

Ricardo Alexandre Peixoto de Queirós

A framework for practice-based learning applied to computer programming



Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2012

Ricardo Alexandre Peixoto de Queirós

A framework for practice-based learning applied to computer programming



*Tese submetida à Faculdade de Ciências da Universidade do Porto
para obtenção do grau de Doutor em Ciência de Computadores*

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2012

Resumo

Aprender através da prática é fundamental para alcançar um melhor desempenho em domínios complexos. No entanto, a aprendizagem só é eficaz se os estudantes tiverem à sua disposição uma grande variedade de exercícios que cubram todo o programa do curso e se as suas soluções são prontamente avaliadas e se é dado o feedback apropriado.

Atualmente, o processo de ensino-aprendizagem em domínios complexos, tais como a programação de computadores, é caracterizado por turmas grandes e currículos extensos. Esta situação representa uma carga de trabalho grande para os professores responsáveis pela criação, entrega e avaliação dos exercícios.

O objetivo principal deste trabalho é promover a aprendizagem baseada na prática em domínios complexos. Este objectivo é conseguido através do desenho de uma *framework* de *e-learning* - chamada de *Ensemble* - caracterizada como uma ferramenta concetual para facilitar a interoperabilidade técnica entre os sistemas e serviços em domínios com avaliação complexa. Estes domínios precisam de uma diversidade de ferramentas, desde os ambientes onde os exercícios são resolvidos, até aos avaliadores automáticos fornecendo *feedback* sobre as tentativas dos alunos, não esquecendo a criação, gestão e sequenciação de exercícios.

A *framework Ensemble* é usada num domínio específico - programação de computadores. Uma instância dessa *framework* aborda as questões relacionadas com a interoperabilidade do conteúdo e da comunicação. As questões de conteúdo são resolvidas com um formato padrão para descrever exercícios de programação como objetos de aprendizagem. A comunicação é alcançada com a extensão das especificações existentes para a interação dos sistemas tipicamente encontrados num ambiente de e-learning como os sistemas de gestão de aprendizagem, repositórios de objetos aprendizagem e sistemas de avaliação. Alguns desses sistemas e serviços foram criados de raiz e outros foram adaptados para suportar as novas especificações de conteúdo e comunicação.

A fim de avaliar a aceitação da solução proposta uma instância da *framework Ensemble* foi validada numa experiência em sala de aula com resultados animadores. Os resultados permitem concluir que o uso desta *framework* tem um impacto positivo sobre a aquisição de habilidades complexas, tais como a programação de computadores.

Abstract

Learning through practice is crucial to acquire a complex skill. Nevertheless, learning is only effective if students have at their disposal a wide range of exercises that cover all the course syllabus and if their solutions are promptly evaluated and given the appropriate feedback.

Currently the teaching-learning process in complex domains, such as computer programming, is characterized by an extensive curricula and a high enrolment of students. This poses a great workload for faculty and teaching assistants responsible for the creation, delivering and assessment of student exercises.

The main goal of this work is to foster practice-based learning in complex domains. This objective is attained with an e-learning framework - called Ensemble - as a conceptual tool to organize and facilitate technical interoperability among systems and services in domains that use complex evaluation. These domains need a diversity of tools, from the environments where exercises are solved, to automatic evaluators providing feedback on the attempts of students, not forgetting the authoring, management and sequencing of exercises.

The Ensemble framework is used on a specific domain - computer programming. A framework instance addresses the content and communication interoperability issues typically found in this domain. Content issues are tackled with a standard format to describe programming exercises as learning objects. Communication is achieved with the extension of existing specifications for the interoperation with several systems typically found in an e-learning environment such as learning management systems, learning objects repositories and assessment systems. Some of these systems were created from scratch and others were adapted to support the new content and communication specifications.

In order to evaluate the acceptability of the proposed solution an Ensemble instance was validated on a classroom experiment with encouraging results. They support the conclusion that the use of this e-learning framework for the practice-based learning has a positive impact on the acquisition of complex skills, such as computer programming.

Résumé

L'apprentissage par la pratique est essentielle pour réaliser de meilleures performances dans des domaines complexes. Néanmoins, l'apprentissage n'est efficace que si les élèves ont à leur disposition un large éventail d'exercices qui couvrent tous les syllabus de cours et si leurs résolutions sont rapidement évalués et être un feedback approprié.

Actuellement, le processus d'enseignement-apprentissage dans des domaines complexes, comme la programmation informatique, est caractérisé par le programme d'enseignement exhaustif et par le taux élevé d'inscriptions des étudiants. Cela pose une charge de travail idéal pour les enseignants et des assistants responsables de la création, la prestation et l'évaluation des exercices des élèves.

L'objectif principal de ce travail est de favoriser l'apprentissage basé sur la pratique dans des domaines complexes. Cet objectif est atteint par la conception d'un cadre d'e-Learning - appelé Ensemble - comme un outil conceptuel pour faciliter l'interopérabilité technique entre les systèmes et services dans les domaines qui utilisent l'évaluation complexe. Ces domaines ont besoin d'une diversité d'outils, des environnements où les exercices sont résolus, les évaluateurs automatiques fournissent une rétroaction sur les tentatives d'étudiants, sans oublier la création, la gestion et le séquençage d'exercices.

Le cadre Ensemble est utilisé sur un domaine spécifique - la programmation informatique. Une instance de cadre porte sur le contenu et les questions d'interopérabilité de communication on trouve habituellement dans ce domaine. Questions de contenu sont cloués avec un format standard pour décrire les exercices de programmation comme des objets d'apprentissage. L'interopérabilité est réalisée avec le développement ou l'adaptation des systèmes et services pour la gestion du cycle de vie des exercices, à savoir leur création, de stockage, de conversion et d'évaluation.

Afin d'évaluer l'utilité et la convivialité de la solution proposée une instance Ensemble a été validé sur une expérience en classe avec des résultats prometteurs. Ces résultats appuient la conclusion que l'utilisation de ce cadre met en place les fondations de l'apprentissage fondée sur la pratique avec un impact positif sur l'acquisition de compétences complexes.

to Márcia and Gabriela

Acknowledgments

First of all, I would like to thanks my supervisor Prof. José Paulo Leal for his support over time. Throughout this thesis he helps me by sharing his knowledge, his competence, his availability and, most important, his friendship. I hope he maintains his confidence in me and we can embrace new projects soon.

I would like to add further thanks to Prof. Cristina Ribeiro for the suggestions and feedback provided in the annual meetings.

My acknowledgements to my friend Alberto Simões for helping me mostly in the LaTeX issues.

A word of appreciation goes also to my friend Mário Pinto for aiding me in the experiment conducted at ESEIG.

I would like to express my deep gratitude to my family for their love and support over time.

Last but not least, I want to thank my lovely wife Márcia for all the support that she gave me during the years I have been working on this thesis. These years were special with the birth of our beautiful daughter Gabriela. To both my endless love!

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Challenge	4
1.3	Approach	5
1.4	Contributions	6
1.5	Thesis structure	7
I	State of Art	9
2	E-Learning systems	11
2.1	E-Learning evolution	12
2.1.1	Early systems	12
2.1.2	Component based systems	12
2.1.3	E-Learning services	13
2.1.4	Specialised E-Learning Services	16
2.2	Learning Management Systems	17
2.2.1	Categories	17
2.2.2	LMS Interoperability	19
2.2.2.1	LMS selection	20
2.2.2.2	Interoperability Facets	21

2.2.3	Conclusions	25
2.3	Repository Systems	26
2.3.1	Categories	27
2.3.2	Repository Interoperability	27
2.3.2.1	Software for Digital Libraries	28
2.3.2.2	Software for Learning Objects Repositories	29
2.4	Assessment Systems	31
2.4.1	Evolution of Assessment Systems	32
2.4.2	Recent Surveys	33
2.4.3	Assessment System Interoperability	35
2.4.3.1	Programming Exercises	36
2.4.3.2	Users	38
2.4.3.3	Assessment results	40
2.4.4	Conclusions	41
2.5	Summary	42
3	E-Learning standards	45
3.1	Frameworks	47
3.1.1	Abstract Frameworks	49
3.1.1.1	IEEE Learning Technology Systems Architecture	50
3.1.1.2	Open Knowledge Initiative	51
3.1.1.3	IMS Abstract Framework	51
3.1.2	Concrete Frameworks	52
3.1.2.1	Open University Support System	53
3.1.2.2	Schools Interoperability Framework	53
3.1.2.3	E-Framework	54
3.1.3	Comparison of E-Learning Frameworks	55

3.2	Content	57
3.2.1	Metadata	58
3.2.1.1	Dublin Core	58
3.2.1.2	IEEE LOM	60
3.2.1.3	ISO/IEC MLR	63
3.2.1.4	Other Metadata Specifications	64
3.2.2	Packaging	65
3.2.2.1	IMS Content Packaging	65
3.2.2.2	ADL SCORM	67
3.2.2.3	IMS Common Cartridge	67
3.2.2.4	Other specifications	70
3.2.3	Assessment	71
3.2.3.1	Quiz based assessments	71
3.2.3.2	Text file based assessments	72
3.3	Communication	77
3.3.1	Learning Management Systems	77
3.3.1.1	Data Integration	78
3.3.1.2	API Integration	78
3.3.1.3	Tool Integration	79
3.3.1.4	Comparison of the integration strategies	81
3.3.2	Repositories	82
3.3.2.1	Data push	83
3.3.2.2	Data pull	84
3.4	Summary	85

II	Architecture	87
4	The Ensemble E-Learning Framework	89
4.1	Architectural model	90
4.1.1	Axial systems	91
4.1.2	Core and Secondary services	92
4.2	Data Model	93
4.3	Integration Model	94
4.3.1	Text File Evaluation Service Genre	95
4.3.1.1	The ListCapabilities function	96
4.3.1.2	The EvaluateSubmission function	97
4.3.1.3	The GetReport function	98
4.3.2	Digital Repositories Interoperability	98
4.3.3	Learning Tools Interoperability	99
4.4	Summary	101
5	Specializing Ensemble to computer programming	103
5.1	Architecture	103
5.2	Data model	104
5.2.1	The life-cycle of a programming exercise	106
5.2.2	PExIL	106
5.2.2.1	Textual Elements	108
5.2.2.2	Specification Elements	109
5.2.2.3	Program Elements	113
5.2.3	Evaluating PExIL	115
5.3	Integration model	116
5.3.1	Digital Repositories Interoperability	116

5.3.1.1	Interface definition	117
5.3.1.2	XML binding	120
5.3.2	Learning Tools Interoperability	120
5.3.3	Evaluation service	122
5.3.3.1	Interface definition	123
5.3.3.2	XML binding	125
5.3.4	Workflow	127
5.4	Tools selection	129
5.5	Summary	130
III	Implementation	131
6	Learning objects repository	133
6.1	Architecture	133
6.2	Implementation details	135
6.2.1	Storage	136
6.2.2	Validation	136
6.2.3	Security	139
6.2.4	User Interface	139
6.2.5	Tests	142
6.3	Case Study: using crimsonHex as a LMS plug-in	143
6.4	Summary	145
7	Programming exercises converter	147
7.1	Exercise format conversion	147
7.1.1	Approach	148
7.1.2	Pivot format	149

7.1.3	Abstract functions	149
7.2	The BabeLO service	150
7.3	Evaluation results	152
7.3.1	Case study 1: repositories exchange	153
7.3.2	Case study 2: automatic assessment	154
7.4	Summary	155
8	Programming Teaching Assistant	157
8.1	Use Cases	158
8.1.1	Teacher	159
8.1.2	Student	160
8.2	Design	161
8.2.1	Class <code>TeachingAssistantLauncher</code>	162
8.2.2	Class <code>TeachingAssistant</code>	163
8.2.2.1	Class <code>ExerciseManager</code>	163
8.2.2.2	Interface <code>ProjectCreator</code>	164
8.2.2.3	Interface <code>ExerciseUtils</code>	164
8.3	Summary	171
IV	Validation	173
9	Ensemble evaluation	175
9.1	Evaluation Model	175
9.2	Experiment Design	176
9.2.1	Methodology	176
9.2.2	Infrastructure	177
9.2.3	Instruments and Data collection	179

9.3 Results and discussion	179
9.3.1 Usefulness	180
9.3.2 Reliability	183
9.3.3 Interoperability	184
9.4 Summary	185
10 Thesis validation	187
10.1 Hypothesis	187
10.2 Data collection and validation	188
10.3 Results and discussion	189
10.3.1 Exercises solving	189
10.3.2 Feedback	193
10.3.3 Attendance	195
10.3.4 Grades	197
10.4 Summary	198
11 Conclusions	199
11.1 Contributions	200
11.2 Opportunities for future work	201
11.2.1 Framework validation	201
11.2.2 Framework extension	202
11.2.3 Ensemble instance improvements	202
A Nielsen’s heuristics	205
B Session survey	207
C Petcha’s user manual	223
C.1 Teacher’s user manual	223

C.1.1	Launching Petcha	223
C.1.2	Creating a project	223
C.1.3	Writing a program solution	224
C.1.4	Defining the exercise statement	225
C.1.5	Defining and generating the test cases	228
C.1.6	Defining the feedback types	234
C.1.7	Packaging and deploying the exercise	236
C.2	Student manual (distributed to the students - PT)	237
C.2.1	Execução do Petcha	237
C.2.2	Criação de um projeto	238
C.2.3	Resolvendo um exercício	240
C.2.4	Testando	241
C.2.5	Submetendo uma solução	243
D	crimsonHex Core Functions	247
D.1	Register function	247
D.2	Submit function	248
D.3	Request and RequestAsset functions	248
D.4	Search function	248
D.5	Report function	250
D.6	Alert function	250
D.7	Create function	251
D.8	Remove function	251
D.9	Status function	252
	References	256

List of Tables

2.1	Features of CMS, LMS and LCMS.	18
2.2	LMS support of content standards.	23
2.3	Integration API in reference LMS.	24
2.4	Digital library software.	29
2.5	Learning Objects Repositories software.	29
2.6	Learning Objects Repositories.	31
2.7	Programming exercise facet (P-partial and F-full)	37
2.8	Users facet (P-partial and F-full)	39
2.9	Assessment results facet (P-partial and F-full)	41
3.1	E-Learning frameworks survey.	56
3.2	Profiling Types of existing LOM APs.	62
3.3	QTI compliance.	72
3.4	Textual information facet.	75
3.5	Data files facet.	75
3.6	Configuration and recommendation parameters facet.	76
3.7	Tools facet.	76
3.8	Metadata facet.	77
3.9	Comparison of e-Portfolio integration strategies	81
5.1	Textual elements.	108

5.2	Specification elements.	109
5.3	Program elements.	113
5.4	Core and extension functions of DRI.	117
5.5	Student's attempt general data.	119
5.6	Student's characteristics particular data.	119
5.7	LTI functions.	121
5.8	LTI launch parameters	121
5.9	Core functions of the Evaluation Engine.	123
5.10	REST request and response for the <code>ListCapabilities</code> function.	124
5.11	REST request and response for the <code>Evaluate</code> function.	124
5.12	REST request and response for the <code>GetReport</code> function.	125
6.1	Average function execution times per interface (in seconds).	142
7.1	PExIL coverage based on the Verhoeff model.	149
7.2	BabeLO REST API.	151
7.3	Download time (ms) and overhead of BabeLO	153
8.1	<code>ProjectCreator</code> interface methods.	164
8.2	<code>ExerciseUtils</code> interface methods.	165
8.3	Generation of input data of test cases (R=random).	168
8.4	Binding PExIL to IEEE LOM.	171
9.1	The ESEIG experiment schedule	177
9.2	Network selected systems	178
9.3	Statistical data on interoperability of the network components	184
10.1	Hypothesis on Ensemble usage.	188
10.2	Survey results averages.	189

10.3 Statistics on student participation.	190
10.4 Statistics on student participation.	194
10.5 Statistics on student attendance.	196
10.6 Statistics on student participation.	198
D.1 Core functions of the repository.	247

List of Figures

1.1	Noteflight application.	2
1.2	Thesis structure.	7
2.1	The evolution of e-learning systems.	12
2.2	Integration classic models.	13
2.3	SOA components.	15
2.4	E-Learning system types.	18
2.5	Timeline of development of major LMSs.	20
2.6	LMS usage in Portuguese higher education institutions.	21
2.7	LMS interoperability facets.	22
2.8	AMS usage.	25
2.9	Evolution of software and specifications for repositories.	28
2.10	Usage of digital library software worldwide.	30
2.11	Generations of the Assessment Systems.	34
2.12	Interoperability facets of Assessment Systems.	36
2.13	Interoperability maturity percentage level of Assessment Systems.	42
2.14	Coverage of Assessment Systems interoperability features.	43
3.1	Steps for establishing e-learning standards [VA06].	46
3.2	E-learning specifications and standards publications (1980-2010).	47
3.3	Simple Framework model [WBR04].	49

3.4	Learning Technology System Architecture [FT99].	50
3.5	OKI architecture.	51
3.6	IAF layered model.	52
3.7	SIF architecture.	54
3.8	E-Framework.	55
3.9	Evolution of e-learning Frameworks.	55
3.10	The hierarchy of elements in the LOM data model.	60
3.11	IMS CP package structure.	66
3.12	Common Cartridge Content Hierarchy.	67
3.13	IMS Common Cartridge package.	69
3.14	Data Integration.	78
3.15	API Integration.	79
3.16	IMS Full LTI.	80
4.1	EeF architectural model.	91
4.2	Structure of an IMS CC package.	94
4.3	Trends on the use of SOAP and REST web services.	95
4.4	The ListCapabilities function.	96
4.5	The EvaluateSubmission function.	97
4.6	The GetReport function.	98
4.7	The IMS DRI Specification.	99
4.8	The IMS Basic LTI Specification.	100
4.9	The IMS LTI Specification v.1.1 - integration with LIS services.	100
5.1	Overall architecture of the Ensemble instance.	104
5.2	Ensemble instance data model.	105
5.3	PExIL data model.	107

5.4	The specification element.	110
5.5	The program element.	113
5.6	Evaluation of PExIL expressiveness.	115
5.7	Network component diagram.	116
5.8	Response specification schema.	120
5.9	The reply type on the ERL specification.	126
5.10	The report type on the ERL specification.	127
5.11	Sequence diagram of the Ensemble instance.	128
5.12	The deployment architecture of the EeF.	129
6.1	UML components diagram of the crimsonHex repository.	134
6.2	Validation of XML files by a W3C XML Schema with Schematron rules [Rob02].	138
6.3	Schematron processing [Rob02].	138
6.4	crimsonHex WebManager.	140
6.5	crimsonHex plugin interface.	144
7.1	Exercise formats conversion using both approaches.	148
7.2	BabeLO architecture.	151
7.3	The Convert function.	152
7.4	Case study 1 - exchanging exercises between repositories	153
7.5	Case study 2 - automatic assessment.	154
8.1	Petcha use cases.	158
8.2	The GUI of Petcha with teacher and student modes.	159
8.3	The UML class diagram of Petcha.	161
8.4	Generation of the exercise descriptions.	166
8.5	An example of an exercise description.	167
8.6	PExIL files.	169

8.7	Structure of the IMS CC manifest file.	170
9.1	System acceptability. Adapted from Nielsen [Nie94].	176
9.2	UML deployment diagram for the Ensemble instance.	178
9.3	Results of each heuristic in the student's profile.	180
9.4	Evaluation of Petcha's flexibility.	181
9.5	Evaluation of Petcha's freedom.	181
9.6	Results of each heuristic in the student profile.	181
9.7	Classification of Pectha by students.	182
9.8	Results of each heuristic in the teacher profile.	182
9.9	Results of the Ease of Use heuristic in the teacher profile.	183
9.10	Reliability of Petcha.	183
10.1	Average and standard deviation of exercises solving by student participation.	190
10.2	Evolution in started exercises.	191
10.3	Evolution of completed/solved exercises.	193
10.4	Average of feedback received by exercise and its helpfulness.	194
10.5	Evolution of feedback and its helpfulness.	195
10.6	Overall attendance of students of both groups.	196
10.7	Evolution of attendance.	197
10.8	Average grades from both groups of students.	197
C.1	New project.	224
C.2	Confirmation of the creation of the project.	224
C.3	Coding the program solution in VSE.	225
C.4	Matrix transposition code.	226
C.5	User interface areas of Petcha.	226
C.6	The Description tab.	227

C.7 Exercise statement.	227
C.8 Data about the program solution.	228
C.9 Test cases management.	229
C.10 Description of the input data.	230
C.11 Definition of a test case (input part).	231
C.12 Definition of a test case (input part).	232
C.13 Automatic generation of test cases.	233
C.14 Test case update.	233
C.15 Invalid test case.	234
C.16 Public test case with feedback.	234
C.17 Feedback levels definition.	235
C.18 Exercise packaging.	236
C.19 Exercise deploy in crimsonHex repository.	237
C.20 crimsonHex repository.	237
C.21 Ecrã inicial do Petcha (modo aluno).	238
C.22 Criação de projeto.	239
C.23 Gestão da resolução de exercícios.	240
C.24 Enunciado do exercício.	240
C.25 Codificação da solução por parte do Aluno.	241
C.26 Gestão de testes.	241
C.27 Novo teste.	242
C.28 Execução local de testes.	243
C.29 Feedback automático providenciado pelo Avaliador.	243
C.30 Feedback automático providenciado pelo Avaliador.	244
C.31 Solução correta.	244
C.32 Feedback automático providenciado pelo Avaliador.	245

C.33 Estatísticas do exercício.	245
C.34 Resolução dos restantes exercícios.	245

List of Acronyms

ACM Association for Computing Machinery.

