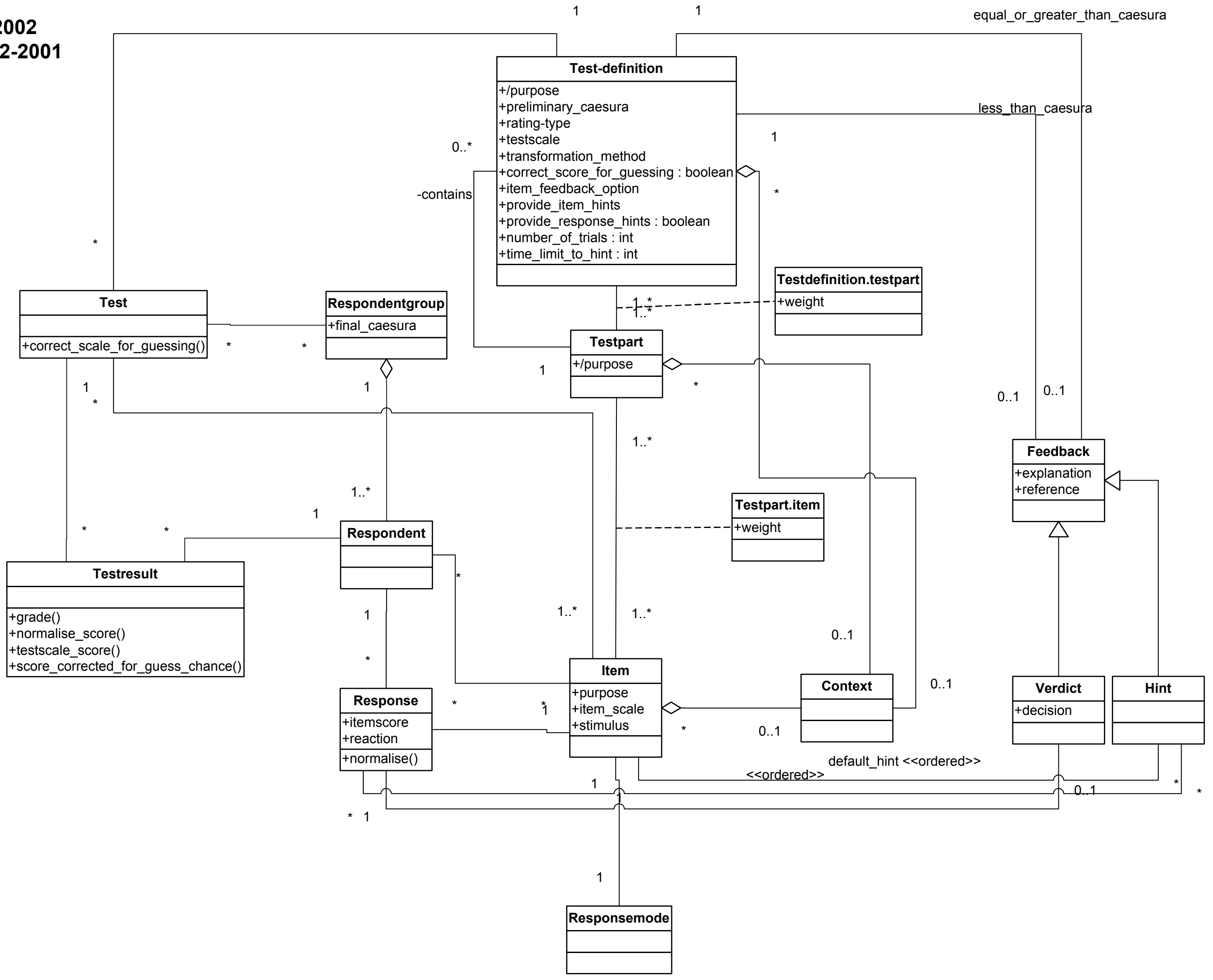
Aggregation/Composition
An object can be composed of other objects.
Composition can be nested. If you remove the whole, you remove the parts (lifetime dependency). Notation: filled diamond at 'whole'-end.
Aggregation. Parts can exist without the whole. Notation: open diamond.