ADL Advanced Distributed Learning.

AMS Academic Management Systems.

API Application Programming Interface.

AS Assessment Systems.

CEN European Committee for Standardization.

CMS Content Management Systems.

DC Dublin Core.

DL Digital Libraries.

E-F E-Framework.

EE Evaluation Engine.

EeF Ensemble E-Learning Framework.

GLC Global Learning Consortium.

HTTP Hypertext Transfer Protocol.

IAF IMS Abstract Framework.

ICPC International Collegiate Programming Contests.

ICT Information Communication Technology.

IDE Integrated Development Environments.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

ILOX Information for Learning Object eXchange.

IMS CC IMS Common Cartridge.

IMS CP IMS Content Packaging.

IMS LD IMS Learning Design.

IMS LODE IMS Learning Object Discovery & Exchange.

IMS SS IMS Simple Sequencing.

IOI International Olympiad in Informatics.

ISO International Organization for Standardization.

JAR Java ARchive.

JAWS Java Web Start.

JAXB Java Architecture for XML Binding.

LAO Learning Application Objects.

LCMS Learning Content Management Systems.

LDAP Lightweight Directory Access Protocol.

LIP Learner Information Package.

LIS Learner Information Services.

LMS Learning Management Systems.

LO Learning Object.

LOM Learning Object Metadata.

LOR Learning Objects Repository.

ITI Learning Tools Interoperability.

LTSA Learning Technology Systems Architecture.

LTSC Learning Technology Standards Committee.

MLE Managed Learning Environment.

OAI-PMH Open Archives Initiative Protocol for Metadata Harvesting.

OJ Online Judge.

OKI Open Knowledge Initiative.

OpenUSS Open Source University Support System.

OSID Open Service Interface Definition.

PCMS Programming Contests Management Systems.

QTI Question and Test Interoperability.

SCORM Sharable Content Object Reference Model.

SIF Schools Interoperability Framework.

SMTP Simple Mail Transfer Protocol.

SOA Service-Oriented Architecture.

SOAP Simple Object Access Protocol.

SWT Standard Widget Toolkit.

TA Teaching Assistant.

URI Uniform Resource Identifier.

VLE Virtual Learning Environment.

WCR Web Content Resources.

WS-BPEL Web Services Business Process Execution Language.

WSDL Web Service Description Language.

Chapter 1

Introduction

"I practice until I have my life in my fingers"

Pianists' expression

For someone to acquire, improve or even maintain a complex skill it is necessary to practice it on a regular basis [GP05, Eck09]. The amount of practice required depends on the nature of the activity and on each individual. How well an individual improves with practice is directly related with its inherent capabilities, its previous know-how about the domain and the type of feedback that is available for improvement. If feedback is either non-existent or inappropriate, then the practice tends to be ineffective or even detrimental to learning.

An apt example of a complex skill is music. Learning music requires discipline and perseverance while acquiring the concept of reading scores, practising an instrument or playing with a group. In order to enhance these skills and motivate young students, instructors use e-learning, mainly in introductory courses, to make the learning of music more appealing. One good example is the NoteFlight¹ web application (Figure 1.1), a tool designed to teach music by creating, viewing, printing and hearing music notation. The tool was recently integrated² with Moodle, a popular Learning Management Systems (LMS). This integration enables instructors to create assignments (e.g. giving students a partial composition to be completed), to manually grade the student submissions and to give them feedback promptly.

Besides music, there are other areas where evaluation is a key component in practice such as management, health sciences, electronics. Playing business games in management courses, or simulating a human patient in life sciences courses, or simulating an electronic circuit in electronics courses are examples of learning processes that require the use of special

¹NoteFlight web site: <http://www.noteflight.com>

²NoteFlight demo with Moodle integration: <http://videos.noteflight.com/MoodleBasicLTI.mov>

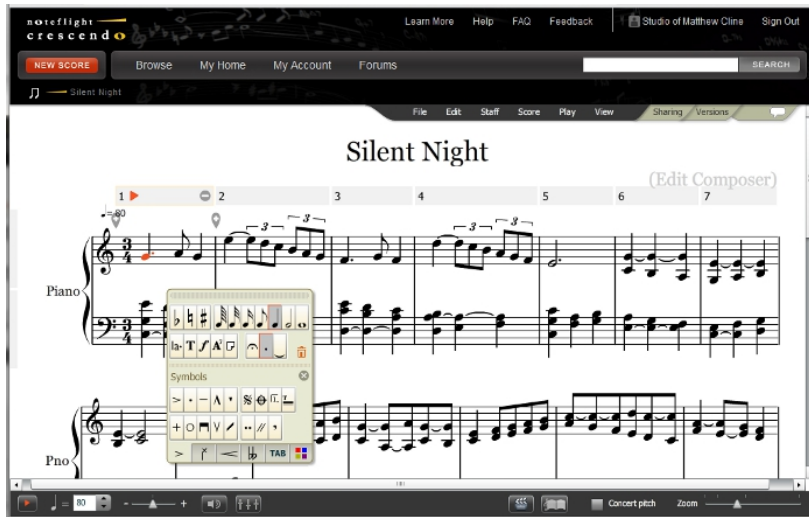


Figure 1.1: Noteflight application.

authoring, rendering and assessment tools. These tools should be integrated in instructional environments in order to provide a better learning experience. However, these tools would be too specific to incorporate in an e-learning platform. Even if they could be provided as pluggable components, the burden of maintaining them would be prohibitive to institutions with few courses in those domains.

1.1 Motivation

The motivation for this thesis comes from yet another domain with complex evaluation: computer programming. Introductory programming courses are generally regarded as difficult and often have high failure and dropout rates [AM05, OG06, RRR03]. Researchers pointed out several causes for these rates [EFMM10]. The most consensual are:

Teaching methods - lectures and programming language syntaxes [LAMJ05, SB06];

Subject complexity - learning how to program means to integrate knowledge of a wide variety of conceptual domains such as computer science and mathematics while developing expertise in problem understanding, problem-solving, unit testing and others. Additionally, students petered out when they need to understand and apply abstract programming concepts like control structures or to create algorithms that solve concrete problems [EFMM10].

Student motivation - the public image of a "programmer" as a socially inadequate "nerd" [Jen02] and the reputation of programming courses as being extremely difficult affects negatively the motivation of the students [GM07].

Many educators claim that "learning through practice" is by far the best way to learn computer programming and to engage novice students [GP05, Eck09]. Practice in this area boils down to solving programming exercises. Nevertheless, solving exercises is only effective if students receive an assessment on their work. An exercise solved wrong will consolidate a false belief, and without feedback many students will not be able to overcome their difficulties.

Assessment plays a vital role in learning [AM05]. However, automatic assessment of exercises other than multiple choice can be a rather complex task. This kind of evaluation differs significantly from evaluations supported by most LMSs, encoded in the IMS Question and Test Interoperability (QTI) specification³. The data model of QTI was designed for questions with a set of pre-defined answers and cannot handle evaluation domains with specialized requirements such as the computer programming. For instance, the assessment of programming exercises requires tests cases, program solutions, compilation lines and other data that cannot be encoded in QTI. Besides the lack of a formal description for programming exercises, the interaction of assessment tools with other systems is not mature enough since there is no communication specifications as stated in several surveys [LQ10a, QL11c].

Automatic assessment in computer programming domains can be applied in two distinct learning contexts: curricular and competitive learning.

Introductory programming courses are part of the curricula of many engineering and sciences programs. These courses rely on programming exercises, assignments and practical examinations to consolidate knowledge and evaluate students. The enrolment in these courses is usually very high, resulting in a great workload for the faculty and teaching assistants responsible for assessing student programs.

While the concept of "winners and losers" can hinder the motivation of students [VD03], competitive learning is a learning paradigm that relies on the competitiveness of students to increase their programming skills [Bur10, SKA08]. This is the common goal of several programming contests where students at different levels compete such as: the International Olympiad in Informatics (IOI)⁴, for secondary school students; the ACM International Collegiate Programming Contests (ICPC)⁵, for university students; and the IEEEExtreme⁶, for IEEE student members. In this context, several tools are used to allow students to train or participate in programming contests. These tools such as Programming Contests Management Systems (PCMS) and Online Judges (OJ) rely also on the assessment of programming exercises.

In both scenarios the manual assessment of programming assignments poses significant demands on the time of teachers [DLO05]. Apart from being time-consuming, manual

³IMS QTI Web site: <http://www.imsglobal.org/question/>

⁴IOI Web site: <http://ioinformatics.org>

⁵ICPC Web site: <http://icpc.baylor.edu/>

⁶IEEEExtreme Web site: http://www.ieee.org/membership_services/membership/students/competitions/xtreme

assessment hinders the consistency and accuracy of assessment results as well as it allows unintended biases and a diverse standard of marking schemes [RSZ10]. This demand stimulated the development of automated learning and assessment systems in many universities [AM05] as a means for grading the programming exercises of students as well as giving feedback on the quality of their solutions [TGPS08, SHP⁺06]. This feedback support is crucial for the computer programming learning [WW08, Mor07], especially for first year students that need to be adequately engaged in order to learn programming [Jen02]. Furthermore, immediate feedback motivates students to continue practising [Dal99, Tru07].

Beyond the automatic assessment other relevant topic in this domain is the availability of programming exercises. It is important that an e-learning system provides a collection of exercises covering a course syllabus and with different levels of difficulty. It has been shown that this can improve the performance of students and their satisfaction levels [WW08]. Students with lower computer skills can begin by solving easier problems in order to learn progressively and to stay motivated to solve the harder problems later [ILH00]. At the same time this gives them experience that is one of the factors that has a greater influence on the student success in learning programming [WLK04]. In recent years, a large number of programming exercises have been developed and published mostly for use in programming contests. These exercises are generally stored in proprietary systems (e.g. Online Judges) for their own use. Despite some efforts [QL12b] to define a common format to describe programming exercises, each of these systems has its own exercise format, making it difficult to share among instructors and students. This poses several issues on the interoperability of the assessment systems with other e-learning systems.

A number of learning tools and environments have been built to assist both teachers and students in introductory programming courses. Rongas, Kaarna, and Kalviainen [RKK04] established a classification for these tools dividing them into four categories: 1) integrated development interfaces, 2) visualization tools, 3) virtual learning environments, and 4) systems for submitting, managing, and testing exercises. To the best of the author's knowledge, no e-learning environment described in the literature integrates all these facets [VRV⁺11, GM07, EFMM10]. Several systems [Jen08, VRV⁺11, XC11, GG08] try to address this issue allowing the integration of automatic assessment tools with course management systems but these approaches rely on ad hoc solutions or proprietary plug-ins rather on widely accepted international specifications for content description and communication among systems.

1.2 Challenge

Assessing the work of students and providing individualised feedback to all students is time-consuming for teachers and frequently involves a time delay. The existent tools prove to be insufficient in complex evaluation domains where there is a greater need to practice [RKK04].

This dissertation addresses the following problem:

Student practice in domains with complex evaluation using e-learning is currently unsatisfactory.

The main goal of this research is to foster the practice-based learning in domains with complex evaluation. The essential tasks required by a practice-based learning environment are already available in a profusion of e-learning systems. There are already systems to manage instructions [LQ11a], to author and archive content [QL12a] and even to evaluate exercises [QL12c]. The challenge is to bind them together in a network that enhances their individual strengths and creates an environment for practising a complex skill. This insight led to the following research question:

*In domains with complex evaluation
is there an increase of effectiveness in practice-based learning
through the use of a network of best-of-breed e-learning systems?*

1.3 Approach

The novel approach of this research is to network the best-of-breed tools for a specific domain, rather than use the traditional approach of integrating components into a single system.

The cornerstone of this approach is an e-learning framework - called Ensemble - that acts as a conceptual tool in the definition and deployment of such kind of e-learning network. This framework relies on interoperability standards and specifications, thus several studies and surveys were conducted to select the most relevant.

Based on this framework a network of systems and services was created and deployed for a specific domain - the computer programming domain. This framework instance comprises several systems and services and their integration poses interoperability issues on two levels: content and communication. Content issues are tackled with a standard format to describe programming exercises as learning objects. Communication is achieved with the extension of existing specifications for the interoperation with several systems typically found in an e-learning environment such as learning management systems, learning objects repositories and assessment systems. Some of these systems were created from scratch and others were adapted to support the new content and communication specifications.

Having in mind this approach the research question stated in the previous section leads to the following hypotheses:

- H1. Exercises solving** - In a practical class students that use Ensemble start, complete and effectively solve more exercises and this advantage is maintained or improved over time;
- H2. Feedback** - In a practical class students that use Ensemble receive more feedback and this feedback is effective in overcoming their difficulties;
- H3. Attendance** - Practical classes have more attendance when students use Ensemble and this attendance is maintained or improved over time;
- H4. Grades** - Students that use Ensemble get better grades in the subject taught within the network and in the programming course.

In chapter 10 this set of observable assertions is tested to validate the dissertation objectives.

1.4 Contributions

This dissertation includes several contributions that can be divided in two facets: conceptual and concrete. Conceptual contributions are related with abstract concepts such as ideas, algorithms, studies, surveys and theoretical frameworks. The conceptual contributions are:

Studies and surveys - e-learning frameworks [LQ10a]; LMS interoperability [LQ11a]; Programming Exercises Assessment Systems [QL12c] and e-learning standardization [QL11c, LQ10b, QL09].

Theoretical frameworks - a conceptual framework for the development of networks in complex evaluation domains [QL11a, LQ11c];

Concrete contributions are related with the implementation of tools and specifications. The concrete contributions are:

Frameworks - An Ensemble framework instance applied to computer programming⁷ [LQ11e, LQ11b, LQ10f, LQF10];

Specifications, systems and services : PEXIL⁸ - a standard format for the description of programming exercises [QL11e, QL11d, QL11b, LQ09e, LQ09c]; CrimsonHex⁹ - a programming exercises repository [LQ10c, LQ10e, LQ09a, LQ09d, LQ09b, LQ10e];

⁷<http://ensemble.dcc.fc.up.pt>

⁸<http://ensemble.dcc.fc.up.pt/Pexil>

⁹<http://ensemble.dcc.fc.up.pt/CrimsonHex>

Moodle plug-in for crimsonHex¹⁰ [LQ10d]; BabeLO¹¹ - a programming exercises format conversion service; Evaluate¹² - a programming exercises evaluation service [LQ11d, LQ10h, LQ10g]; Petcha¹³ - a teaching assistant system [QL12b].

1.5 Thesis structure

The organization of this dissertation is depicted in Figure 1.2.

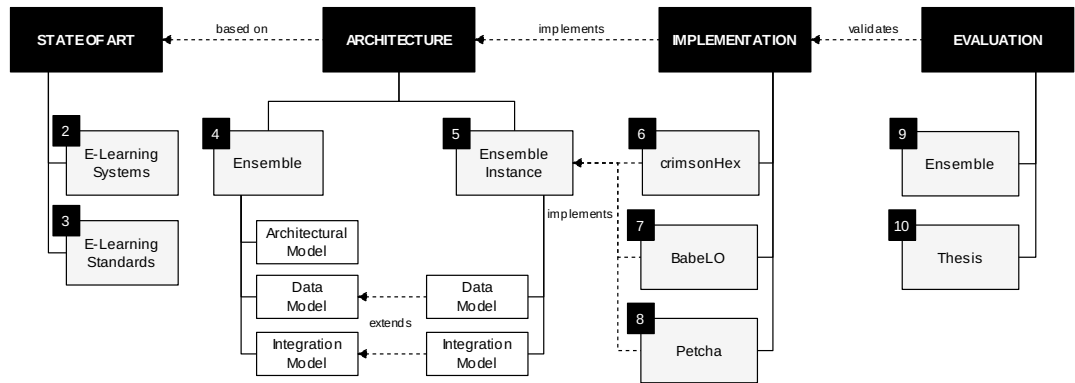


Figure 1.2: Thesis structure.

This dissertation is composed by nine chapters (gray rectangles) plus the introduction and conclusions. These chapters are organized in logical parts (black rectangles). Each part is described as follows.

Part 1 (State of Art) - gathers information on e-learning systems and standards. It is composed by two chapters: e-learning systems and e-learning standards. The former starts by tracing the evolution of e-learning systems from monolithic systems to specialized services and stresses the interoperability features of three systems typically founded in introductory programming learning environments [RKK04]: Learning Management Systems, Repositories and Assessment Systems. The latter gathers information on e-learning standards in order to choose the most suitable for the domain of automatic evaluation of exercises. The organization of this chapter is based on several studies found in the literature [JBK05, VA06]. In these studies the outcomes of the standardization efforts on e-learning interoperability are divided in two levels (content and communication). The chapter adds a new level (Frameworks) due to its

¹⁰<http://ensemble.dcc.fc.up.pt/ChMoodlePlugin>

¹¹<http://ensemble.dcc.fc.up.pt/BabeLO>

¹²<http://ensemble.dcc.fc.up.pt/Evaluate>

¹³<http://ensemble.dcc.fc.up.pt/Petcha>

relevance in the last years on the development of new e-learning systems based on Service-Oriented Architecture (SOA).

Part 2 (Architecture) - comprises two chapters. The first one presents the design of the Ensemble e-learning framework as a conceptual tool to organize a network of e-learning systems and services that use complex evaluation. The chapter details the three framework models (architectural, data and integration). The second chapter presents the use of the Ensemble framework in a specific domain: the computer programming domain. The presentation comprises the following topics: 1) the architecture of an Ensemble instance for the computer programming domain; 2) the data model based on an interoperability language for programming exercises; 3) the integration model of the systems and services involved through the creation and extension of existent communication specifications; 4) selection of tools that comprises a deployable Ensemble instance.

Part 3 (Implementation) - details the implementation from the scratch of three systems and services. The crimsonHex repository is a specialized and extensible repository of programming exercises described as learning objects. The BabeLO service handles the conversion between different programming exercises formats. The Petcha system acts as an automated teaching assistant in computer programming courses helping both teachers to author programming exercises and students to solve them. It also coordinates a network of heterogeneous systems, integrating assessment systems, learning management systems, learning object repositories and integrated programming environments.

Part 4 (Evaluation) - includes two chapters. Firstly, an Ensemble acceptability evaluation is performed focusing on the usefulness, reliability and interoperability of a network of systems and services for the computer programming domain. Then, the thesis is validated by proving the Ensemble effectiveness to achieve the thesis goals. These evaluations rely on a classroom experiment made in a Polytechnic Superior school in Porto. Although both chapters share the experiment conditions, the characterization of the experiment is made on the first chapter. It includes the design of the controlled experiment, the instruments used in this study and how the data was collected.

The **last chapter** resumes this dissertation pointing out the major contributions and future work perspectives. Finally, four **appendices** are included at the end of the document. The first two appendices include a list of heuristics and the usability survey (based on these heuristics) answered by the students at the final of the experiment, respectively. The third appendix is the user manual of Petcha for both teachers and students. The last appendix is dedicated to the crimsonHex Application Programming Interface (API).

Part I

State of Art

Chapter 2

E-Learning systems

*"There are two fundamental equalizers in life -
the Internet and education."*

John Chambers, CEO, Cisco Systems Inc. 1999

Nowadays, the learning experience is no longer confined within the four walls of a classroom. Computers and the Internet have broadened this horizon by creating a way of delivering education known as e-learning. E-learning or Electronic Learning can be defined as the delivery of educational content via any electronic media, including the Internet, satellite broadcast, audio/video tape, interactive TV, CD-Rom and others [TS05]. However there are other activities such as practising, evaluating, assessing and giving feedback, that are undertaken by teachers and students and goes beyond content delivery even though they may operate on content that has been delivered. In this context e-learning systems play a relevant role in the teaching-learning process facilitating the dissemination of knowledge by teachers and its absorption by students.

This chapter starts by tracing the evolution of e-learning systems from monolithic systems to specialized services. These services can be easily recombined in different learning processes. This chapter focuses on learning processes within domains with complex evaluation. In this domain there are several candidates to offer services such as those referred by Rongas, Kaarna, and Kalviainen [RKK04], namely: Learning Management Systems, Learning Objects Repositories and Assessment Systems. These three types of system are detailed in the remainder sections.

2.1 E-Learning evolution

The evolution of e-learning in the last decades has been astonishing. In fact, e-learning seems to be constantly reinventing itself, finding new uses for technology, creating new tools, discovering new concepts. Platforms for supporting e-learning have been evolving for some years, exploring many approaches and producing a great variety of solutions. In spite of their number, these platforms can be grouped according to their characteristics and their ability to interact with each other. Thus, a good way to understand them is by studying both their architectural features and the standards they support.

2.1.1 Early systems

The genesis of e-learning, despite some efforts to foster remote education [Har06], coincides with the development of network communication in the late 1960s, more precisely, with the invention of e-mail and computer conferencing (1971). These innovations contribute to the collaboration between teachers and students and initiate a new education paradigm shift [Wil05]. During the 1980s and 1990s, there was a significant growth in the number of part-time students and also in non-traditional learners, such as women's returning to the workforce after child rearing [Wil05]. The growth in lifelong learning market made the educational institutions eager to accommodate the needs of these non-traditional students and to offer them e-learning courses based on the Internet.

2.1.2 Component based systems

In their first generation e-learning systems (Figure 2.1) were developed for a specific learning domain and had a monolithic architecture [DOL⁺07].