Inheritance
Commonalties are defined at higher level: superclass.
More specific classes are subclasses.
A subclass inherits all features from its superclass.
At any place where you can use an instance of a class you can always use any instance of any subclass.
Notation: open arrow pointing to superclass

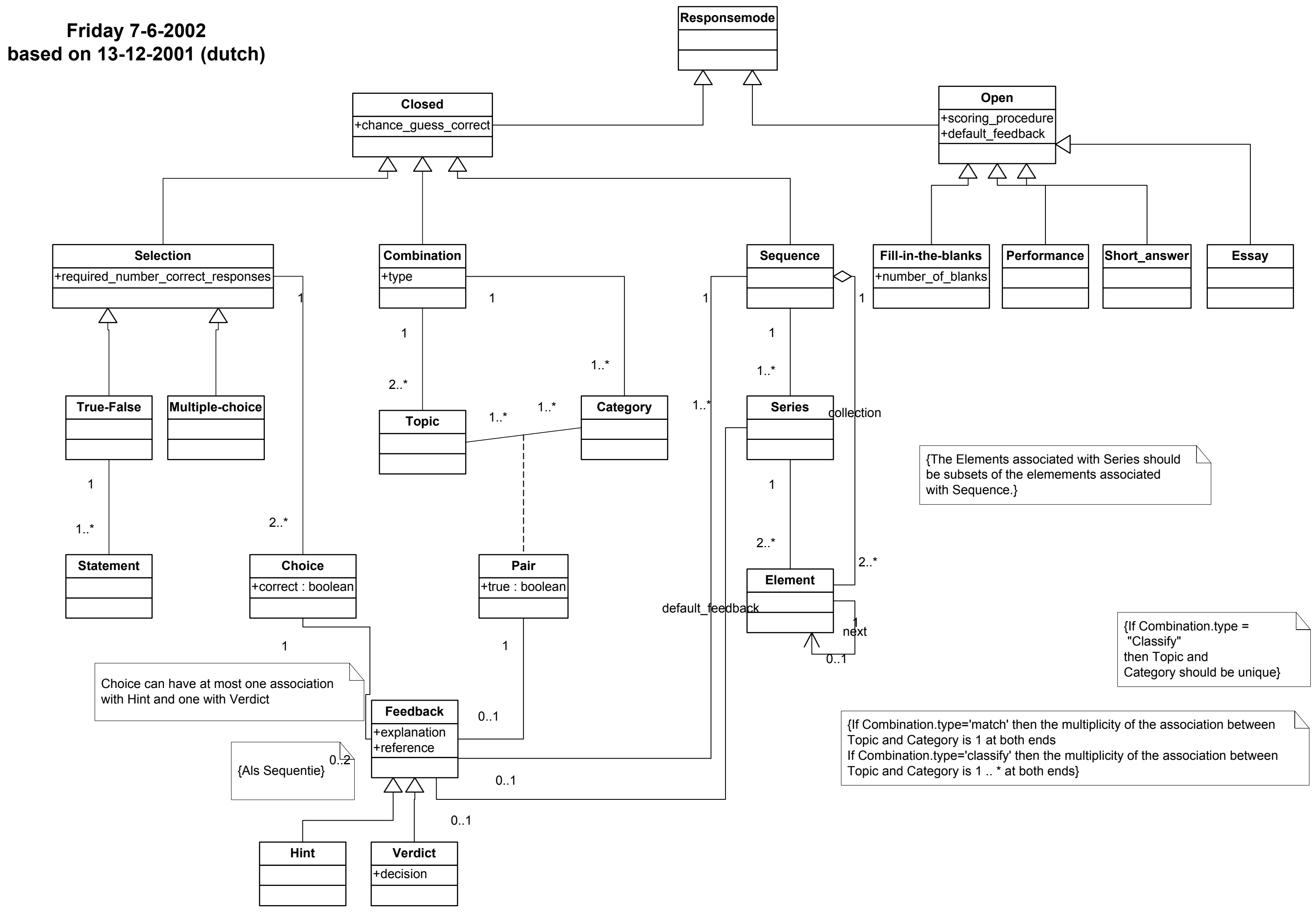
Dependency relationship
Is a relation between model-elements, not between the instances. Indicated by a dashed arrow from the depending element to the dependant element.