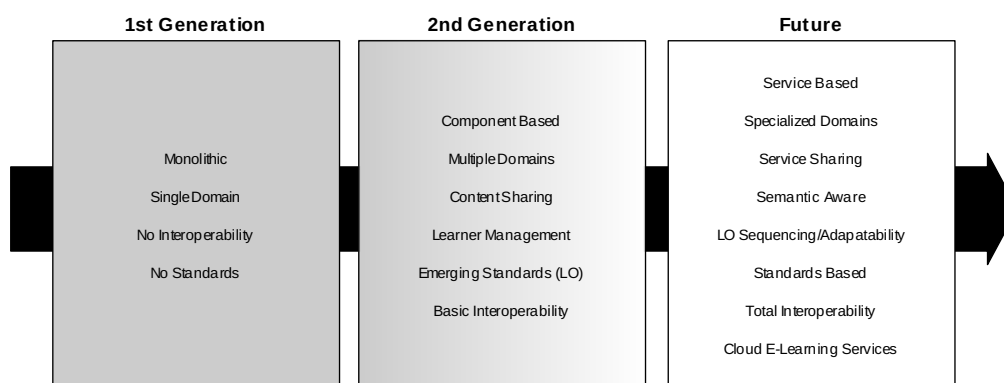


Figure 2.1: The evolution of e-learning systems.

Gradually, these systems evolved and became domain-independent, featuring reusable tools that can be used virtually in any e-learning course. The systems that reach this level of maturity usually follow a component oriented architecture in order to facilitate tool integration. Different kinds of component based e-learning systems target specific aspects of e-learning such as student or course management. This architectural model structures software around pluggable and interchangeable components, thus enabling the development of larger systems, resulting from the collaboration of different teams. In some cases component oriented architectures led to oversized systems that are difficult to reconvert to changing roles and new demands. This is particularly true in e-learning. A criticism to this approach is that it reduced e-learning to the use of one-size-fits-all systems, i.e., systems that 1) can be used on any learning subject but fails to address specific needs of each of them, and 2) can be used by any student but is not able to adapt to unique characteristics of individuals [DOL⁺07].

2.1.3 E-Learning services

Component based integrated environments became the corner stone of e-learning. Although they tend to incorporate a growing number of tools, they cannot afford to be isolated from other software systems operating in academic institutions. There are several strategies routinely used to achieve this integration. The most common are depicted in Figure 2.2. Integration usually includes at least one Web application, and this is typically designed based on the well known three-tier architectural pattern [Eck95]. There is a potential for integration in any the three classical tiers: presentation, logic and data.

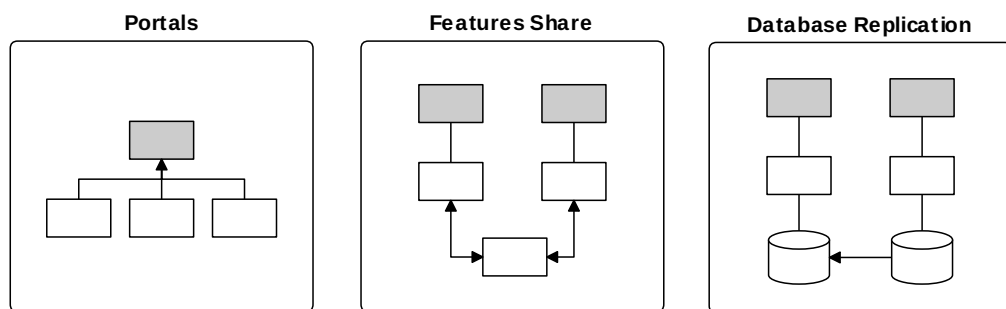


Figure 2.2: Integration classic models.

The **portal** strategy aggregates content from multiple sources with a common presentation layer. It integrates at the presentation tier, providing an unified web interface to a number of independent subsystems, including e-learning systems. The major advantage of this strategy is the fact that it gives users a sense of unity, sometimes at the cost of compromising consistency.

In the **feature sharing** approach the presentation is independent while sharing some features. The integration is at the logic tier and is becoming increasingly popular as more systems expose their functionality using web services. User authentication based in directory services, such as Lightweight Directory Access Protocol (LDAP), is an apt example of this type of integration. It should be noted that the use of web services for feature sharing is not a remedy for this problem. Although service oriented architectures typical (but not necessarily) use web services, using web services does not automatically qualify a system as SOA.

Finally, integration may occur at the data tier where different applications share the same content. In this context the partial **database replication** is arguably the most common example. For instance a LMS may import data on students, courses and student enrolments in courses from administrative systems to avoid the burden of entering this data manually. These integration models are usually combined. For instance, a portal that provides an unified presentation may also adhere to a single sign-on mechanism shared with other services.

These approaches to system integration are a collection of pragmatic solutions that raises their own problems. In fact, integration on specific points creates the same kind of entanglement found in monolithic systems. Since integration is not driven by architecture, there is no coherence among a disparate set of connections that ties up the system and compromises future changes. This problem is not specific of e-learning systems and is generally approached using SOA.

SOA [Erl05] is already a mature architectural pattern with established principles and technologies and can be defined as a systematic approach to system development and integration. Instead of point-to-point integration, SOA proposes the development of systems around the concept of interoperable services. These services must be loosely coupled, allowing them to be easily recombined in different processes (typically business processes).

Figure 2.3 shows the SOA components and their relations, namely: contract - collection of all the messages supported by the service; service - implementation of the functionality promised by the contracts it exposes; message - unit of communication; service consumer - software that interacts with a service through the exchange of messages; and end point - an Uniform Resource Identifier (URI) that specifies where the service can be found and consumed.

The communication between these components is generally based on Web services. The Web Service Description Language (WSDL) provides a description of how to use a service. The definition of how several Web services cooperate to achieve a given goal cannot be handled by the WSDL specification and, in this case, the use of coordination models (e.g. orchestration, choreography) is needed to define an interoperable integration model. This model facilitates

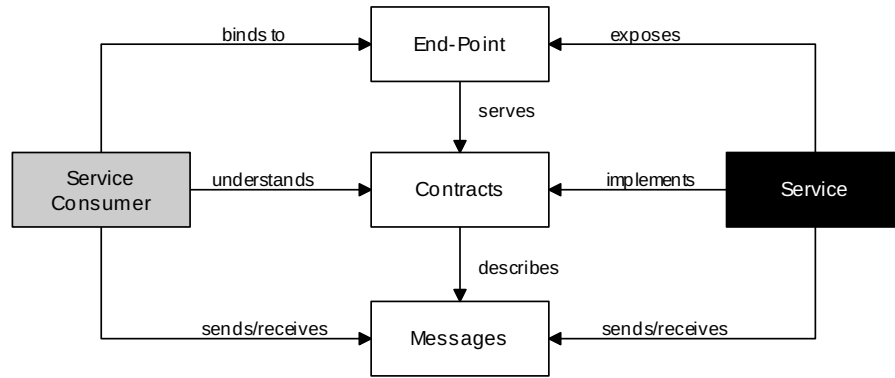


Figure 2.3: SOA components.

the expansion of automated process integration and the management of the workflow within services.

This architectural pattern is appropriate in contexts where components (services) participate in several processes, and process configuration needs to be flexible. This concept of process is applicable to e-learning: flexible learning processes can be used to congregate the best e-learning services available for each particular domain and create an instructional environment more adapted to the student needs and requirements. As this type of architectures became mainstream, a few initiatives to adapt SOA to e-learning emerged [LQ10a]. These new frameworks and APIs contributed with the identification of service usage models and are generally grouped into logical clusters according to their functionality [ASQ⁺06].

Service oriented architectures cannot be seen as a silver bullet for all e-learning system integration problems. The adoption of SOA creates new challenges such as 1) the integration of heterogeneous services based on semantic information, 2) the automatic adaptation of services to users (both learners and teachers), and 3) the lack of a critical mass of services to supply the demand of e-learning projects.

1) Web services do standardize communication but they do not ensure that every service assigns the same meaning to what is being communicated. This is a general problem with heterogeneous services and affects particularly e-learning since what is being communicated involves complex data (e.g. learning objects) and complex functions (e.g. automatic assessment).

2) Other criticism to integrated e-learning systems is the lack of focus on the student and on the learning domain. Tools tend to be too common to all learning domains and the same contents are presented to all students enrolled in a same course. On its own, the adoption of SOA will not address this problem and there is a risk that it may actually increase it, if it is not driven by pedagogical concerns. For instance, if the same course content is offered to

a wider range of students with different backgrounds then it will be actually be less focused on students. However, the use of services creates a possibility for a systematic approach to use adaptability in e-learning. On one hand, a service contract is a good place to ensure that the data on which to base adaptability can be effectively gathered. On the other hand, adaptability in itself may be provided as a service that can be configured in a learning process.

3) Last but not least, the number of e-learning services is still very small. Before reaching a critical mass of e-learning services, available to participate in reconfigurable pedagogical processes, it will be difficult to support the claims of SOA in the e-learning context. Surely, many infrastructural services are common to any SOA service, but there are few truly pedagogical services available.

2.1.4 Specialised E-Learning Services

The pressure to adopt SOA in e-learning is mostly fuelled by managerial needs of academic institutions, rather than pedagogical concerns of teachers. In some cases is an internal need, of combining infrastructures of autonomous departments with different responsibilities within an academic institution. In other cases results from external pressure, of linking with other institutions in order to offer join e-learning programs. In these cases the resulting platform typically relies on an LMS, thus having the same problems of component based systems, especially from an educational viewpoint.

Traditionally, features are added to LMS by integration of new components. These components are specific to an LMS and tend to be very general, in order to be reusable in as many courses as possible. By contrast, a service may be reused on different systems, thus making more sense to specialize it to a specific purpose. Moreover, a service can make use of certain hardware or software; for instance a specific program available only on a particular platform. Last but not least, specialized e-learning services are able to participate in multiple and easily reconfigurable learning processes. A learning process is a collection of related and structured activities. In this context, each activity is implemented by a specialized e-learning service. Services may participate in several learning processes and new processes can be created or reconfigured.

This dissertation emphasis the learning processes in domains with complex evaluation. These learning processes involve several learning systems such as those referred by Rongas, Kaarna, and Kalviainen [RKK04], namely:

Learning Management System - to manage and retrieve the exercises to the learners;

Learning Objects Repository - to persist exercises and related meta-information;

Assessment System - to evaluate and produce feedback on learners' attempts.

These types of services are very different in nature. The LMS is not in strict sense a specialized service. It is a system designed to be a complete and generic e-learning solution rather than a service. Nevertheless, since a typical LMS is a component based system, it may be extended to incorporate the features it lacks to communicate with other services, and provide a front-end both for learners and teachers. The LMS possess a user interface and learners can interact directly with it. The other two, the repository and the assessment systems, provide truly specialized services.

In the following sections each of these systems are detailed with emphasis on their interoperability support.

2.2 Learning Management Systems

Nowadays, an LMS plays a central role in any e-learning architecture and can be defined as a software application for the administration, documentation, tracking, and reporting of training programs, classroom and on-line events [Ell09].

2.2.1 Categories

Several designations and respective acronyms are used to typify LMSs (Figure 2.4). The following list includes the most common: Content Management Systems (CMS) , Learning Management Systems (LMS), Learning Content Management Systems (LCMS) , Managed Learning Environment (MLE) and Virtual Learning Environment (VLE). These five types of e-learning systems have a considerable overlap and they are difficult to differentiate. For the sake of simplicity only the first three will be considered as representative of the main categories of e-learning systems.

While these three categories still share common characteristics, they also have some distinctive features that justify distinguishes among them. The CMS was introduced in the mid-1990s mostly in the on-line publishing industry. This type of system can be defined as a data repository that also includes tools for authoring, aggregating and sequencing content. The main goal of these tools is to "simplify the creation and administration of on-line content" [Nic01]. CMSs are focused on content with the main purpose to store information and provide access to it. CMS content is organized in small self-contained pieces of information to improve reusability at the content component level. These content components when used in the learning domain are called Learning Objects (LOs).

The LMS goal is to simplify the administration of learning/training programs within an or-

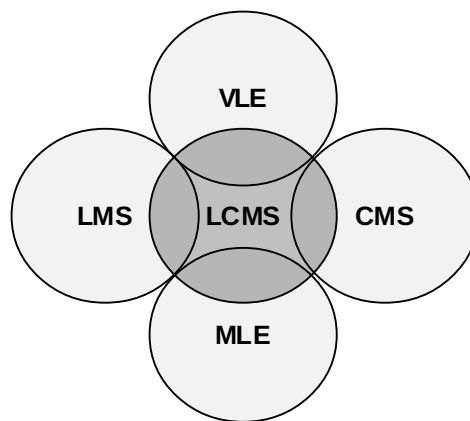


Figure 2.4: E-Learning system types.

ganization [HK06]. Two usage scenarios are relevant: learners can use the LMS to plan their learning experience and to collaborate with their colleagues; teachers can deliver educational content and track, analyse and report the learner evolution within an organization. Most LMSs are structured around courses rather than content, thus they are only reusable at the course level. LMS also do not support the creation of instructional content and teachers must resort to third party content creation tools. A LCMS combines the administrative and management features of a LMS with the content creation of a CMS. In a LCMS, you would have libraries of LOs that can be used either independently, or as a part of larger instruction sets. For instance, one LO can be used in several courses with several learners.

Table 2.1 relates the three categories of applications based on several main features [HK06, Don02]. Each feature may have a robust (R) or limited (L) support in these categories or, simply, no support.

Table 2.1: Features of CMS, LMS and LCMS.

Features	CMS	LMS	LCMS
Manage learners	-	R	L
Manage content	R	L	R
Create content	R	-	R
Launch and track e-learning	-	R	L
Assessment and feedback	-	R	R

Nevertheless, the trend in e-learning systems is integration, therefore most of them evolved to the same set of standard features and many of these acronyms are recurrently used as synonyms. In the course of this section the LMS is taken as the representative of e-learning systems since the term LMS is often used to refer to both LMS and LCMS, although the LCMS is a further development of the LMS.

As every kind of software, LMSs continue to evolve to meet market demands, namely:

SOA - in these architectures LMSs expose their functions as services and consume services from their operational environments, improving their interoperability with other e-learning systems [DOL⁺07];

Web 2.0 - with the recent appearance of Web 2.0 tools and the popularity of social networking tools like Facebook and Twitter, there has been a great demand to use similar tools in the LMS to enhance the communication among teachers and students;

Talent Management - Talent Management software systems are an extension of traditional human resource management systems. Some researches [Ber09] shows that in 2009 more than 70% of large companies have an LMS already and almost 1/3 of these companies are considering replacing or upgrading these systems with integrated talent management systems [LL10];

Mobile Learning - with more students working at distance, there has been also a strong demand to make e-learning applications accessible through mobile devices (e.g. Smartphones, Tablets) know as Mobile Learning or m-learning;

”Software as a Service” - with Software as a Service (SaaS) schools can relieve the financial burden of maintaining their LMSs by outsourcing the hosting service;

Open Source Software - commercial LMS (e.g. Blackboard, WebCT) have dominated the education market in previous years, but as costs increase, schools and companies are now looking for other options such as open-source solutions (e.g. Moodle, Sakai) that are financially more attractive.

2.2.2 LMS Interoperability

Interoperability is the ability of different computer systems, applications or services to communicate, share and exchange data, information and knowledge in a precise, effective and consistent way. In the e-learning field this topic is extremely important since there is the need for all systems that typically compose an e-learning environment to communicate and share data consistently. In this context, several organizations have been developing specifications and standards in the last years focusing on the content and communication interoperability [Reh03, DOL⁺07].

In the following sub-subsections an LMS interoperability comparative study is conducted. Several studies have been conducted to analyse and evaluate LMSs from pedagogical and institutional perspectives [Bri98]. However, the author is not aware of any study to evaluate the interoperability of LMSs with other systems typically found in an educational institution.

This study is part of an effort to select an LMS on which to base the development of e-learning systems integrating heterogeneous components. Given the number of LMS vendors it would be impracticable to study them all. Therefore, two LMS vendors are chosen and their interoperability features are analysed.

2.2.2.1 LMS selection

A good number of LMSs that were developed in the past fifteen years are still in use and under active development. For the purpose of this study the focus is on a few systems that are representative of the LMS universe in terms of their characteristics and market share. A simple categorization of this type of systems is according to their development models. There are fundamentally two: open source systems, such as Moodle, Sakai, .LRN or Dokeos; and commercial systems such as WebCT/Blackboard or Desire2Learn. Figure 2.5 presents a timeline of several initiatives grouped by their development model. In these two categories the most popular systems were selected taking as reference the available data on global LMS usage [DW09].

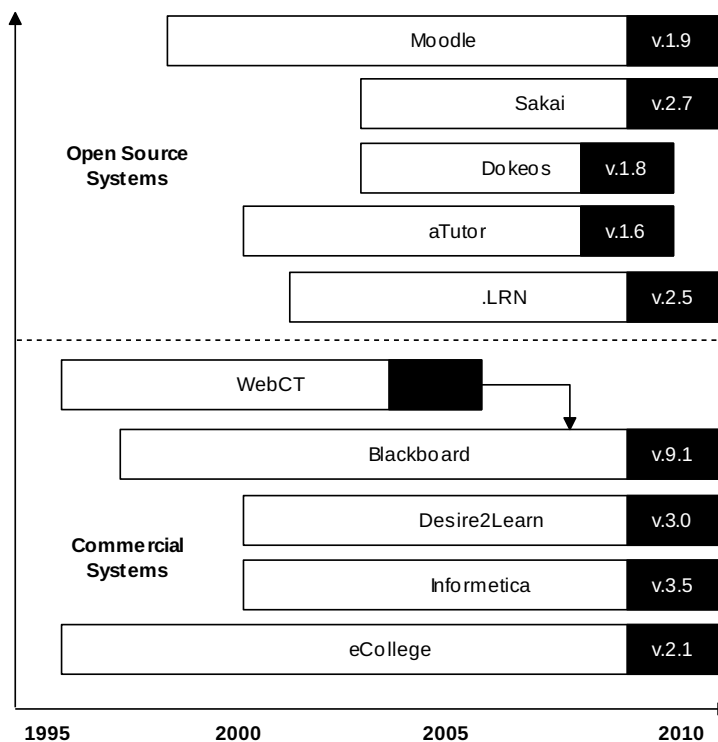


Figure 2.5: Timeline of development of major LMSs.

The Figure 2.5 shows that the first major LMSs adopted a commercial development model but since the beginning of this century there has been a shift towards open source systems. In fact, this shift was already recognized as a trend in LMS development [DW09]. In spite of the

growing popularity of open source, commercial systems are still relevant and they must be included in any representative sample of LMSs. As part of this study a survey was conducted in 2011 on e-learning systems usage in Portuguese higher education institutions. About 20 different institutions responded and the results for LMS usage are shown in Figure 2.6.

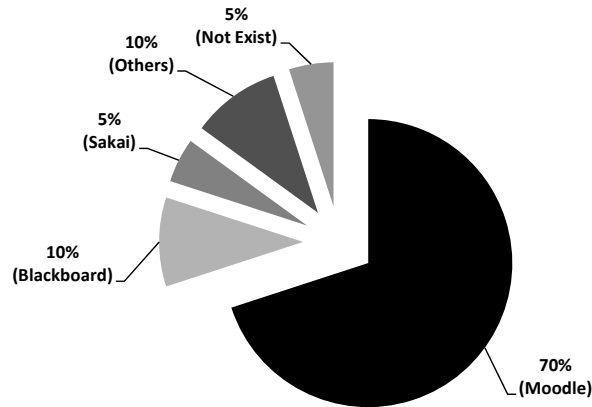


Figure 2.6: LMS usage in Portuguese higher education institutions.

The selection of the LMSs for this interoperability study was based on the two most popular LMS - Moodle and Blackboard. They represent the two main development models used by LMS vendors (open source and commercial); and combined they have a significant share on the LMS market (33.2% on the international market [DW09] in 2009 and 80% on this recent survey in Portuguese institutions). The following paragraphs provide an overview of the selected systems.

Moodle (version 1.9.9 - 8th June 2010) is a free and open-source LMS written in PHP and created by Martin Dougiamas. Its name is an acronym for Modular Object-Oriented Dynamic Learning Environment. In early January of 2010, Moodle had a user-base of 46,624 registered sites with 32,464,992 users in 3,161,291 courses in 209 countries and in more than 75 languages [CF07].

Blackboard (version 9.1 - 1th April 2010) was developed by Blackboard Inc. in 1997 and is an on-line proprietary LMS that is used by over 3700 educational institutions in more than 60 countries. In February 2006, the virtual learning environment called WebCT (Course Tools) was acquired by Blackboard Inc. (Blackboard, 2005) and, as part of the acquisition terms, the Blackboard brand was assumed until now.