Appendix 2: Domain model reached on 13-12-01

Friday 7-6-2002
based on 13-12-2001
(dutch)



Friday 7-6-2002
based on 13-12-2001 (dutch)


Responsemode
Closed
+chance_guess_correct
Open
+scoring_procedure
+default_feedback
Selection
+required_number_correct_responses
Combination
+type
Sequence
Fill-in-the-blanks
+number_of_blanks
Performance
Short_answer
Essay
True-False
Multiple-choice
Topic
Category
Series
collection
Statement
Choice
+correct : boolean
Pair
+true : boolean
Element
default_feedback
next
Feedback
+explanation
+reference
Hint
Verdict
+decision
1
2..*
1..*
0..1
0..2
Choice can have at most one association with Hint and one with Verdict
{Als Sequentie}
{The Elements associated with Series should be subsets of the elemements associated with Sequence.}
{If Combination.type = "Classify" then Topic and Category should be unique}
{If Combination.type='match' then the multiplicity of the association between Topic and Category is 1 at both ends
If Combination.type='classify' then the multiplicity of the association between Topic and Category is 1 .. * at both ends}

Appendix 3: Datadictionary

Class Association	Attributes	Operations	Definition	Alias
Test-definition			A test-definition is the blueprint of weighed testparts, from which one or more assessments can be determined.	
	/purpose		Derived attribute that represents the purpose of the test-definition. The purpose is composed of the purposes of the individual testparts, which in turn are composed of the purposes of the individual items.	
	+preliminary_caesura		The preliminary caesura is the initial value on which the cut score is based. The caesura can be adjusted for every test, when necessary. The caesura value is a value on the test score scale. A cut score is a specified point on a score scale, such that scores at or above that point are interpreted differently from scores below that point.	cut-off score cut score
	transformation_method		Attribute holding In the transformation method the way to convert a normalised score to the testscale, is laid down. A major distinction is 'linear' vs. 'non-linear'. The latter needs to be expanded further. The transformation method contains the formulae according to which the derived score is calculated.	
	testscale		Attribute testscale is the scale that has to be used when converting from normalised score to the score on the testscale. Examples of testscales: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 or A, B, C, D, E, F or 0 t/m 15.	
	correct_score_for_g		A boolean attribute indicating whether the score needs to	

Class Association	Attributes	Operations	Definition	Alias
	uessing		be corrected for the effects of guessing.	
	rating_type		Attribute rating-type is the representation of the rating/grade. Valid types can be 'pass/fail', 'positive/negative', 'sufficient/insufficient', etc, and always occur as tupel.	
	item_feedback_option		Attribute indicating when feedback for all items in a testpart should be given. Possible values are: - none: no feedback - immediately: feedback is provided after every item has been answered - after completion of (sub)test: feedback is given after all items in a (sub)test have been answered or when the (sub)test has been submitted. (It is possible to complete or submit a test without answering every item.) - after score above caesura: feedback on item level is provided only when the score is above the caesura (i.e. answer key is released only then).	
	provide_item_hints		Attribute indicating when hints at item level are given. Values are: - none - on request - after time limit - both on request as after time limit	
	provide_response_hints		Boolean attribute indicating whether hints are given in reaction to a response.	
	number_of_trials		Attribute indicating how often a question may be answered.	
	time_limit_to_hint		Attribute holding time in milliseconds that need to lapse, before a hint is shown, when hints are displayed automatically. This attribute is not needed when no hints are provided.	

Class Association	**Attributes**	**Operations**	**Definition**	**Alias**
Testpart			A testpart is a (logical) grouping of components of a test. The constraint that a testpart can consist only of one or more items or one or more testdefinitions applies, because both testdefinitions and items have weights.	
	/purpose		A derived attribute representing the objective of a testpart composed of the sum of objective of all children in the hierarchy.	
Testdefinition.testpart			Association class determining the weight of the testparts in the testdefinition. The weight is given in the attribute weight. This is a relative measure from all testparts at the same level.	
	weight		Attribute representing the relative weight of the testpart in the testdefinition.	
contains			Association between testpart (1) and testdefinition (0..*), named 'contains', indicating that every testpart can contain testdefinitions.	
Testpart.item			This association class determines the relative weight of the individual items in a testpart. This is noted in the attribute weight, which represents a relative weight of the total weight of all items in the testpart.	
	weight		Attribute containing the relative weight.	
Item			An instrument to measure a learning objective. An item is characterised by a context, question and responsemode. The conversion of the measurement into a discrete value is determined via the scale.	
	purpose		Attribute containing the description of the intended measurement of the item.	
	stimulus		Attribute holding the request which directs the efforts of the user to formulate a response.	Question
	item_scale		Attribute containing the scale in which the score of the respondent is expressed.	

Class Association	Attributes	Operations	Definition	Alias
Context			The environment in which an item, testpart or testdefinition is being taken. The context largely determines the response of the user. Every item has a context. This context can be linked directly to the item or be inherited from testpart or testdefinition.	
Responsemode			Defines the expected form of reaction by a user. Major categories are open response and closed response.	
Closed			Closed response allows the user to select from an enumerated set of answers.	Closed response
	chance_guess_correct		Attribute holding the chance that the correct answer is obtained via guessing.	corrective scoring
Selection			An item for which answer(s) consists of a choice from an enumerated set.	
	required_number_correct_responses		Attribute holding the number of correct answers which has to be provided to obtain a 100% score, when multiple correct answers are possible. In case of single response items, the number is 1, with multiple response items more than 1 is possible.	
Choice			Possible response a participant might select. There should be a minimum of 2. Choices contain correct answer(s) and distracters.	
	correct		Boolean (right/wrong) attribute indicating whether the choice is right or wrong.	
True-False			A response style where the participant indicates whether a statement is correct or incorrect.	
Statement			A declaration. Minimum of 1	
Multiple-choice			A response mode where the participant selects a choice as being the correct answer.	
Combination			An item type for which the answer consists of matched	Match

Class Association	**Attributes**	**Operations**	**Definition**	**Alias**
			pair(s) or classification(s).	Classify Classify disjunct
	type		Attribute indicating the type of combination: match or classify.	
Topic			The elements, including their description, of the first (given) category. There are at least 2 Topics in a Series. Every Topic is related to 1 Category and every Category is related to 1 Topic. A multiplicity at both ends of the association between Topic and Category of 1 implies disjunct classified items There are 2 constraints. - If Combination.type='classify', then the names for Topic and Category have to be unique. - If Combination.type='match', then multiplicity of association between Topic and Category at both ends is 1. If Combination.type='classify' then the multiplicity of the association between Topic and Category is 1..* (at both ends) (or 1 in case of disjunct classify).	
Category			Second category. A combination consists of 1 or more categories. There is a constraint in case of classify items, where names of Topic and Category have to be unique. This does not apply to matching items. A Topic can belong to multiple categories and every Category can consist of multiple topics.	
Pair			Association class used to determine whether pairs (Cartesian product) to be made are correct	
	true		Boolean attribute indicating whether the pair is correct.	
Sequence			A response mode where the participant has to order the	