2.2.2.2 Interoperability Facets

The interoperability features of a system reflect the operational environment where it is expected to be deployed. The operational environment of an LMS includes different systems

and services with which it may have to communicate and exchange data. As depicted in Figure 2.7 two broad classes of systems were identified that usually integrates in the operational environment of an LMS, each corresponding to a different facet in LMS interoperability. A layer of infrastructural systems and services was also identified that is domain independent but that plays an important role in LMS interoperability. For the purpose of this study, the

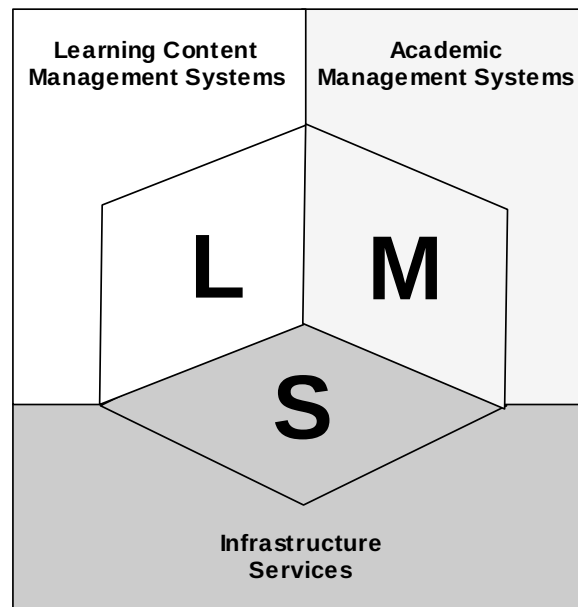


Figure 2.7: LMS interoperability facets.

identified broad classes of systems are the following:

LCMS are used for the development, management and publishing of digital learning content (e.g. Learning Objects) that the LMS delivers. Examples of these systems are the Learning Object Repositories, e-Portfolio Systems, Authoring Tools, Specialized Evaluators and others.

AMS are used for managing academic data information of an educational institution. Typical features of these systems are the management of courses, classes and students, the enrolment of students in courses, the submission of summaries and grades by teachers, payment of fees, among others.

Apart from these facets the LMS is supported by infrastructure services providing basic functions that are not specific to e-learning, such as directory services for authentication and authorization or printing services. This infrastructure also includes the web or application server, the database engine and the operating system. In many cases this infrastructural layer is used for implementing ad hoc interoperability solutions.

In the following paragraphs the selected systems - Moodle and Blackboard - are analysed and compared regarding these two facets. In each facet the remote systems, the existent standards and the interoperability issues are identified and categorized.

The **Learning Content Management facet** focuses on the interoperation with systems that provide pedagogical content and services delivered by the LMS. The content delivered by an LMS can be created, obtained, gathered or evaluated in several types of systems such as Learning Objects Repositories, E-Portfolio systems, Authoring Tools, Specialized Evaluators or Quizzes.

The integration with e-learning content management systems can be implemented on the LMS data or business layer. In the former the integration uses the import/export features of both system and relies on the support of common formats. In the latter the integration relies on the existence of compatible web services in both systems.

Data integration is the simplest and most popular form of integration in content management and assumes an important role in the LMS interoperation with system types that do not require a tight integration, as is the case with authoring tools (e.g. Reload, Hot Patatoes). For instance, the Reload¹ authoring tool can be used to create learning objects in Sharable Content Object Reference Model (SCORM) format and Blackboard supports and imports SCORM packages. Another example is the the Hot Potatoes² system that enables the creation of quizzes - interactive multiple-choice, short-answer, jumbled-sentence, crossword, matching/ordering and gap-fill exercises - in HTML format. Moodle includes an activity that imports the quiz (HTML file) previously generated in the Hot Potatoes system.

Table 2.2 lists some of the most important e-learning content standards and specifications defined in the last years by educational organizations. For each standard the LMS support status is presented.

Table 2.2: LMS support of content standards.

Specifications	Moodle 1.9	Blackboard 9.1
IMS CP	yes	yes
SCORM	yes	yes
IMS CC	partial	partial
IMS QTI	yes	yes
IMS LD	no	no
IMS SS	no	no

¹Official Web site: <http://www.reload.ac.uk/>

²Official Web site: <http://hotpot.uvic.ca/>

The studied LMS support almost all the LO package standards with exception of the recent IMS CC that is only partially supported³. In relation to the design and sequencing of learning activities standards are not yet supported by these LMS, probably due to their complexity.

It is possible also to integrate an e-learning tool with an LMS on the business layer. For instance, the IMS Learning Tools Interoperability (LTI)⁴ provides a uniform standards-based extension point in LMSs allowing remote tools to be integrated. Although this specification is still not explored by the major LMS vendors, obtaining the certified support for IMS LTI is already a major milestone in their development plan. Another integration approach is through the use of a common API. The LMSs provide APIs to allow developers to extend their predefined features through the creation of plug-ins. Table 2.3 enumerates the approaches used by the selected LMSs to address the interoperability issues regarding the integration with other system types.

Table 2.3: Integration API in reference LMS.

System types	Moodle 1.9	Blackboard 9.1
Repositories	Repository API	Building Blocks API
E-Portfolios	Portfolio API	Building Blocks API
Assessment systems	OPAQUE ws	no

Moodle version 2.0 includes several APIs to enable the development of plug-ins by third parties to access repositories and portfolios. Blackboard uses the Building Blocks technology to cover the integration issues with other systems. A Building Block is simply a web application that runs on the Blackboard application server. This technology allows third parties to develop modules using the Building Blocks API. For instance, the company Verben Consulting LLC created a building block that provides a search user interface that allows searching in the Merlot⁵ referatory and returns matching results along with the metadata for each LO.

The **Academic Management facet** aggregates all the information regarding administrative, financial, technical or scientific processes usual in educational institutions. Unlike in content management, there is a sole type of system in this facet - the Academic Management Systems (AMS). The AMS typically manages processes such as the enrolment of students in courses, the management of grades or the payment of fees. This interoperability facet is not as mature as the LCMS facet and there are still few standards available. This fact burdens the integration of academic management systems with LMSs that must resort to ad hoc solutions based on the infrastructural layer. Figure 2.8 synthesizes the 20 responses

³Moodle 2.2 supports IMS CC package import and version 2.3 (release date: 18th June 2012) will export CC packages.

⁴Official Web site: <http://www.imsglobal.org/lti/>

⁵Official Web site: <http://www.merlot.org>

received from different institutions regarding the AMS usage. This data shows that no

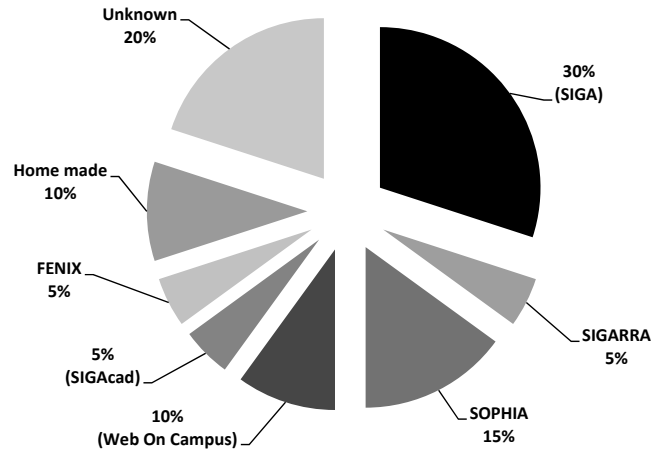


Figure 2.8: AMS usage.

system is clearly preferred by Portuguese educational institutions. The choices are divided by the systems SIGA, SIGARRA, SOPHIA and Web on Campus. It should be noted that most of these evolved from home grown systems and are in use in different schools from the same university or polytechnic institute. In some cases spin-offs were created to develop and commercialize these systems but the size of these companies cannot be compared with those developing other types of systems related to e-learning, such as LMSs.

There is an obvious gain in integrating AMSs and LMSs: avoiding the duplication of processes. For instance, course management is required in both systems and with a tight integration it can be performed in just one of them. Other processes can be performed in only one side and reflected in the other such as: enrolment of students, exams schedule, summaries, absences and grades management. Currently, educational institutions use ad hoc solutions to implement this type of integration. For instance a LMS may import data on students, courses and student enrolment in courses from administrative systems to avoid the burden of entering this data manually. These integration models are usually combined. For instance, a portal that provides and unified presentation may also adhere to a single sign-on mechanism shared with other services.

2.2.3 Conclusions

The main conclusion of this study is that there is still a long road ahead in LMS interoperability. In general it is not straightforward to connect an LMS to another system. A lot of work has already been done in defining standards but many of them are supported neither

by the LMS nor by the system that surround them.

LMS interoperability was analysed based on two facets: Learning Content Management and Academic Management. There is a huge asymmetry among these facets: the first has a larger number of systems and mature standards; the systems in the second facet are mostly home-grown with few and immature standards to regulate both content (e.g. academic records, course forms, grades, summaries) and communication.

The content management facet is much more developed then the academic management facet. Content formats, especially those of learning objects, are already mature and widely supported by the analysed systems. The notable exception is the recent Content Cartridge of IMS that is not yet supported, as is not its companion specification - the Learning Tools Interoperability - that is still being implemented in Moodle and Blackboard. This specification promises to be a major step towards content interoperability among e-learning systems. Meanwhile, to integrate LMS with content management systems one must resort to system specific APIs.

On the academic management facet there are no AMS standing out from the crowd and most of those in use, at least in Portuguese higher education institutions, are home grown systems. Standards in this facet are few and immature and not widely supported by existing AMS. As a consequence, the integration of LMS and AMS relies on infrastructure services. A set of integration strategies were presented that are commonly used for implementing these ad hoc integrations.

2.3 Repository Systems

A repository can be thought of as a storage area from which users can publish/retrieve resources. Most of these resources are described by metadata for discoverable purposes. The need for repositories has been growing in the last decade, since more educators are eager to create and use digital content and more of it is available. This growth led many to neglect interoperability issues that are fundamental to share educational resources and to (re)use them on different contexts. In this section an interoperability study is conducted on the existing repository software distinguishing two types of repositories - digital libraries and learning objects repositories - two concepts that overlap to a certain extent. It also focuses on the distinction between the actual repositories and the software used to implement them, highlighting the differences in software features for both categories.

2.3.1 Categories

There are several typologies to classify the repositories available in the literature. McGreal [McG08] categorises repositories in three types based on resource location: 1) those that house content primarily on site (e.g. MIT Open Courseware); 2) those that provide metadata with links to resources housed at other sites (e.g. Merlot) – also called referatories [Rog03]; 3) those that provide both content and links to external content (e.g. Ariadne). Ochoa [OD09] conducted a detailed quantitative study of the process of publication of learning objects in repositories and grouped repositories in five types: Learning Object Repositories, Learning Object Referatories, Open Courseware Initiatives, Learning Management Systems and Institutional Repositories. Other relevant studies [Ter08, RSS10, Fay10] categorize repositories mainly by general characteristics as language used, subject area, end users, fee type, quality control, etc.

In this dissertation, repositories are divided in two groups: digital libraries (DLs) and learning objects repositories (LORs). The evolution of the software and specifications for repositories can be seen in Figure 2.9. In the next subsection both are detailed.

Digital libraries store documents in digital formats and metadata on those documents. Learning objects repositories store learning objects, which are objects containing educational content and metadata on those contents. At first sight a LOR seems to be basically a DL storing educational content but there are also differences related with metadata and packaging. The metadata schemata used by DLs is generic (e.g. Dublin Core - DC, Metadata Encoding & Transmission Standard - METS) while those used by LORs are specific to e-learning (e.g. LOM). Also, LORs package content and metadata in a single unit using e-learning standards (e.g. SCORM) while DLs usually keep content and metadata separated.

In spite of these differences and similarities there are several references in the literature on attempts to implement LORs using DLs software [Les06, RWS06]. A reasonable explanation for this is the lack of software specifically for implementing LORs. In fact, unlike what happens with DLs, most implementations of LORs use home-grown software.

2.3.2 Repository Interoperability

This subsection presents an interoperability study on the existing repositories. Two types of repositories software are analysed - digital libraries and learning objects repositories. This analysis distinguish between the actual repositories and the software used to implement them, highlighting the content and communication specifications supported by the both.

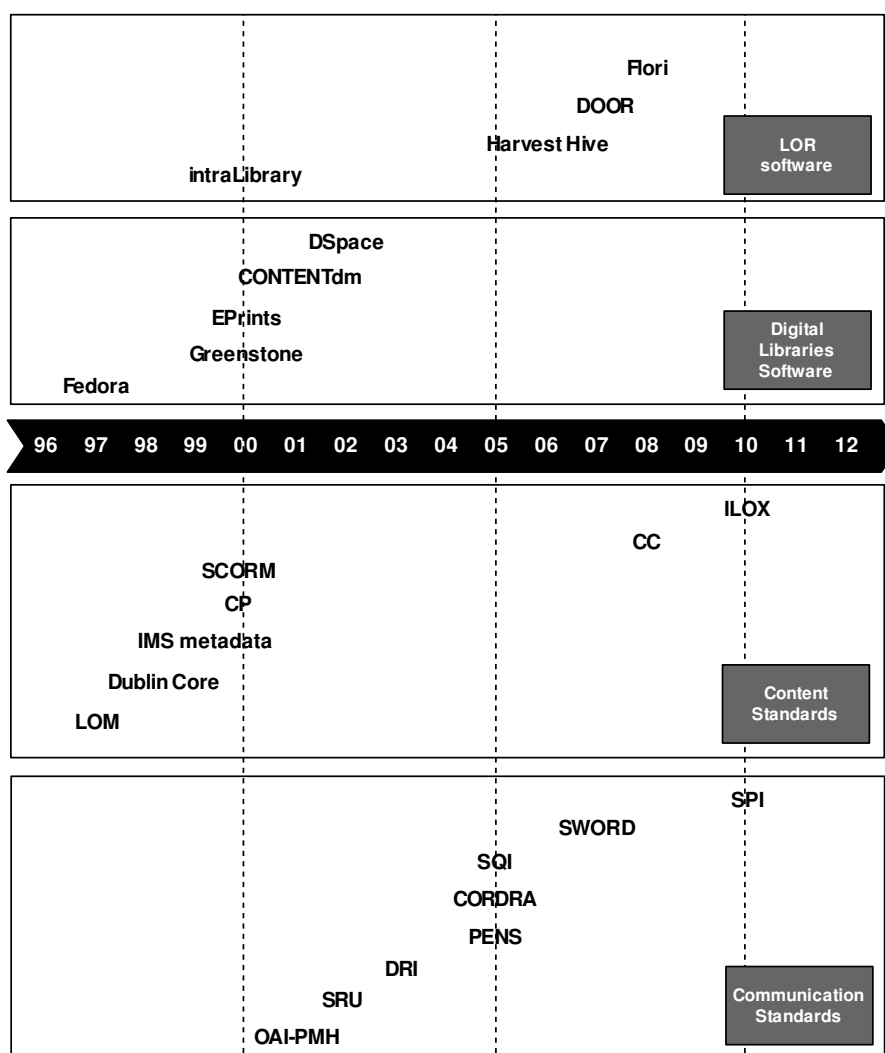


Figure 2.9: Evolution of software and specifications for repositories.

2.3.2.1 Software for Digital Libraries

A non-exhaustive list of software to create DLs is presented in Table 2.4 [RSS10].

According to the Directory of Open Access Repository (OpenDOAR) - a directory of academic open access repositories - and the Registry of Open Access Repositories (ROAR) data, at December 2011, the majority of repositories (in a sample of 2164 repositories) are built using the DSpace software, as shown in Figure 2.10.

Software for Digital Libraries (DL) was designed for repositories of digital content in general not specifically for LOs. Thus, this type of software in general lacks some of the features required by Learning Objects Repository (LOR) such as the support for: 1) e-learning

Table 2.4: Digital library software.

Repository	Commercial	Metadata	Communication
CONTENTdm	Commercial	DC, METS	OAI, Z39.50
DigiTool	Commercial	DC, MODS,METS	OAI, Z39.50
DSpace	Free	DC, MODS,METS	OAI, SWORD, OpenSearch, SR*
EPrints	Free	DC, MODS,METS	OAI, SWORD
Equella	Commercial	LOM	OAI, SWORD
Fedora	Free	LOM	OAI, SWORD
Greenstone	Free	METS	OAI, Z39.50
Zentity	Free	DC, METS	OAI, SWORD, RDFS

metadata (e.g. LOM) and content packaging (e.g. SCORM); 2) federated searching; 3) classification using folksonomies; 4) versioning, reviewing and evaluation; 5) interoperability with e-learning systems. However, the newest versions of some of the systems listed in Table 2.4 recently added support for metadata export using learning standards such as LOM. These efforts justify that in a near future this type of repositories could converge.

2.3.2.2 Software for Learning Objects Repositories

A LOR is a repository that manages learning objects and their respective metadata. Although much useful work has been done and considerable progress made, the existing software for creating LORs is still to a great extent confined to home-grown software.

Table 2.5: Learning Objects Repositories software.

Repository	License	Metadata	Communication
ARIADNE	Free	DC,LOM,MLR	SPI,SWORD,PENS,SQI,OKI
Flori	Free	-	-
HarvestRoad Hive	Commercial	LOM/SCORM	OAI, OKI
IntraLibrary	Commercial	DC, LOM/SCORM	SWORD,SR*

Table 2.5 presents a list of software for creating LOR with the licensing and supported standards. The majority of the LORs available nowadays are Web applications developed using non-repository software due to the lack of support of these tools for the e-learning requirements as stated above. A non-exhaustive list of LOR is presented in Table 2.6. All these LORs are free of charge provided that they are not used for commercial purposes.

Most LORs store multidisciplinary content rather than specialized domain content. Examples of specialized domain content are the programming exercises that are mostly enclosed

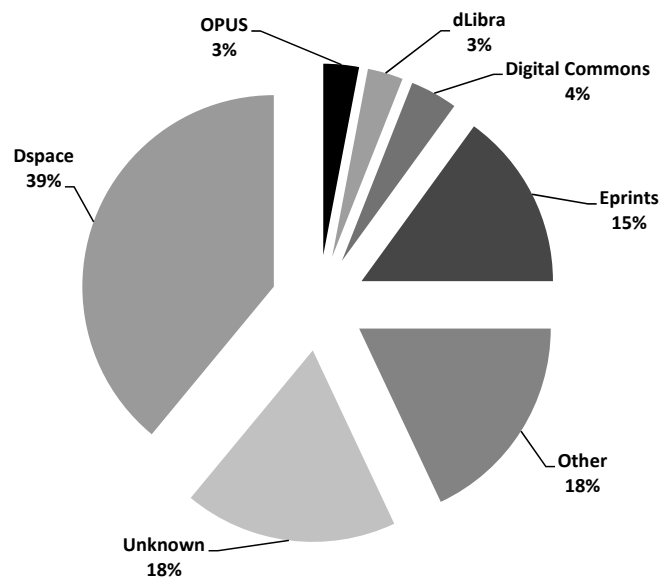


Figure 2.10: Usage of digital library software worldwide.

in Online Judges such as the UVA (University of Valladolid) Online Judge - an automated judge for programming exercises created in 1995 with the aim of training users who participate in worldwide programming competitions. The judge is hosted by the University of Valladolid and its archive contains over 2700 exercises. The set of exercises is continuously being extended but it lacks interoperability features such as standardization of the exercises content as learning objects and implementation of communication specifications to improve accessibility for the educational community of teachers and students [VRV⁺11].

In fact repository interoperability is one of the main concerns stated by users of a comprehensive survey [Tea06] made by the Jorum team. Existing repositories usually store learning objects from several domains (referatories). They provide on-line catalogues through specific and tightly coupled web-based interfaces. These interfaces provide tools for the management throughout the life cycle of learning objects, namely, submission, comment/review, browse/search and download. It has also been noticed that most of the existing repositories do not store actual learning objects. They just store meta-data describing Learning Object (LO), including pointers to their locations on the Web, and sometimes these are dangling pointers. Although some repositories list a large number of pointers to LO, they have few instances in any category, such as programming problems. Last but not least, the LO listed in these repositories must be manually imported into a LMS. A specialized system such as an Assessment System to perform specific evaluations or an intelligent tutor system cannot query the repository and automatically import the LO it needs. In summary, current repositories provide specialized search engines for LO and are not adequate for feeding specialized services.

Table 2.6: Learning Objects Repositories.

Repository	#LO	Type
Bepress	52768	Legal
BerkleeShares	123	Music
Connexions	19783	General
GEM	47321	General
LeMill	43734	General
LO.NET	302	General
LRE (European Schoolnet)	>200K	General
Maricopa	1818	General
Merlot	30398	General
Scriptorium	512	Humanities
Wisc-Online	2133	General

2.4 Assessment Systems

The most effective type of exercises in any learning domain, both for knowledge acquisition and for student grading, are seldom quizzes. Text file automatic evaluation differs significantly from quiz evaluation. One good example is programming exercises. Automatic assessment of programming exercises has become an important method for grading students' exercises as well as giving feedback on the quality of their solutions [AM05, SHP⁺06, TGPS08].

This section surveys Assessment Systems (AS) for programming exercises focusing on their interoperability features. Nowadays there are a large number of AS referenced in several surveys found in literature [RSZ10]. The majority of these surveys enumerates and compares the same set of features such as how the analysis of the code is made, how the tests are defined or how grades are calculated. These surveys seldom address the AS interoperability features, although they generally agree on the importance of the subject, due to the comparatively small number of systems that implement them. This lack of interoperability is felt at content and communication levels. Both levels rely on the existence of specifications that uniformly describes the content of programming exercises and the way they should be shared among the systems that are typically coupled with AS. Examples of these systems are LMSs, PCMSs, OJs, LORs and IDEs.

The main goal of this section is to gather information on the interoperability features of the existent ASs and to compare them regarding a set of predefined criteria such as content specification and standard interaction with other tools. The intended benefit of this survey is twofold: 1) to fill the gap on ASs interoperability features found in most surveys; 2) to

help instructors, educational practitioners and developers when they have to choose an AS to integrate in their e-learning environments.

2.4.1 Evolution of Assessment Systems

In recent years, programming courses in secondary schools and universities are characterized by extensive curricula and large classes. In this context, the assessment of programming assignments poses significant demands on the instructor's time and other resources [DLO05]. This demand stimulated the development of automated learning and assessment systems in many universities [AM05]. These systems assess programming exercises and assignments submitted by students, and provide evaluation data and feedback. They present a wide variety of features related with programming language support, evaluation type, feedback, interoperability, learning context, security and plagiarism.

Early systems [Ree89, JU97, MGH98, SMK01] assess exercises and assignments in a single programming language respectively, Pascal, ADA, Prolog and Scheme. With the advent of the Internet and the increase of platforms heterogeneity, web interfaces began to play an important role in the dissemination of several systems [PRS⁺03, Jue03, LS03, BGNM04]. The last two were among the first AS to support multiple programming languages, such as Java, C++ and the C.

The standard way of evaluating a program is to compile it and then execute it with a set of test cases comprising input and output files. The submitted program is accepted if compiles without errors and the output of each execution is what is expected. This evaluation strategy has been shown to bring undesirable pedagogical issues such as student frustration and confusion [Tan09b, TC10]. Several systems [JU97, SMK01, PRS⁺03, Jue03, BGNM04, MMR06] test not only the behaviour of single programs but also analyse the structure of source code. This approach guarantees that the program was written in a particular way, following a particular algorithm or used certain data structures. To assess the correctness of student submissions Edwards [EP06] used unit tests defined by teachers. Another important issue is the non-determinism of the program outputs where different correct (or acceptable) solutions to the same programming exercise may not always produce exactly the same output [Tan09a]. Mooshak [LS03] deals with non-determinism using dynamic correctors invoked after each test case execution. For instance, if the solution is a set of values that can be presented in any order then a dynamic corrector can be used to reduce the output to a normal form.

Depending of the learning context (competitive or curricular) the systems may provide feedback to help students to solve a particular exercise. The feedback generation relies on static and dynamic program analyses [AM05]. The development of AS with high quality feedback (e.g. compilation errors, execution errors, execution tests) show good results

[MKKN05, HGST05] and along with visual, incremental and personalized feedback should shape the future regarding this topic [SG10].

The AS interoperability is also an important issue to address. An evaluator should be able to participate in learning scenarios where teachers can create exercises, store them in a repository and reference them in a LMS and where students can solve exercises and submit to AS who delivers an evaluation report back to students. [LJ99, BBF⁺11] were early systems that try to address this issue allowing the integration with course management systems. Nowadays with the advent of SOA the trend is service orientation rather than component-based systems. An evaluator system as a service will automate the existent business logic in distributed e-learning scenarios allowing more flexibility in the comprised workflows and keeping the systems simple and easy maintainable. Leal and Queirós [LQ10h] specified a service for programming exercises evaluation in a well-known e-learning framework called the E-Framework. This work was used in the Edujudge project with positive results [VRV⁺11].

Luck and Joy [LJ99] analysed security issues on AS covering robust environments, privacy, plagiarism and data integrity. Security can be handled from ad hoc solutions to solutions based on virtual machines in order to execute the programs on a safe and controlled environment. Other concerning is the increase of plagiarism [ELC07, CKLO03].

Regarding the learning context, AS can be used in two contexts: curricular and competitive learning. In the former, teachers use practical classes, assignments and examinations to evaluate students' evolution. The latter relies on the competitiveness of students to increase their programming skills mostly in computer programming contests. In this last context, automated judge systems (or Online Judges) are used to run programming contests and to practice for such contests. These systems include automatic evaluators and many of these systems organize their own contests, such as, Mooshak [LS03], University of Valladolid Online Judge (UVA-OJ), SPOJ (Sphere Online Judge), DOMJudge and others.

2.4.2 Recent Surveys

In the last decade several surveys appeared reporting AS features and trends. The history of the field from 1960s is characterized by a number of projects that automatically assess student programming exercises using a test-based approach [DLO05]. Three generations of AS (Figure 2.11) were identified.

The first-generation was represented by several initiatives to automate testing, however their usability was confined to their particular computing laboratories. The second generation was characterized by command-line-based AS. The third generation made use of web-based technologies to leverage the use of AS worldwide and provide additional support for educators in the form of assessment management and reporting facilities. The paper also mentions four

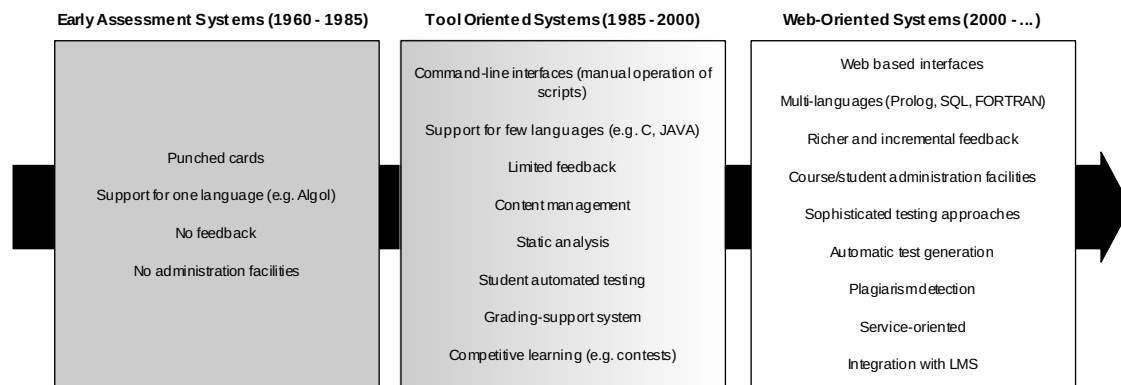


Figure 2.11: Generations of the Assessment Systems.

development directions in this field: evaluation of GUI programs, meta-testing (evaluation of the students' tests), service orientation adoption and use of interoperability standards. Ala-Mutka [AM05] organizes AS features according to whether they need execution of the program (dynamic analysis) and/or can be evaluated from the program code (static analysis). In one hand, dynamic analysis is often used to assess functionality, efficiency, and testing skills. In other hand, static analysis is used to provide feedback from style, programming errors and software metrics. The authors conclude that automated systems approach should always be pedagogically justified and state that systems are in-house built and no common standards or interfaces exist.

Liang [LLXW09] details dynamic and static analysis methods of existing AS. The paper also enumerates several unsolved issues in this area such as security, algorithms for automatic generation of test data in dynamic analysis and low accuracy and precision of correctness in static analysis. Finally the authors claim as new directions in the AS development the content standardization.

Ihantola [IAKS10] gathers information on AS from 2006 to 2010 and discuss their major features such as tests definition, resubmission policies and security features. The author expects new research to emerge from the following fields: integration of automatic assessment on LMS and automatic assessment of web applications.

Romli [RSZ10] enumerates approaches for automatic programming assessment, test data generation and integration of both. The authors conclude that there is a lack of use of existing test data generation techniques (commonly used to test software) in the scope of automatic programming assessment. The same survey made an exhaustive study on 41 assessment tools that appeared in the last 50 years focusing on the evaluation methods and test data generation techniques used. Dynamic analysis is the most used method to assess programs with 74% of the tools studied using it. This is explained since program correctness is the most important quality factor while evaluating a program. In dynamic analysis the

test data assumes a relevant role. The process of creating tests can be labour demanding. Manual generation is time-consuming and error-prone and seldom covers the potential range of a program. In spite of these issues, the study shows that the most used method for feed the assessment systems with test data is through manual data entry. This is due to the complexity inherent to the automatic generation of test data.

Beyond these facets, all above surveys stated the need for interoperability and security on AS. The former can be achieved by the creation and adoption of content and communication standards. The latter is a well-know issue that should not be overlooked and can be addressed by the use of secure environments (sandbox) to execute untested code and algorithms to filter out malicious code.

2.4.3 Assessment System Interoperability

The previous surveys show that interoperability is the main trend on AS. Moreover, this topic was never analysed in the above surveys. Thus, this subsection analyses existing AS regarding their interoperability features. Given the multiplicity of systems found a multi-criteria approach was applied for the selection of tools based on its effective use. The tools should be flexible enough to allow the configuration of exercises and the management of users. The former covers not only the selection of existing exercises on the evaluation tool but also the support for adding new exercises. The latter refers to the support of the tool to select users that will solve the exercises.

Based on this multi-criteria approach 15 tools were selected. After the selection of the tools, an iterative process was applied to identify which facets (current sub-subsections) will be used to verify the interoperability maturity level of the selected tools. A set of facets was initially identified based on the issues and trends raised on the previous surveys in conjunction with the author's background in working with interoperability on automated assessment. After reading the published papers of the tools and consulting their official websites, a revised set of facets was obtained. Figure 2.12 shows the selected facets.

These facets are also synchronized with the main objective of a typical automatic evaluation system - to evaluate a user's attempt to solve a programming exercise and produce an assessment result. Each facet includes three interoperability maturity levels:

Level 0 - manual configuration of data;

Level 1 - data import/export;

Level 2 - services invocation.

In order to belong to Level 0, the evaluation tool must support the configuration of data

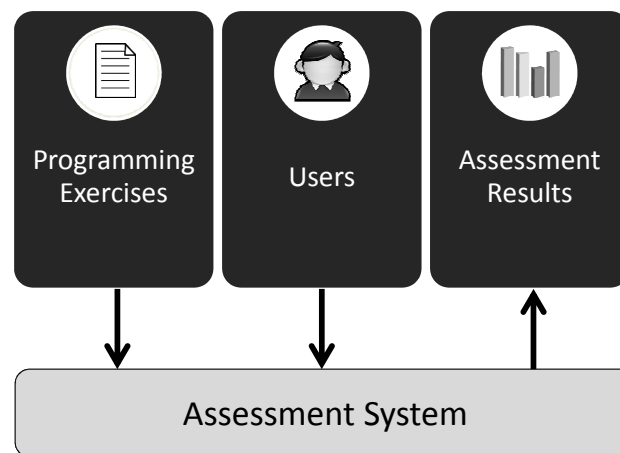


Figure 2.12: Interoperability facets of Assessment Systems.

by allowing either the selection of existing data or the addition of new data. In the Level 1, the evaluation tool must also support the import/export of data from/to other sources. In the last level, the evaluation tool should also support the communication with other tools through the invocation of web services. In the next sub-subsections the three facets are detailed and for each facet the respective interoperability maturity levels of the selected AS are presented.

2.4.3.1 Programming Exercises

Nowadays one can find a large number of programming exercises. Despite their number, these exercises exist only in AS silos and seldom include mechanisms to share the exercises among researchers and instructors in an effective manner. Moreover, each of these systems dictates the persistent format of an exercise that may not be interoperable with other automatic evaluation systems. This is a significant barrier in the creation and sharing of programming exercises and can only be addressed through the standardization of exercise content and its storage on public repositories. Based on these facts, the abstract maturity levels are the following:

Level 0 - manual configuration of exercises;

Level 1 - import/export of exercises;

Level 2 - integration with repository services.

In the Level 0, the evaluation tool should support the selection of exercises and the addition of new exercises. In this level, the tool relies on ad-hoc or internal formats to describe

exercises data.

In the Level 1, the evaluation tool should also provide mechanisms to import/export exercises from/to other sources. In this level, the tool must explicitly support an exercise format. There are few exercise formats. Typically an exercise format can be obtained by modelling a programming exercise into a LO definition. This definition describes an exercise as a learning package composed by a set of resources (e.g. exercise descriptions, test cases, solution files) and a manifest that describes the package and its resources in terms of its contents, classifications, life-cycle and several other relevant properties.

In the Level 2, the evaluation tool should also support the communication with other tools, typically LOR, through web services. A LOR is a system used to store and share learning objects. The repository should support simple and advanced queries to retrieve LO and export them to other systems through a set of web services. In this communication, a service broker (e.g. exercise format conversion service) can be used when the evaluator does not support the format of the exercises stored in the repository. Based on these levels, the following table enumerates for each tool the maturity level regarding the management of programming exercises. According to the Table 2.7, all systems support the configuration

Table 2.7: Programming exercise facet (P-partial and F-full)

Systems	Level 0	Level 1	Level 2
AutoGrader	F	-	-
BOSS2	F	-	-
CourseMaker	F	-	-
CTPracticals	F	-	-
DOMJudge	F	-	-
EduComponents	F	-	-
GAME	F	-	-
HUSTOJ	F	P	-
Moe	F	P	-
Mooshak	F	F	F
Peach3	F	P	-
Submit!	F	-	-
USACO	F	-	-
Verkkoke	F	F	-
Web-CAT	F	F	P

of exercises. However, only six tools provide a way to export exercises and only three support bidirectional transfers with other sources. These systems often use exercises formats. HUST Online Judge uses FPS (FreeProblemSet) as an XML format for transporting exercises

information between Online Judges. Peach3 system uses PEF (Peach Exchange Format) as a programming task package containing all task-related information and serving as a unit for storage and communication. Verkkoke system relies on SCORM packages to wrap all the exercise data. The second level of interoperability is only achieved by Mooshak and partially by Web-CAT. Mooshak is a system for managing programming contests on the Web. Its version 1.6a2 supports the communication with repositories complying with the IMS DRI specification using a broker service responsible for the conversion between formats. Web-CAT is an automatic grading system using student-written tests. This system can communicate with other repositories, such as CollabX, through specific plug-ins. Unlike Mooshak, the interaction with repositories is not standard-based.

Based on these facts one can conclude that most systems use internal and proprietary formats. Those who adhere to explicitly formats do not reach a consensus to use a single format. This non-compliance to a single format leads to the standard fragmentation for describing exercise content. One solution to address this issue is instead of creating new formats one should start looking for broker services responsible for the conversion between formats. Other issue is the relation with repositories of learning objects. The majority of AS store the exercises inside their systems hindering the proliferation and sharing of exercises. In order to communicate with repositories the evaluation systems must follow communication standards rather than ad hoc implementations.

2.4.3.2 Users

In order to select and solve exercises users must be authenticated in the evaluation system and have authorization to submit their solutions. The users' facet also specialises the abstract maturity levels with the following:

Level 0 - manual configuration of users;

Level 1 - import/export of users;

Level 2 - integration with user directories services to provide authentication/authorization.

Table 2.8 shows the maturity level of automatic evaluation tools regarding the users' facet.

In the Level 0, the evaluation tool should support the configuration of user's data.

In the Level 1, the evaluation tool should also provide mechanisms to import/export users from/to other sources. In this level, the tool can export a list of users based on standard formats. There are few standards that formalize users' data and how data is sent. Two know-standards are the IMS Learner Information Services (LIS) and the IMS Learner Information Package (LIP). The former is the definition of how systems manage the exchange of

Table 2.8: Users facet (P-partial and F-full)

Systems	Level 0	Level 1	Level 2
AutoGrader	F	F	P
BOSS2	F	-	-
CourseMaker	F	-	-
CTPracticals	F	F	P
DOMJudge	F	F	P
EduComponents	F	F	P
GAME	F	-	-
HUSTOJ	F	-	-
Moe	F	-	-
Mooshak	F	F	P
Peach3	F	-	-
Submit!	F	-	-
USACO	F	F	-
Verkkoke	F	F	-
Web-CAT	F	F	-

information that describes people, groups, memberships, courses and outcomes within the context of learning. The IMS LIS is focused on the connection between an LMS and an AMS. The latter addresses the interoperability of internet-based learner information systems with LMS. It describes mainly the characteristics of a learner.

In the Level 2, the evaluation tool should also support the communication with other tools to provide authentication and authorization facilities. User authentication is based on directory services such as LDAP or Active Directory. User authorization relies on AMS that manages academic processes such as the enrolment of students in courses, the management of grades or the payment of fees. They are the best candidates to offer authorization services since they store information about courses and students enrolled in them. The communication with AMS is not standardized. This fact burdens the integration of AMS with evaluation systems that must resort to ad hoc solutions.

According to the Table 2.8, all systems support the manual configuration of users for a specific assignment or course. More than a half of the systems studied allow the import/export of users in non-standard formats. However only five partially support the communication with authentication services (mostly with LDAP). One can conclude that AMS are still immature in terms of standard communication with other systems since any system was found interacting with it. AutoGrader, CTPracticals, EduComponents and Verkkoke benefit from the fact that they are integrated with LMS thus taking advantage of its authorization

facilities.

2.4.3.3 Assessment results

After the submission of the student the assessment system evaluates the program and returns an evaluation result.

The assessment results facet also specialises the abstract maturity levels with the following:

Level 0 - visualization of evaluation results;

Level 1 - export of assessment results;

Level 2 - integration with LMS.

In the Level 0, the evaluation tool should support the visualization of the assessment results. The result data is essential for the success of an assignment and can include feedback and grades. This information should be present to the user on the evaluation tool graphical interface.

In the Level 1, the evaluation tool should also export evaluation reports to other sources. There are few standards that formalize evaluation results. A formalization of an evaluation report can be found in the Evaluation service [LQ10h] - a contribution for the E-Framework. An implementation of this service evaluates an attempt to solve a programming exercise and produces a detailed report. This evaluation report includes information to support exercise assessment, grading and/or ranking by client systems. The report itself is not an assessment, does not include a grade and does not compare students.

In the Level 2, the evaluation tool should also communicate with other tools.

A typical scenario would be when the evaluation tool sends the grades to the LMS grade book. A common interoperability standard that is increasingly supported by major LMS vendors is the LTI specification. It provides a uniform standards-based extension point in LMS allowing remote tools and content to be integrated into LMS. Currently, only a subset (IMS Basic LTI) of this specification is implemented by the major LMS. This subset exposes a unidirectional link between the LMS and the application. For instance, there is no provision for accessing run-time services in the LMS and only one security policy is supported. Table 2.9 shows the maturity level of AS regarding the assessment results facet. It shows that all systems present the evaluation results to users and the majority allows their exportation in non-standard formats. Regarding the communication with other systems, four systems support the communication with LMS by providing the evaluation results on the LMS grade book. AutoGrader, CTPracticals and EduComponents are integrated with

Table 2.9: Assessment results facet (P-partial and F-full)

Systems	Level 0	Level 1	Level 2
AutoGrader	F	F	P
BOSS2	F	-	-
CourseMaker	F	-	-
CTPracticals	F	F	P
DOMJudge	F	F	-
EduComponents	F	F	P
GAME	F	-	-
HUSTOJ	F	-	-
Moe	F	-	-
Mooshak	F	F	-
Peach3	F	F	-
Submit!	F	-	-
USACO	F	-	-
Verkkoke	F	F	P
Web-CAT	F	F	-

specific LMS, respectively, CascadeLMS, Moodle and Plone. Verkkoke is the only that do not depends on a specific LMS and can be integrated on any LMS that supports the SCORM specification.

2.4.4 Conclusions

This section starts by synthesising the interoperability facets of the AS included on the above survey. Figure 2.13 depicts the percentage of interoperability maturity of each AS.

One can conclude that half of the studied systems did not reach 50% of the maturity rate. This illustrates that there are a lot to do in this field regarding the integration of AS with other systems. Figure 2.14 depicts the coverage of interoperability features of the AS studied organized by facet.

The major conclusion to take is that there is no specific trend on interoperability facets since the distribution of interoperability coverage is equitably distributed among the three facets. Based on this study, two issues were detected that can hinder the proliferation of AS: the lack of content standards for describing programming exercises and to communicate with other e-learning systems. This section fills the gap existent in most surveys since all of them point to interoperability as an issue for AS use and a trend for AS development but never explained in detail what are the paths to follow in order to achieve interoperability

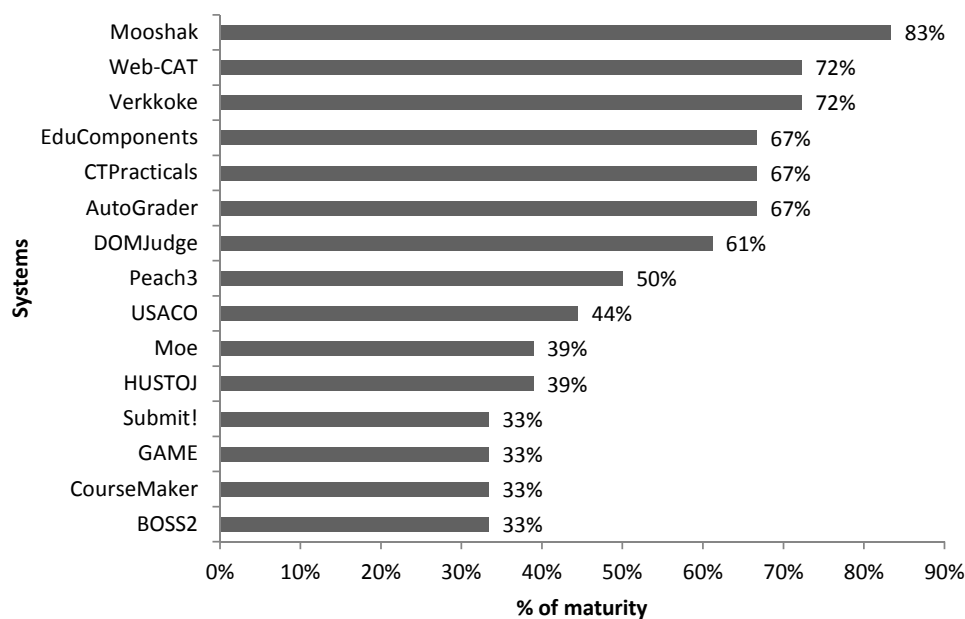


Figure 2.13: Interoperability maturity percentage level of Assessment Systems.

on this domain. The results achieved on this survey may also prove useful to instructors and computer science educators when they have to choose an AS to be integrated in their e-learning environment.