Class Association	Attributes	Operations	Definition	Alias
			answers.	
Series			A succession of elements as part of a Sequence. A Sequence consists of 1 or more Series; every Series belongs to 1 Sequence. Every Series consists of a minimum of 2 Elements. Every Element belongs to 1 Series.	
Element			Part of the order that must be arranged in a Sequence item. There is an association 'next' which determines the next Element. Every Sequence consists of a (aggregate) collection of 2 or more Elements, determined via the association 'collection'.	
next			Association with Element to determine the next Element.	
collection			Association between Sequence and Element indicating that there may be a collection of possible combinations.	
Open			Items with a response mode in which participants must create their own product or response rather than choose from an enumerated set.	open response construct response
	scoring_procedure		Attribute holding the criteria to be used when evaluating the responses. Is related to feedback model.	scoring formula scoring rubric answer key key
	default_feedback		Attribute holding feedback that will be provided regardless of the response. This feedback is created in advance. There is no judgement passed whether the response is correct or incorrect.	
Fill-in-the-blanks			A response mode where the participant completes a phrase by entering a word(s) or number(s).	
	number_of_blanks		Attribute holding the number of spaces which are left	

Class Association	**Attributes**	**Operations**	**Definition**	**Alias**
			blank.	
Performance			Type of item in which the response mode holds the process. If the process results in products, like journals, essays, then these products often are evaluated using different criteria than used for e.g. essays.	
Short_answer			Item type where requested response mode consists of a short answer (max. 10 words).	
Essay			Item type where requested response mode consists of an essay (multiple paragraphs).	
Response			The registered reaction of a user to an item, including his score on the item_scale.	Answer
	reaction		Attribute holding the registered reaction of the user to an item.	
	itemscore		Attribute holding the score of the reaction on the item_scale.	
		normalise()	Operation that normalises the score value. A normalised score is necessary to compare scores of individual items. The following provides an example of a possible normalisation. It assumes a linear item_scale and a linear normalised scale. *min_value = minimum value on item scale* *max_value = maximum value on item scale* *normalised_score = (itemscore / (max_value – min_value) * 100*	scale score derived score
Feedback			Feedback is term used when stimulus is given to a participant according to their responses. Feedback can be provided at item, testpart and test level. Feedback can occur in two forms. Positive feedback is given after correct answers, hint is provided after incorrect and correct answers or when no answer has been given. This implies 2 specialisations of the class Feedback: Hint	

Class Association	Attributes	Operations	Definition	Alias
			and Verdict. These classes are used in both parts of the UML model (Test-definition and Responsemode). In the upper part, feedback occurs at the level of test and response, in the lower part at item level. Feedback at test level is given based on the caesura. Every Feedback belongs to 1 Test-definition; every Test-definition has 0 to 1 Feedback. The association equal_or_greater_than_caesura applies to Feedback on tests that have been scored successfully; the association 'less_than_caesura' applies to test that have been failed. For open items, Feedback occurs via Hint or Verdict to the Response. Every Response has 0 or 1 Verdict; every Verdict belongs to 1 Response. Every Response can have multiple, ordered Hints. These Hints can be inter-dependent. Feedback at this level is constructed only after Response has been given. It is also related to the scoring_procedure (attribute of class Open). There is an order association name 'default_hint' between Item and Hint, which is given as expansion of the stimulus-situation. So, this hint is not dependent on response. In the lower part of the UML model (Responsemode) Feedback occurs in several manners.	

Class Association	Attributes	Operations	Definition	Alias
			For Selection, there is an association between Choice and Feedback. Every Choice has 0 .. 2 Feedback in the form of either Hint or Verdict; every Feedback belongs to 1 Choice. A constraint is applied: there can be at max 1 Hint plus 1 Verdict, but not 2 Hint or 2 Verdict. For Combination, Feedback is defined via the association class Pair, indicating that Feedback is possible both for correct as incorrect pairs. Every Pair has 0 ..2 Feedback, as Hint or Verdict, every Feedback belongs to 1 Pair. For Sequence, default_feedback occurs on the whole sequence, but can also happen as Hint or Verdict on Series.	
	explanation		Attribute holding the actual description of the feedback. This could be an explanation why an answer is correct or incorrect, or could contain an example of how to complete the answer.	
	reference		Attribute holding a reference to the location where the explanation is given, e.g. textbook.	
equal_or_greater_than_caesura			The association equal_or_greater_than_caesura applies to Feedback on tests that have been scored successfully.	
less_than_caesura			The association 'less_than_caesura' applies to Feedback on tests that have been failed.	
default_feedback			Association between Sequence and Feedback, providing default feedback on the whole sequence, without commenting on parts of the Sequence. This Feedback is defined in advance.	
Hint			Extension of the stimulus situation, often because no or an incorrect answer has been given.	
default_hint			Association between Item and Hint for a default hint which is independent of given response.	
Verdict			Feedback which encompasses a judgement about the	