2.5 Summary

This chapter traces the evolution of e-learning systems from monolithic systems to specialized services. These services can be easily recombined in different learning processes to assist on the teaching-learning process. This chapter focused on learning processes within domains with complex evaluation such as computer programming learning. In this domain there are several candidates to offer services such as those referred by Rongas, Kaarna, and Kalviainen [RKK04], namely: LMSs, LORs and ASs.

In the LMS side, an interoperability comparative study was conducted to select an LMS on which to base the development of e-learning systems integrating heterogeneous components. Two LMSs vendors were chosen (Moodle and Blackboard) and their interoperability features were analysed by splitting in two facets (learning content management and academic management) reflecting the broad classes of systems of a typical LMS operational environment.

In the LOR side, an interoperability study was conducted on the existing repository software distinguishing two categories of repositories - digital libraries and learning objects repositories. It also focused on the distinction between the actual repositories and the software used

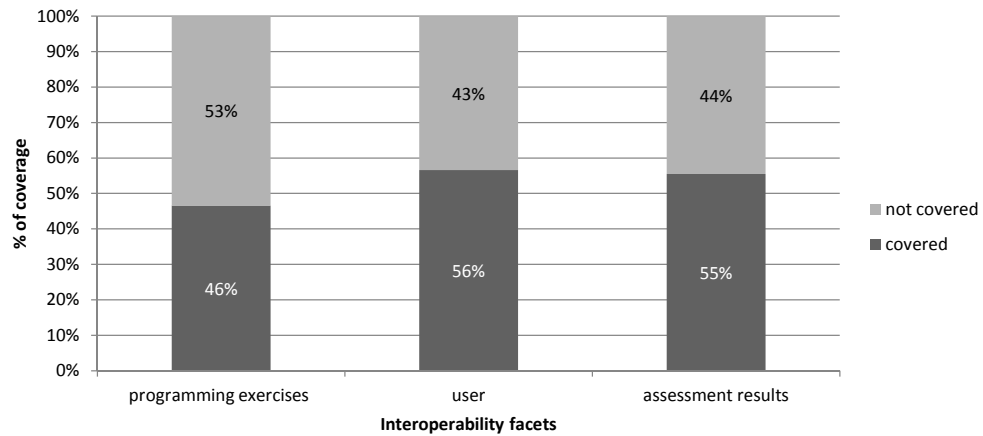


Figure 2.14: Coverage of Assessment Systems interoperability features.

to implement them, highlighting the differences in their interoperability features.

Finally, a survey was made on ASs for programming exercises focusing on their interoperability features. The survey gathered information on the interoperability features of the existent ASs and compared them regarding a set of predefined criteria such as content specification and interoperation standards with other tools.

Chapter 3

E-Learning standards

*"The nice thing about standards
is that there are so many to choose from"*

Tanenbaum, 1981, p. 221

The evolution of e-learning systems in the last two decades was impressive. In their first generation, e-learning systems were developed for a specific learning domain and had a monolithic architecture [DOL⁺07]. These systems were often developed as in-house solutions making it difficult, if not impossible, to foster their interoperation with other systems and to share the content they manage. In order to address these shortcomings, several organizations have created standards and specifications to be used in the systems design and implementation ensuring the following benefits [VA06]: 1) interoperability - content can be easily disseminated within heterogeneous systems solving problems such as translation, communication and information exchange; 2) re-usability - content can be assembled, disassembled, and re-used quickly and easily; 3) manageability - systems can track information about learners and content; 4) accessibility - learners can access the appropriate content at the appropriate time on the appropriate device; 5) durability - content is produced once and transplanted many times in different platforms and systems with minimum effort; 6) scalability - learning technologies can be expanded in functionality in order to serve broader populations and organizational purposes.

A **standard** can be defined as "documented agreements containing technical specifications or other precise criteria to be used consistently as rules, guidelines, or definitions of characteristics, to ensure that materials, products, processes and services are fit for their purpose"¹. The process of creating standards (depicted in Figure 3.1) is achieved through the co-operation

¹Standards definition - ISO/IEC (ISO/IEC JTC1 Directives) - <http://www.iso.ch/infoe/intro.html>

of several types of participants of the e-learning community such as developers, vendors and users [VA06]. This process comprises four steps.

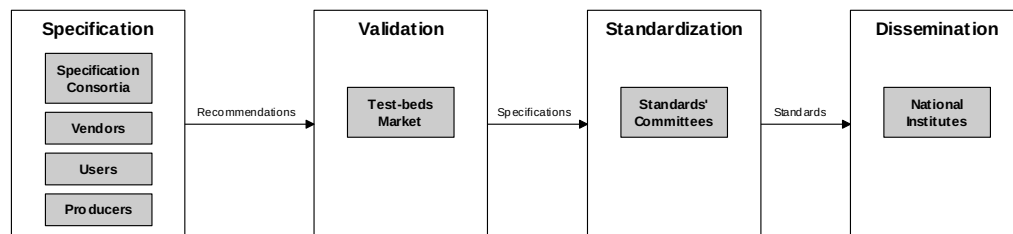


Figure 3.1: Steps for establishing e-learning standards [VA06].

In a first step a consortia, for instance the IMS Global Learning Consortium (GLC) or European Committee for Standardization (CEN), makes a needs analysis and a specification of requirements for a specific learning domain or workflow. Then, educational vendors integrate and validate their e-learning systems' conformance regarding the previous requirements. In a third step, accredited standards bodies such as IEEE International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) refine and consolidate the specifications and recognises them as standards. Finally, the accepted standards are disseminated to all the participants of the e-learning community.

This process clearly distinguishes between a specification and a standard. The former is an experimental and evolving work in progress where many individuals and organizations contribute with recommendations. The latter is an accredited standard, which is based on actual implementations and experience, and provides very clear and unambiguous guidelines for implementation and conformance. Despite their differences, both are composed by a set of documents which are published on the Web. Typically they provide [Fri04b]:

- a data/conceptual model specifying the standard's "normative" content in abstraction;
- "bindings" specifying how the data model is expressed in a formal idiom (e.g. XML);
- an API or "service definition" defining points of contact between cooperating systems.

The aim of this study is to gather information on e-learning standards in order to choose the most suitable for the domain of automatic evaluation of exercises. The organization of this chapter is influenced by several studies found in the literature [JBK05] [VA06]. In these studies the outcomes of the standardization efforts on e-learning interoperability are divided in two levels (content and communication). In this chapter a new level (Frameworks) was added because of its relevance in the last years on the development of new e-learning systems based on SOA. Thus, the three levels are:

Frameworks - development of infrastructures and services to support learning;

Content - specification of the information models (e.g. format, syntax and semantics of data) to be exchange;

Communication - specification of the protocols and interfaces to guarantee a standard communication among heterogeneous systems.

The following sections detail these levels. For each level, the most prominent standards and specifications and their main contributions in the e-learning field are enumerated. The selection of the standards and specifications was based on surveys [QL11c, FCN⁺11]. One such survey [FCN⁺11] shows the number of publications related to a variety of e-learning standardization fields from the IEEE Digital Library and ACM Digital Library in the last 30 years (1980-2010). Figure 3.2 shows the results of this survey.

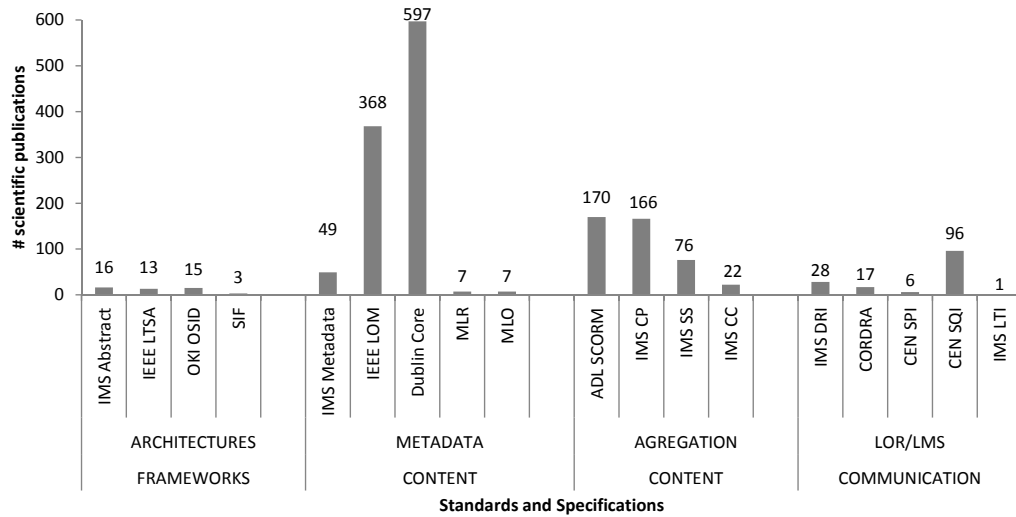


Figure 3.2: E-learning specifications and standards publications (1980-2010).

For a more descriptive and detailed information about organizations, projects, activities and standards one can consult the CEN WS-LT Learning Technology Standards Observatory² - an accessible web based repository for e-learning technology standards.

3.1 Frameworks

The architectures of e-learning platforms had a considerable evolution in the last two decades. Starting with the early monolithic systems developed for specific learning domain to domain-independent systems featuring reusable tools that can be used virtually in any e-learning

²Official Web site: <http://www.cen-ltso.net/>

course [DOL⁺07]. These last systems follow a component oriented architecture in order to facilitate tool integration. Integrated environments have been successfully used to leverage the advantages of Information Communication Technology (ICT), but have also been target of criticism. These systems based around pluggable and interchangeable components, led to oversized systems that are difficult to reconvert to changing roles and new demands such as the integration of heterogeneous services based on semantic information, the automatic adaptation of services to users (both learners and teachers), and the lack of a critical mass of services to supply the demand of e-learning projects. These issues triggered a new generation of e-learning platforms based on services that can be integrated in different scenarios. This new approach provides the basis for SOA. In the last few years there have been initiatives to adapt SOA to e-learning [LQ10a]. These initiatives, commonly named e-learning frameworks, had the same goal: to provide flexible learning environments for learners worldwide. Usually they are characterized by providing a set of open interfaces to numerous reusable services organized in genres or layers and combined in service usage models. These initiatives use intensively the standards for e-learning content sharing and interoperability developed in the last years by several organizations such as Advanced Distributed Learning (ADL), IMS GLC, Institute of Electrical and Electronics Engineers (IEEE).

Over the years the word framework has been used to define a work environment specially designed to solve common and complex problems in different domains. Due to its broad definition it is often used as a buzzword, especially when applied to software. A software framework may include support programs, runtime environments, code libraries and other tools, in order to assist the developer in a software project. Usually the functions of a framework are exposed through an API. The code provided by the framework is usually divided in frozen-spots (services already developed in the framework) and hotspots (set of common code that must be overridden or specialized by user code) [MdL01]. Hence, this twofold code feature amongst with the inversion of control is one of distinguishing keys that separate the current frameworks from generic code libraries. An e-learning framework can be defined as a specialized software framework. In the e-learning field, this term has been associated with several initiatives to adapt SOA to e-learning.

Based on service oriented approaches the process of moving from a framework to a working implementation can be defined by four key concepts [WBR04]:

Broad Vocabulary - describes all possible "services" for a domain such as e-learning;

Reference Model - combines these services for specific learning-teaching requirement;

Design - specifies the use of standards and specifications for these combinations;

Artifact - implements (software, process, workflow) a design.

The relationship of the key concepts of e-learning frameworks is shown in Figure 3.3.

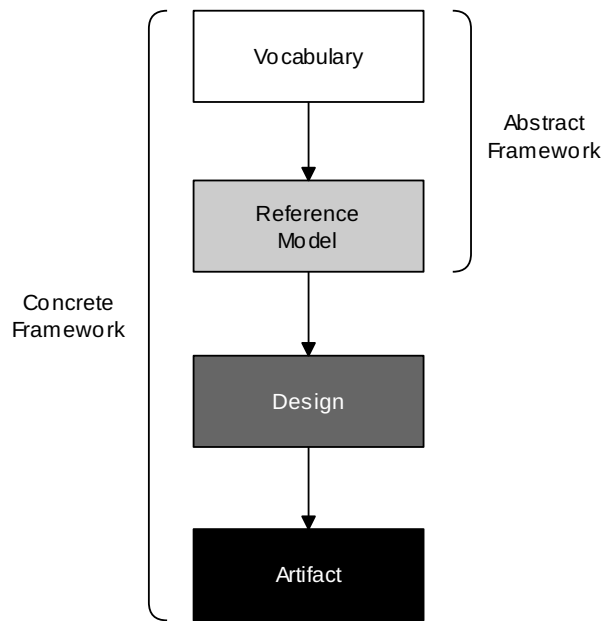


Figure 3.3: Simple Framework model [WBR04].

A Framework provides a vocabulary of Services (e.g. digital repositories services), from which a Reference Model (e.g. describing content management) is derived. A particular Design (e.g. repository management application) is modelled based on the Reference Model which is then implemented as an Artifact.

While abstract frameworks provide a broad vocabulary and a reference model for the development of e-learning systems, concrete frameworks provide also designs and/or artifacts. The remainder of this section categorizes the e-learning frameworks based on these groups.

3.1.1 Abstract Frameworks

Abstract frameworks aim only at the creation of specifications, recommendations and best practices for the development of e-learning systems. In this subsection three initiatives belonging to this category are detailed, more precisely, the IEEE Learning Technology Systems Architecture³, the Open Knowledge Initiative⁴ and the IMS Abstract Framework⁵, and , in the chronological order of their first definition.

³Architecture & Reference Model - Working Group 1 Web Site - <http://ltsc.ieee.org/wg1/>

⁴Open Knowledge Initiative (OKI) Web Site - <http://sourceforge.net/projects/okiproject/>

⁵IMS Abstract Framework Web Site - <http://www.imsglobal.org/af/>

3.1.1.1 IEEE Learning Technology Systems Architecture

The IEEE Learning Technology Standards Committee (LTSC) is chartered by the IEEE Computer Society Standards Activity Board to develop internationally accredited technical standards, recommended practices, and guides for learning technology. The IEEE LTSC has developed a number of internationally accredited standards. The IEEE Learning Technology Systems Architecture (LTSA) is one of the standards that define a pedagogical and implementation neutral high-level architecture for information technology-supported e-learning systems. This standard, whose first draft was presented in 1996, covers a wide range of systems and promotes interoperability and portability by identifying abstract, high-level system interfaces. The LTSA system components, depicted in the Figure 3.4, identify the abstract and high-level interfaces for e-learning systems.

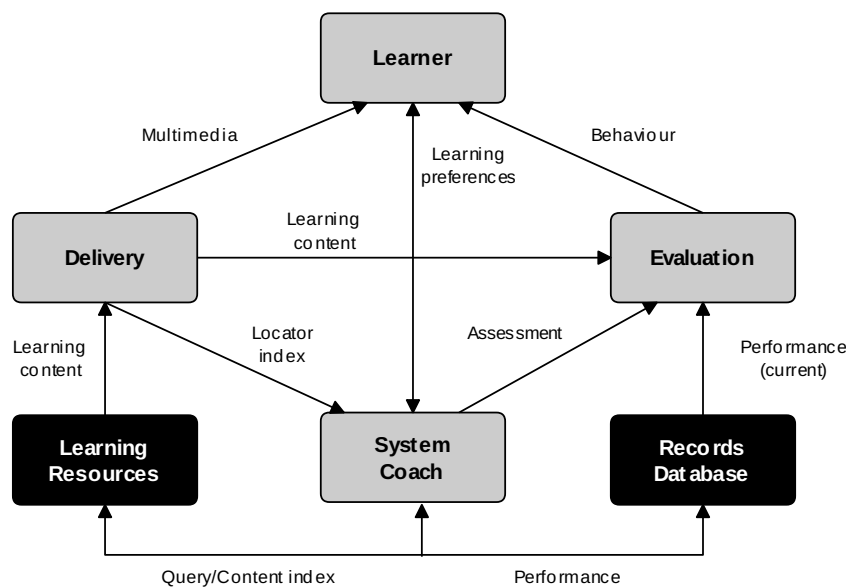


Figure 3.4: Learning Technology System Architecture [FT99].

The LTSA system components are based on: 1) processes (depicted as gray rectangles): are described in terms of boundaries, inputs, process (functionality), and outputs; 2) stores (depicted as black rectangles): are described by the type of information stored, and by search, retrieval, and updating methods and 3) flows (arrows): are described in terms of connectivity (one/two-way, static/dynamic connections, etc.) and the type of information across the flow. Currently the LTSA working group status is defined as inactive at the web site of this project. Despite the inactivity, several works can be found in the literature focusing on the extension of this architecture [KSSM10].

3.1.1.2 Open Knowledge Initiative

The Open Knowledge Initiative (OKI) is a project created in 2001 and is currently leaded by the Massachusetts Institute of Technology. The main goal of this project is to provide a framework for higher education learning systems. The result of this initiative is an open and extensible architecture that specifies, based in the concept of Open Service Interface Definition (OSID), how the components of an educational software environment communicate with each other. The OKI OSID define standard interfaces that allow one application (an OSID consumer) to access specific data and functionality from another application (an OSID provider). For example, a digital repository can make its collections of learning objects accessible to a LMS by exposing an implementation of the Repository OSID. The LMS then needs only to call functions of the digital repository as defined by the OSID. An architectural view of the framework can be seen in Figure 3.5.

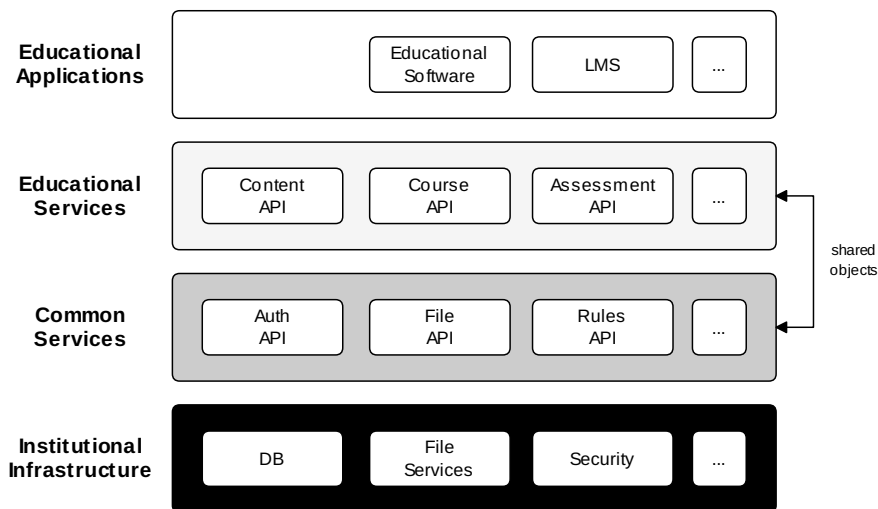


Figure 3.5: OKI architecture.

Recently, OKI has produced a Java OSID binding. There are also bindings for other programming languages such as PHP, MS .NET and C. In 2005, OKI announced the creation of XOSID (XML OSID) providing a language neutral XML representation of the OSID, which until the date were only available as Java API.

3.1.1.3 IMS Abstract Framework

The IMS GLC is a global coalition of academic, commercial and government organizations, working together to foster learning. The IMS Abstract Framework (IAF) is an abstract representation of the services and interfaces providing a context for IMS to develop its specifications. The IAF is represented as a layered model, as shown in the Figure 3.6.

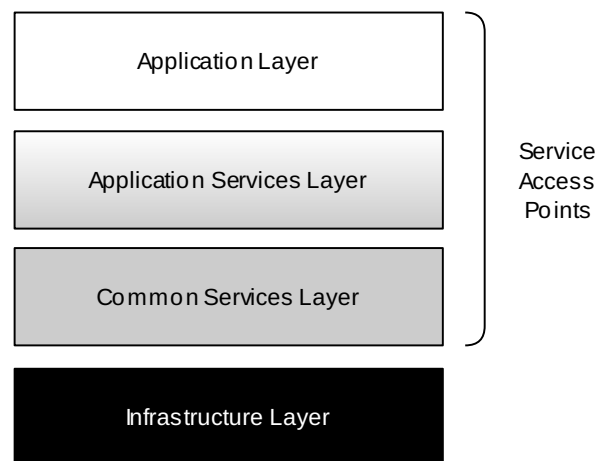


Figure 3.6: IAF layered model.

The model comprises four layers:

Application - set of systems, tools, agents, etc. providing a set of e-learning functionality;

Application services - set of services providing a specific e-learning functionality to the applications (e.g., course management);

Common services - set of services providing the generic services to be used by the application services (e.g., authentication);

Infrastructure – set of services enabling the exchange of the data structures in terms of physical communications.

The access to a service is made through its service access points. Each service has a single access point but, being an abstract framework, IAF does not address the implementation of the access points. It should be mentioned that this specification has not been updated since 2003.

3.1.2 Concrete Frameworks

Concrete frameworks extend the goals of abstract frameworks by providing also complete service designs and/or components that can be integrated in actual implementations of artifacts. In this subsection three initiatives are detailed, more precisely, the Open University Support System ⁶, the Schools Interoperability Framework ⁷ and the E-Framework ⁸.

⁶Official website of OpenUSS - <http://openuss.sourceforge.net>

⁷Schools Interoperability Framework (SIF) Web Site - <http://www.sifinfo.org/>

⁸Official website of e-Framework for Education and Research - <http://www.e-framework.org>

3.1.2.1 Open University Support System

The Open Source University Support System (OpenUSS) is a project to make a virtual university platform under an open-source license. The OpenUSS is a part of the CampusSource initiative, set up by the state of North Rhine-Westphalia, Germany. The OpenUSS is based on a component-oriented architecture divided into two types of components: foundation and extension. The former represents the main and domain-oriented components (e.g. Assistant, Student, Enrolment). The latter represents all the domain neutral functions of OpenUSS (e.g. Discussion, Chat, Lecture). Whilst the primary user groups are universities, the community of developers can contribute with API and reference implementations for the OpenUSS. OpenUSS is currently implemented in several universities in Germany and Mexico and over 10000 users rely on it worldwide. Nevertheless, OpenUSS has not been updated since 2001.

3.1.2.2 Schools Interoperability Framework

The Schools Interoperability Framework (SIF) is an industry initiative, supported by the SIF association. The objective of this association is to develop an open specification to enable educational applications, such as K-12 instructional and administrative software applications, to interact and share data. The framework is composed by two specifications: an XML specification for modelling educational data and a SOA specification for sharing that data within a SIF Zone. A SIF Zone is a logical group of applications, in which software application agents communicate with each other through a central communication point – the Zone Integration Server. A SIF Zone consists of a server and one or more software applications with a SIF Agent (a SIF-enabled application) distributing one or more SIF data objects over a network. Figure 3.7 shows an example of a SIF Zone.

Users are able to develop their own SIF Implementations. A SIF Implementation consists of one or more SIF Zones deployed to meet customer needs. The implementation must comply with the SIF Implementation Specification. This specification defines the architectural requirements and communication protocols for the software components and the interfaces between them. The specification makes no assumption of specific hardware/software needed to develop SIF applications.

Currently, the SIF specification is being used by numerous vendors in all USA states as well in the UK and Australia. The future plans for the framework are the development of a K-12 Data Model and an inclusion of a Web Services infrastructure. The new version (2.4) of the Specification is in the final stage of a release cycle.

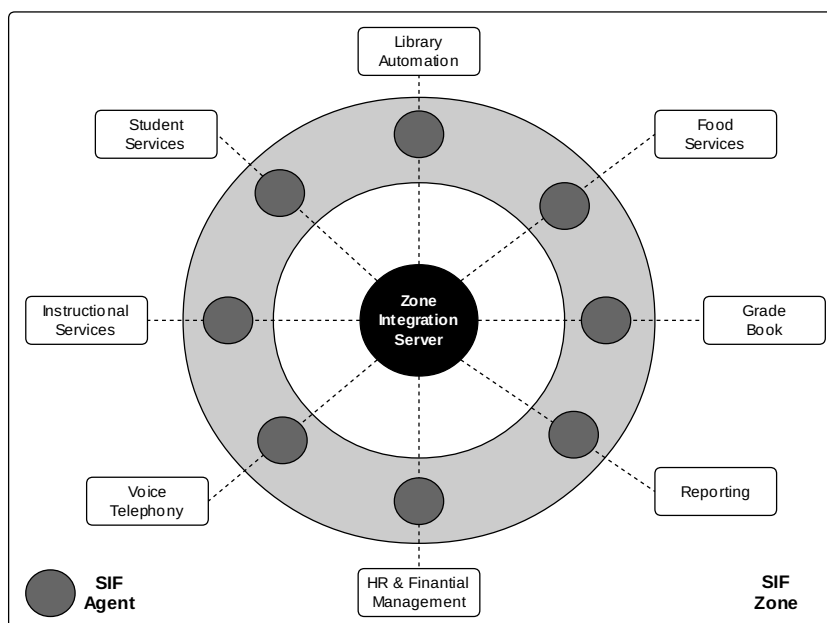


Figure 3.7: SIF architecture.

3.1.2.3 E-Framework

The E-Framework (E-F) is an initiative that seeks to promote the use of the SOA in the analysis and design of software for education and research. The E-F has four funding partners (JISC⁹, DEEWR¹⁰, SURF¹¹ and MoE¹²). The framework relies on a service-oriented approach that promotes the creation and reuse of software services that can be used by different applications in different contexts. As shown in Figure 3.8 the E-F is composed by the following components:

Service genre - a generic or abstract service expressed in terms of behaviours (e.g. authenticate, harvest, search). A Service Genre does not specify how a service works; rather it only specifies what a service should do;

Service expression - a realisation of a single service genre by specification of exact interfaces and standards used. Specifications and Standards used by Service Expressions are not defined by the e-Framework;

Service Usage Model - a model of the needs, requirements, workflows, management policies and processes within a domain.

⁹JISC Web site: <http://www.jisc.ac.uk/>

¹⁰Department of Education, Employment and Workplace (DEEWR) Web site: <http://www.deewr.gov.au>

¹¹SURF Web site: <http://www.surf.nl/>

¹²Ministry of Education (MoE) Web site: <http://www.minedu.govt.nz/>

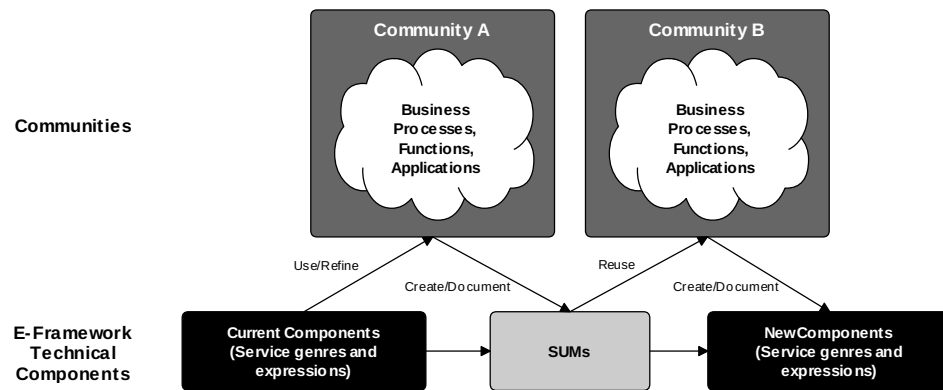


Figure 3.8: E-Framework.

All of the E-F components are described in template documents, which can be found on the E-F website. The E-F is currently the target of several updates through the large community of programmers who use the project wiki to submit new contributions.

3.1.3 Comparison of E-Learning Frameworks

The previous section details the frameworks overall architecture and main characteristics and divided them in two groups: abstract and concrete frameworks. The Figure 3.9 traces the evolution of these initiatives based on the previous grouping.

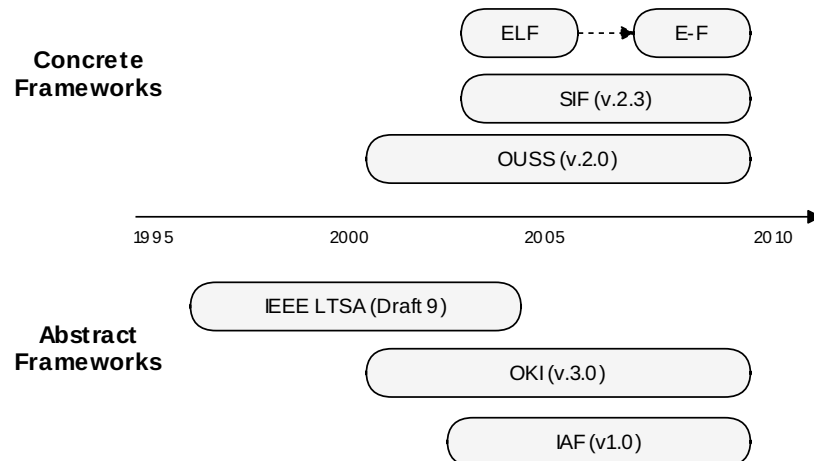


Figure 3.9: Evolution of e-learning Frameworks.

Figure 3.9 suggests that, in the last decade, the trend is the appearance of concrete frameworks rather than abstract frameworks. It is worth noting that none of the concrete frameworks mentioned actually implement artifacts. At most, these projects include user

contributed components, which can be integrated in artifacts for systems using the framework, but are not part of the framework itself.

Table 3.1: E-Learning frameworks survey.

Facets	Features	LTSA	OKI	IAF	SIF	E-F
Impact and Maturity	Creation date	1996	2001	2003	2003	2007
	1st vs. date	1996	2003	2003	2003	-
	Last vs. date	2001	2006	2003	2009	-
	Cited proj.	-	3	-	37	4
	Contributions	inactive	yes	yes	yes	yes
Architec. Models	Main Model	layered	layered	layered	flat	layered
	SOA	yes	yes	yes	yes	yes
Adopted Standards	Content	-	-	-	SCORM	SCORM,CP
	Metadata	LOM	LOM	LOM	LOM	LOM,DC
	Web Service Bindings	SOAP	SOAP	SOAP,REST	SOAP,REST	SOAP,REST
		-	JAVA,PHP	JAVA	JAVA	JAVA
User Groups	Framework	ESV	ESV	IMS	ESV	ESV
	End Users	HE	HE	HE	K-12	HE

Also five of these frameworks were compared regarding: impact and maturity, architectural models, adopted standards and user groups. From Table 3.1 one can conclude that some frameworks have a very low update frequency (IAF) for several initiatives and one of them is already inactive (LTSA). The frameworks with the most recent updates are the E-F and SIF. In the case of the E-F, it has been receiving great amount of input from the e-learning community. On the other hand, SIF is the most widely used framework with 37 cited projects in the project web site.

All the frameworks adhere to a service-oriented approach. Most of them use the layered architectural model. In this model components communicate only with components in the neighbouring layers. In particular, the LTSA has five layers in its architecture, but only one layer (system components) is normative. In the flat model there is no restriction to the communication among components. The SIF framework is a special case in applying this model since it uses a central component that orchestrates all the communication between applications. These frameworks use different main concepts to present their inner structure. OKI and IAF are an exception since they share their main concepts, which is probably due to the fact that these projects are cooperating [CDK⁺02].

In regard to standards some of them are common to almost all frameworks. For instance,

LOM for metadata content, WSDL for service description, Simple Object Access Protocol (SOAP) for web service and Java for language binding are common to all frameworks.

Finally, the Educational Software Vendors are the most common framework users, with the exception of IAF. IMS uses the framework to develop internal specifications (e.g. IMS Enterprise Services Specification). Regarding e-learning systems end users, the Higher Education sector is the most targeted.

Based on this survey, one can conclude that E-F and SIF to be the most promising e-learning frameworks since they are the most active projects, both with a large number of implementations worldwide. In the E-F, the contribution can be done by proposing new service genres, service expressions and service usage models. On SIF this type of contribution cannot be done to the concrete framework. However, new agents can be developed, such as those related with learning objects repositories.

3.2 Content

The concept of educational resource, course, student, summary or grade must be formally described in order to be shared among all the systems in an educational institution. For instance, the difficulty to reuse a course in schools with LMS coming from different vendors (or even from the same vendor) is an apt example of the problems found currently in the majority of the LMSs. These interoperability issues affect the flexibility of the teaching-learning process and lead to a decrease of end user satisfaction and learning success.

In order to overcome these shortcomings, practitioners of e-learning started valuing more the interchange of course content and learners' information, which led to the definition of standards for e-learning content sharing and interoperability. In this context, several organizations have developed specifications and standards in the last years. These specifications define, among many others, standards related to learning resources (commonly named learning objects) such as packaging them, describing their content, organizing them in modules and courses and communicating them among e-learning systems.

LO are context independent, transportable and reusable pieces of instruction that are digitally managed and delivered [MR03]. They are units of instructional content that can be used, and most of all reused, on web based e-learning systems. The LO definition was targeted for LMS and thus they are specialized on content presentation. They encapsulate a collection of interdependent files (HTML files, images, web scripts, style sheets) with a manifest containing metadata. This metadata is important for classifying and searching LO in digital repositories and for making effective use of their content in LMS. Standardized metadata plays an important role in keeping LO neutral to different vendors, both of LMS and of repositories.

In this study dozens of standards and specifications were found. For the sake of readability only the most prominent [LQ10b] are detailed organized in three facets: metadata, content aggregation and assessment.

3.2.1 Metadata

Metadata is data that is used to describe other data. IEEE defines metadata as information about an object, be it physical or digital. It is typically used to facilitate the management, discovery and retrieval of resources on the Web.

Metadata standards formalize the metadata structure promoting interoperability at two levels: support information exchange between machines and facilitate resource discovery by human users on the Web. Different communities have defined metadata standards that fit their needs. From a simple standard such as Dublin Core that is widely used in the digital library community to domain specific initiatives such as LOM (educational) or MPEG-7 (multimedia) that cover more elements within a domain.

In the remainder of this dissertation, the term metadata is used to represent Educational metadata since all research that is presented has been applied in the field of technology enhanced learning. Educational metadata standards extend the scope of the generic metadata with information that has particular educational relevance.

The following sections focus on the three established metadata standards most relevant to learning materials, the ISO 15836 Dublin Core (DC), the IEEE 1484.12.1-2002 Learning Object Metadata (LOM) and the ISO/IEC MLR.

3.2.1.1 Dublin Core

One of the earliest international metadata standards is the DC. DC metadata is a set of vocabulary terms (based on property-value pairs) which can be used to describe generic resources for the purpose of discovery. It was developed in 1995 by a group of librarians and content experts. It was called "Dublin Core" since it was created on a workshop in the city of Dublin (Ohio, United States). The Dublin Core Metadata Initiative (DCMI) is responsible for the development, standardization and promotion of the Dublin Core Metadata Elements Set (DCMES) that includes two levels: Simple and Qualified.

The **Dublin Core Simple**, which has been standardized as ISO Standard 15836-2003, includes fifteen elements (Title, Creator, Subject, Description, Publisher, Contributor, Date, Type, Format, Identifier, Source, Language, Relation, Coverage and Rights) that covers a broad range of resource types over the Web. A study [War04] reveals that over almost 1 million metadata instances from the Open Archives Initiative (OAI) that uses DC metadata

schema, found that of 15 DC data elements, five (creator, identifier, title, date, and type) are used 71% of the time and the five least used elements (language, format, relation, contributor and source) are used less than 6% of the time.

All these elements are optional and may be repeated if required. Despite the DC acceptance several issues raised related with the semantic of the element set and their coverage. The former is related to the need of define more specific semantics for some elements (e.g. distinguish between different dates such as date of submission and date of publication associated with the life-cycle of a resource). The latter reflects the absence of several characteristics of resources that are not covered, some of which are important in specialized domains (e.g. description of the intended audience of a resource) These issues led to the refinement and extension of the DC Simple element set resulting on the "qualified" Dublin Core metadata. The **Dublin Core Qualified** includes three additional elements (Audience, Provenance and Rights Holder), and a group of element qualifiers that refine the semantics of the elements.

In the last years, the DCMI needed to update DC metadata to facilitate extensibility and harmonize it with the principles of the semantic web with a set of specifications that comprises:

DCMI Abstract Model¹³ - specifies an abstract model for DC metadata. It defines the components used in DC metadata and describes how those components are combined to create information structures [BC10];

DCMI Metadata Terms - defines all the metadata terms maintained by the DCMI;

Guidelines for Dublin Core Application Profiles - provides guidelines for the creation of Dublin Core Application Profiles (DCAP). A DCAP is a set of documents that describes the metadata used in a particular application. The Singapore Framework¹⁴ for DCAP is the framework used for designing such metadata applications;

Guidelines for encoding DC metadata - defines several bindings of the DC metadata in RDF, XML and HTML/XHTML meta and link elements.

Currently, the DC metadata is widely used to describe resources mostly in the digital library domain. Several specifications use (or are based on) this standard: the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) uses it as an integral part of the specification defining the minimum metadata requirement for harvesting resources; the Open Source Metadata Framework (OMF) specification uses it to describe open source documentation; the Public Broadcasting Metadata Dictionary (PBCore) metadata standard extends the DC metadata for the public broadcasting community and the Search and Retrieve

¹⁴<http://dublincore.org/documents/singapore-framework/>

by URL (SRU) standard - a search protocol for repositories - uses it to formalize the search results. The CEN has recently standardized Metadata for Learning Opportunities based on the Dublin Core Abstract Model (RDF).

The DCMI is also actively engaged in covering the broad field of education with a new profile. These efforts are in sync with the new boost of technology in education and the increase number of educational resources stored in repositories spread over the Web. The DC-Education Application Profile Task Group is working on a Education Application Profile¹⁵ using the aforementioned Singapore Framework. Despite this promising initiative, there still not exist public implementations making the framework barely used. Also the required expertise and extra efforts in several areas inhibits potential clients.

3.2.1.2 IEEE LOM

The IEEE Standards Association published the IEEE 1484.12.1 – 2002 a standard for learning object metadata (LOM). The IEEE LOM is an international two-part standard for the description of "learning objects" and is composed of a conceptual data schema¹⁶ and an XML binding¹⁷ of that schema.

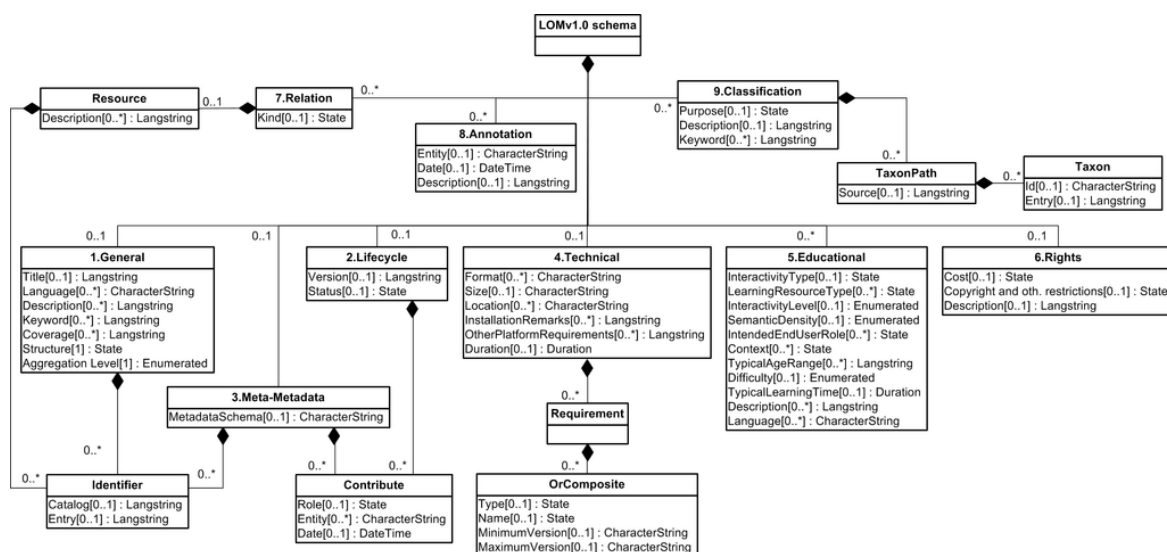


Figure 3.10: The hierarchy of elements in the LOM data model.

The LOM data schema specifies a set of characteristics for LO description, the vocabularies that may be used for these descriptions and defines how this data model can be refined by additions or constraints. The main purpose of LOM is to support the LO reusability, to aid discoverability, and to facilitate their interoperability, usually in the context of on-line

¹⁵http://dublincore.org/educationwiki/DC_2dEducation_20Application_20Profile

¹⁶1484.12.1—2002, Standard for Learning Object Metadata.

¹⁷1484.12.3—2005, Standard for XML Binding for Learning Object Metadata data model.

learning management systems. The LOM conceptual data schema consists of a hierarchy of elements as shown in Figure 3.10. The first level is composed of the following nine categories¹⁸.

General - describes the learning object as a whole. This category includes elements such as identifier, title, language, keywords;

Lifecycle - describes features related to the history and current state of the LO such as version, status, and contributors;

Metametadata - groups information about the metadata such as identifier, contributors and language used in the metadata;

Technical - describes the technical requirements and characteristics of the LO such as MIME type, size, required software/hardware;

Educational - describes educational and pedagogic characteristics of the LO such as interactivity type, learning resource type, interactivity level, semantic density, educational context, typical age range;

Rights - describes the intellectual property rights and conditions of use for the LO (whether or not any cost is involved, and whether copyright and other restrictions apply);

Relation - describes features that define the relationship between this LO and others ("based on", "part of", etc.);

Annotation - provides comments on the use of the LO and information on when and by whom the comments were created;

Classification - describes where the LO can be classified within a particular classification system.

Each category contains simple sub-elements with data or complex sub-elements aggregating other elements. The data schema also specifies the value space and data-type for each of the simple data elements. The value space defines the restrictions, if any, on the data that can be entered for that element (e.g. any string of Unicode characters, a declared list of a controlled vocabulary or a specified format such as date and language codes). The semantics of LOM elements are determined by their context: they are affected by the parent or container element in the hierarchy. For instance, the various **description** elements in the LOM data model derive their meaning from their parent elements.

These categories cover many facets of a LO. However, it is not possible for a generic standard such as IEEE LOM to fully meet the specific requirements and accommodate

¹⁸<http://www.ieeeltsc.org>

the particular needs of different educational communities. Fortunately, it was designed to be straightforward to extend. According to Al-Khalifa and Davis [AKD06], an important feature of LOM is that it is simple to use and has an inherent extension capability. Therefore, the practice of generating Application Profiles (AP) of the IEEE LOM has emerged¹⁹ and a number of different AP have been developed worldwide [CZS10]. The following list enumerates four profiling types that have been used to refine the LOM data model [Fri04a]:

Extension - defines extensions to LOM elements while preserving its set of categories;

Reduction - simplifies LOM, reducing the number of LOM elements and its choices;

Hybrid - extends and simultaneously reduces the number of LOM elements;

Combination - combines the LOM elements with elements from other specifications.