Class Association	Attributes	Operations	Definition	Alias
			answer being correct or incorrect.	
	decision		Attribute detailing whether answer is correct or incorrect, or right or wrong, etc.	
Test			A test as taken by respondents. A test is derived from a Test-definition, which forms a blueprint.	
		correct_scale_for_guessing()	This operation calculates the corrected normalised scale incorporating the guess factor. The algorithm is: *Determine the guess chance for every item. This chance is 0% for closed items.* *Determine the guess chance for a testpart by weighing the guess chance of its items.* *Determine the guess chance for the test-definition by weighing the guess chance of the testparts.* The result is the starting point (0) of the normalised scale.	corrective scoring
Respondent			A persons participating in a test by answering questions.	
Respondentgroup			A population of respondents with specific properties. It contains 1 or more Respondents.	
	final_caesura		Attribute final_caesura is the caesura that is applied to the Respondentgroup. Initially the final_caesura equals the initial_caesura of the Test-definition. The final_caesura is a value on the testscale.	
Testresult			The Testresult contains all actions by a Respondent. Every Testresult belongs to 1 Respondent.	
		score_corrected_for_guess_chance()	Operation to calculate a normalised score corrected for chance of guessing the correct answer. The algorithm is: *Determine for every selected response item which has been answered incorrectly by the respondent, the chance*	

Class Association	Attributes	Operations	Definition	Alias
			of guessing the correct answer (this chance is 0 for correctly answered items). *Determine the normalised chance of guessing correctly for the test, by weighing above-mentioned values, according to the score-procedure.* *Subtract the chance of guessing from the normalised score.* *If this results in a negative value, the score equals 0.*	
		normalise_score ()	Operation to calculate the normalised score for the test. The algorithm is: *Determine the items of the test.* *Calculate the normalised score for every item_score.* *Determine the weighed normalised score of every testpart using the weight of every item.* *Determine the weighed score of every test-definition on the basis of each testpart.*	
		testscale_score( )	Operation to transform the normalise_score() or the score_corrected_for_guess_chance() to the testscale as defined in the Test-definition. When attribute correct_score_for_guessing is true score_corrected_for_guess_chance() is used. Then the normalised scale is also adjusted by means of score_corrected_for_guess_chance(). If attribute correct_score_for_guessing is false, the normalised score is used, but not the corrected scale. The algorithm is: *is Test-definition.correct_score_for_guessing true*	

Class Association	**Attributes**	**Operations**	**Definition**	**Alias**

			than
			  is Test-definition.transformation_method 'linear'
			  then
			    increments = number of discrete increments on Test-definition.testscale
			    correction = Test.correct_scale_for_guessing()
			    max_score = maximum on the normalised score scale
			    *testscalescore = increments / (maximum – correction) * Testresult.score_corrected_for_guess_chance()*
			  else
			    ?????
			   si
			
			else
			  is Test-definition.transformation_method 'linear'
			  then
			increments = number of discrete increments on Test-definition.testscale
			    max_score = maximum on the normalised score scale
			    *testscalescore = increments / (maximum – correction) * Testresult.normalised_score()*
			  else
			    ????
			  si
			si	

		grade()	Operation to determine the value of the score on the basis of the Testresult.testscale_score() and Respondentgroup.final_caesura. The grade is determined by testscale_score being above or	

Class Association	Attributes	Operations	Definition	Alias
			below the caesura, and is expressed in the terms defined in Test-definition.rating_type.	

OTEC 2002/25