Table 3.2 presents a set of LOM profiles [SZC12] and the respective profiling types. It shows that all examined LOM profiles select a subset of data elements from the IEEE LOM Standard. None of them extend LOM elements while preserving its set of categories. On the other hand, none of the examined LOM APs define data elements from multiple namespaces. Additionally, only two of them (CELEBRATE and LRE) define new local data elements which are not included at the IEEE LOM Standard. The analysis of the profiling types of existing LOM profiles and their comparison demonstrates that the LOM element set is sufficient since it covers the most of the needs of the communities but larger enough that must be reduced. Another good example of a LOM-based metadata is the Canadian Core (CanCore) Metadata Application Profile. The CanCore standard is a LOM streamlined version that reduces the complexity and ambiguity of this specification.

Table 3.2: Profiling Types of existing LOM APs.

Types	ANZ-LOM	CELEBRATE	DET LRM	LRE	UK-LOM	VET
Extension	-	-	-	-	-	-
Reduction	X	X	X	X	X	X
Hybrid	-	X	-	X	-	-
Combination	-	-	-	-	-	-

The LOM has been widely implemented by repositories and other learning resource service brokers and providers (e.g. JORUM, ARIADNE, European SchoolNet, GLOBE), partly as a result of its status as an international standard, and partly through its association with other influential specifications, such as those produced by the IMS GLC (e.g. Content Packaging,

¹⁹The IEEE LOM Application Profiling Tool is a web-based tool for the creation of LOM application profiles (<http://asklomap.sourceforge.net/>).

Question and Test Interoperability and Vocabulary Definition and Exchange) and by ADL (e.g. SCORM).

A recent study [OKVD11] analysed the LOM use and quality of 630.317 metadata instances harvested from Global Learning Objects Brokered Exchange (GLOBE) repositories that provide a metadata harvesting service based on the OAI-PMH protocol. This study concludes interesting points regarding the frequency of LOM data elements and, more precisely, educational data elements. The study concludes that only a small fraction of the LOM standard is frequently used to describe the learning objects. Only 20 of the 50 data elements are used more than 60% of the time. Also, 16 data elements are used less than 10% of the time corroborating the Friesen study [Fri04b] conclusion that the added complexity of LOM is not used in the real world. The study also concludes that 4 out of 11 educational data elements (Learning Resource Type, Intended End User Role, Typical Age Range and Context) are used more than 40% of the time, 3 elements (Language, Interactivity Level and Interactivity Type) are used more than 10% but less than 20%, and the 4 remaining elements (Description, Difficulty, Semantic Density and Typical Learning Time) are used less than 10% of the time. While these values are lower than desirable, they provide proof that LOM is used in the real-world to capture educational information about digital objects.

Despite the generalized use of LOM and its intrinsic benefits, some issues exist. One such issue is that the LOM conceptual data schema is not based on an abstract model shared with other metadata schemata. This makes semantic interoperability with other metadata standards problematic [BC10]. Essentially it is impossible to import elements from other metadata schema (e.g. Dublin Core) hindering metadata modularization. Based on these limitations it is necessary for the IEEE LOM standard to accommodate general and non-educational characteristics (e.g. technical, rights, accessibility) within the standard data schema rather than importing solutions from other domains.

3.2.1.3 ISO/IEC MLR

Despite the wide use of both DC and LOM to describe learning resources, several semantic and interoperability issues are still not addressed as stated in the previous sub-subsections. These standards had similarities but they were not interoperable, and implementers of systems were left to solve the interoperability issues with workarounds. It was in this context, following the formation within ISO/IEC of SC36 (subcommittee 36, IT for Learning, Education and Training – ITLET), that a project²⁰ on Metadata for Learning Resources (MLR) was initiated.

The ISO MLR will be a "a harmonized standard" with LOM and DC due to the adoption

²⁰ISO (2011). ISO/IEC 19788-1:2011 Information technology – Learning, education and training – Metadata for learning resources – Part 1: Framework. International Organization for Standardization

of LOM as a result of the market success of SCORM [HM11]. This initiative proposes the adoption of a semantic model that will maximize ISO MLR's compatibility with current efforts in DC and LOM.

The ISO MLR will be a multi-part standard composed of six parts: 1) the framework; 2) data elements; 3) the core application profile; 4) technical elements; 5) education elements; 6) availability and rights management. However, other parts may be defined in the future. Part 3 promotes interoperability among repositories. It is expected that user communities will enhance this application profile by adding data elements from others standards (LOM and DC) and by adding vocabulary constraints.

3.2.1.4 Other Metadata Specifications

The Attention Profiling Mark-up Language (APML) enables the description of topics and sources that a person is interested in to be shared in the form of an XML file. The attention profile may be generated explicitly by the person concerned or may be derived from attention data, i.e. information about what a person has been looking at derived from a record of their activities. This specification is being developed by a community with no direct affiliation to any formal specification or standards body. Notwithstanding the specification's draft status several prototype services have implemented it including digg.com and the BBC's radiopop.co.uk.

CEN has endorsed a workshop agreement and a commitment to develop a European Norm for Metadata for Learning Opportunities. This work has its origins in course description metadata initiatives from several European countries, and describes a common model for learning opportunities so that they may be aggregated by other services. The initial focus is on course advertising; however there is scope for wider application to course description for other purposes.

There is a long history of work on standardizing competency definitions and the like, including the IEEE Reusable Competency Definition (IEEE, 2007) and HR-XML (HR-XML consortium, no date). Currently working group 3 of ISO subcommittee 36 (ISO/IEC JTC1 SC36 WG3) is developing a conceptual reference model for competences and related objects. Again, there is potential scope to apply this model to the educational outcomes object in the DC-Education domain model.

Other standards for metadata such as METS, MODS, PREMIS and MIX are mostly related to digital libraries. The most prominent are the first two. The Metadata Encoding and Transmission Standard (METS) is a XML standard for describing metadata regarding objects within a digital library [Dig07]. The standard is supported by the Network Development and MARC Standards Office of the Library of Congress. The Metadata Object Descrip-

tion Schema (MODS) is an XML-based bibliographic description schema developed by the United States Library of Congress' Network Development and Standards Office. MODS was designed as a compromise between the complexity of the MARC format used by libraries and the simplicity of DC metadata.

3.2.2 Packaging

The need for educational resources sharing among learning systems and authoring tools motivated the development of common formats to encapsulate learning resources into units of instruction. These formats apply structure and learning taxonomies so that the structure of the units of instruction and their behaviour (sequencing of activities) can be uniformly represented, interchanged and reproduced across heterogeneous environments.

Content packaging formats should be neutral and allow the encapsulation of separate resources ranging from a single educational resource to entire courses. At the same time can be complemented with definitions of how content is presented to the learner and the conditions under which a piece of content is selected, delivered, or skipped during presentation. This subsection details the most important specifications used to package content such as IMS CP, SCORM and IMS CC.

3.2.2.1 IMS Content Packaging

Packaging the learning resources complements content description and is crucial to facilitate the deployment, storage and reuse of learning resources. One of the earliest efforts was from the Aviation Industry Computer-Based Training Committee (AICC). The AICC association developed in 1998 a content package format called AICC HACP consisting of four comma separated ASCII files that define details about the learning content including a URL. In 2000 the IMS Global launched the IMS Content Packaging (IMS CP). An IMS CP learning object (Figure 3.11) assembles resources and meta-data into a distribution medium, typically an archive in ZIP format, with its content described in a manifest file at the root level. The manifest file - named `imsmanifest.xml` - adheres to the IMS CP schema and contains the following sections:

Metadata - describes the package as a whole;

Organizations - describes the organization of the content within a manifest;

Resources - contains references to resources (files) needed for the manifest and metadata describing these resources;

Sub-manifests - defines sub packages.

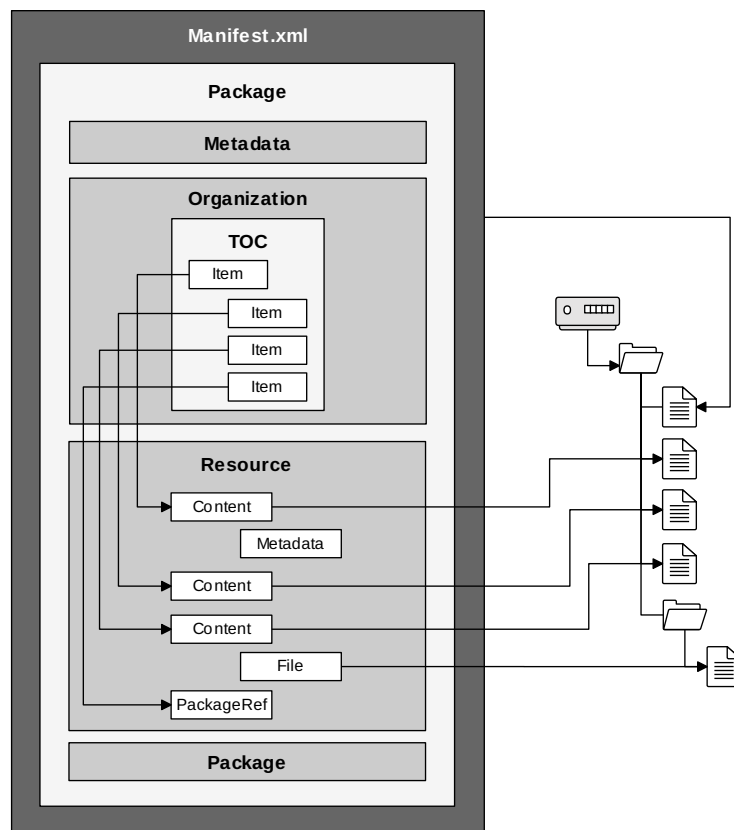


Figure 3.11: IMS CP package structure.

Meta-data information in the manifest file usually follows the IEEE LOM schema, although other schemata can be used. These meta-data elements can be inserted in the metadata section of the manifest to describe the learning object as a whole or can be included in the resources section to describe each resource of the package. The IMS CP specification includes a manifest section called Organizations to design pedagogical activities and articulate the sequencing of instructions. By default, it uses a tree-based organization of learning items pointing to the resources (assets) included in the package. However, other standards could be accommodated in this section, such as IMS Learning Design (IMS LD) and IMS Simple Sequencing (IMS SS). The IMS LD specification is a meta-language for describing pedagogical models and educational goals. Several IMS LD-aware tools are available as players (e.g. CopperCore, .LRN) and authoring/export tools (e.g. Reload, LAMS). The IMS SS is a specification used to describe paths through a collection of learning activities. The specification declares the order in which learning activities are to be presented to the learner and the conditions under which a resource is delivered during an e-learning instruction. Despite all these specifications, the design of more complex adaptive behaviour is still hard to achieve [ADjH⁺06].

3.2.2.2 ADL SCORM

Other well-known package format is the Sharable Content Object Reference Model (SCORM). SCORM was created by the Advanced Distributed Learning initiative (ADL) with the first production version launched in 2001. It is an application profile for content packaging that extends the IMS CP specification with more sophisticated sequencing and Contents-to-LMS communication. It defines communications between client side content and a host system called the run-time environment, which is commonly supported by learning management systems. The most recent version of SCORM specification is SCORM 2004 (4th edition) from 2009.

3.2.2.3 IMS Common Cartridge

In 2008, IMS GLC proposed the IMS Common Cartridge (IMS CC) . Common Cartridge was developed primarily to support the use of digital course materials and digital books in the instructional context. It was not designed as a replacement for SCORM. The specification defines an open format for the distribution of rich web-based content. Its main goal is to organize and distribute digital learning content and to ensure the interchange of content across any Common Cartridge conformant tools. The latest revised version (1.2) was released in October 2011²¹. The IMS CC package organizes and describes a learning object based on two levels of interoperability: content and communication as depicted Figure 3.12.

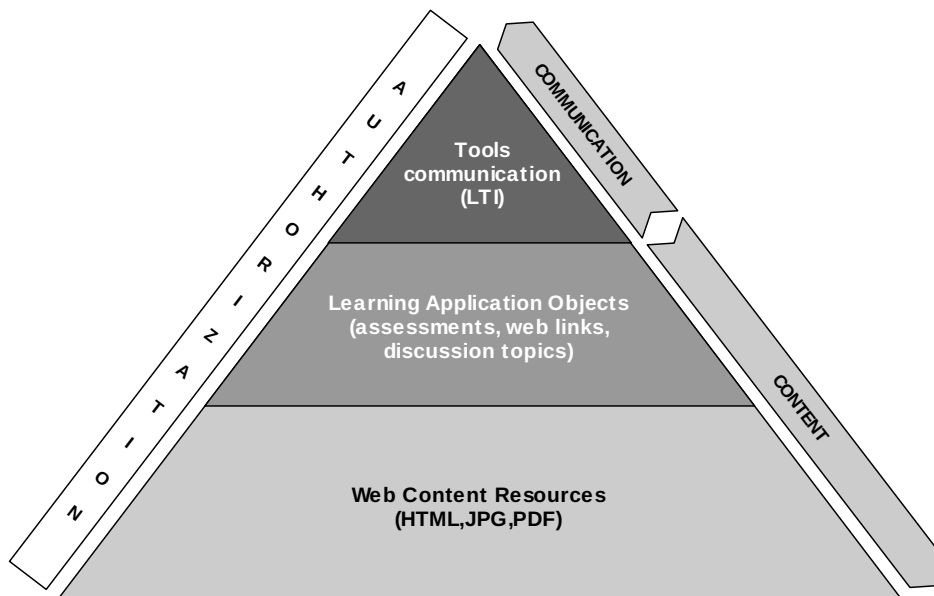


Figure 3.12: Common Cartridge Content Hierarchy.

²¹IMS CC v. 1.2 Specification: <http://www.imsglobal.org/cc/ccv1p2/imscv1p2-profilev1p2-Overview.html>

In the **content level**, the IMS CC includes two types of resources:

Web Content Resources (WCR) - static web resources that are supported on the Web such as HTML files, GIF/JPEG images, PDF documents, etc.

Learning Application Objects (LAO) - special resource types that require additional processing before they can be imported and represented within the target system. Physically, a LAO consists of a directory in the content package containing a descriptor file and optionally additional files used exclusively by that LAO. Examples of Learning Application Objects include QTI assessments, Discussion Forums, Web links, Basic LTI descriptors, etc.

In the **communication level** the cartridge describes how the target tool of the cartridge (usually a LMS) should communicate with other remote web applications using the IMS Basic LTI specification. Both levels enhance the interoperability of the cartridge among a network of e-learning systems. In this scope a new IMS CC specification feature is introduced to support authorization at two levels: either the whole cartridge can be protected or individual resources can be protected. The following subsections detail the most important elements of the CC content hierarchy.

The Common Cartridge builds upon a profile of the CP package. Figure 3.13 provides a view of the CC manifest. The manifest is composed by four sections: metadata, organizations, resources and authorizations. The Metadata section is used to store the cartridge metadata restricted to a loose binding of LOM elements based on the DC specification. The Organization section is used to represent the Common Cartridge Folder content type as a structural approach to organize content. The Resources section is used to refer assets included in the cartridge.

The manifest includes the LOM metadata standard to describe the cartridge and the learning resources could be included in the cartridge package. The metadata could be include at two levels: cartridge and resources. At the cartridge level one must use metadata according to the Common Cartridge profile of the IEEE LOM (loose binding) which describes the range of a mapping from the core elements of the Dublin Core specification v1.1 to IEEE LOM. At the resources level one could use the original IEEE LOM namespace. There will be scenarios where resources may need to be specified within the organization, but should not be made visible in player mode upon default import of the cartridge. One such situation is the inclusion of a solution program within the cartridge. To meet these needs, the common cartridge supports the optional “roles” meta-data associated with the resource in the manifest file. The supported roles in the IMS CC version 1.1 are: Learner, Instructor and Mentor. If case of absence of the role the resource would be viewable by all users.

An IMS CC learning object supports **authorization** at three levels: on cartridge import,

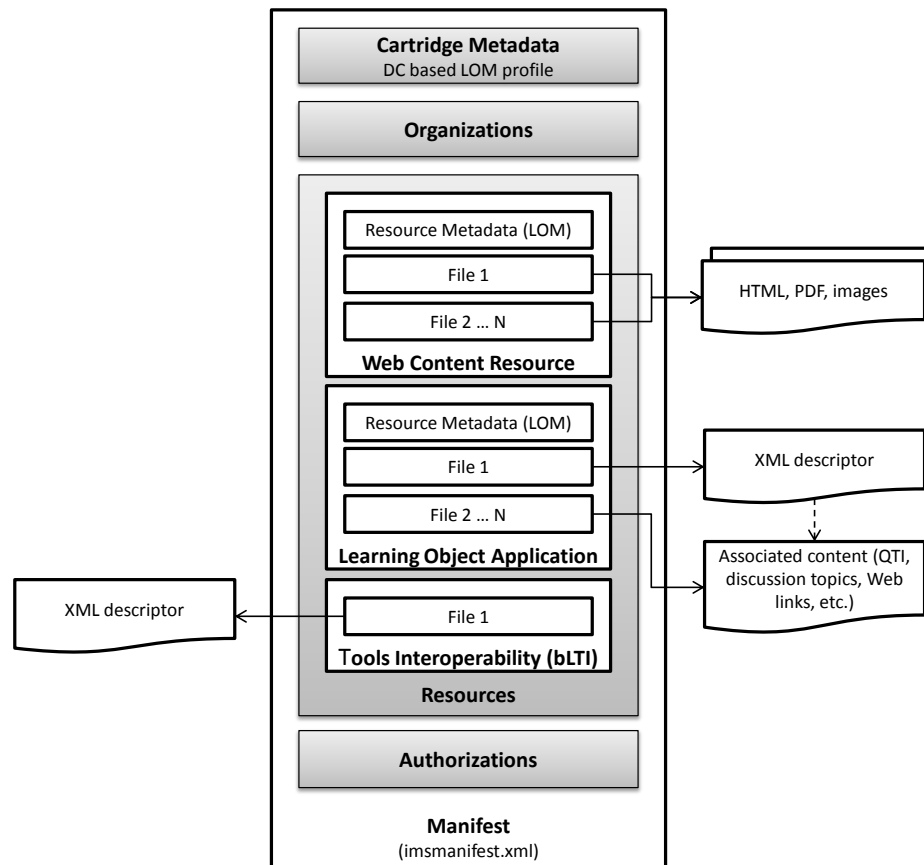


Figure 3.13: IMS Common Cartridge package.

on cartridge usage and on usage of specific resources in the cartridge. The mechanism by which the authorized access to particular resources is enforced by the tool is not defined by the profile.

The Common Cartridge uses the IMS QTI specification as a data model for **questions and tests**. This specification is represented on the manifest through two LAO resource types: assessments and question banks. An assessment represents an ordered question set (e.g. Multiple Choice, True/False, Fill in the Blanks, Pattern Match, and Essay) and may include optional attributes (e.g. number of attempts, time limit and whether late submission is allowed) that apply to the set as a whole. A question bank can embed any of the question types supported by the CC v1.1 profile of QTI. Only one question bank can optionally be included in a cartridge.

A **Web Link** is a LAO resource that extends a standard HTTP link. The extension comprises a title and a URL describing a set of window open features such as the dimensions of the window. This approach allows the cartridge to minimize its storage space and to have content updates after distribution.

A **Discussion Topic** is a LAO resource used to initiate a discussion activity. Upon import, the discussion topic content is stored by the tool using its own internal representation. As the cartridge content is added to an actual course, an associated discussion topic is created in the default tool discussion forum. The Discussion topic schema supports the use of plain text or HTML for the discussion content and allows the attachment of other resources through the use of the attachment element.

A **Basic LTI resource** refers to an XML file that contains the information needed to create a link in a Tool Consumer (TC) (e.g. LMS). Upon the user's click, the execution flow passes to a Tool Provider (TP) along with contextual information about the user and Consumer. The Basic LTI link is defined in the resource section of an IMS Common Cartridge. The `href` attribute in the resource entry refers to a file path in the cartridge that contains an XML description of the Basic LTI link. A BLTI link contains several elements. The most important are: the `title` and `description` elements contain generic information about the link; the `custom` and `extensions` elements allow the Tool Consumer to extend the basic communication data; the `launch_url` element contains the URL to which the LTI invocation is sent; the `secure_launch_url` element is the URL to use if secure HTTP is required. The LTI message signing is performed by a security mechanism designed to protect POST and GET requests called OAuth. OAuth 1.0 specifies how to construct a base message string and then sign that string using a secret. The signature is then sent as part of the POST request and is validated by the Tool Provider using OAuth. Upon receipt of the POST, the TP will perform the OAuth validation using the shared secret.

In a recent study [KC] the CC and SCORM specifications were compared regarding interoperability from the point of view of key users: the teachers. Teachers showed special interest towards CC packages and in particular their use in Moodle system. In detail, teachers enjoyed the possibility of editing a package, taking some elements and mixing them with their own teaching resources very much in the same way as they do in with the non-digital resources in their classrooms.

3.2.2.4 Other specifications

The large number of package and metadata standards means that there may be several versions of a learning object (e.g. an English version and a French version) and available in different formats (e.g. IMS Common Cartridge or as an SCORM 2004 package) from several heterogeneous repositories. In order to overcome the potential data exchange issues, Massart created the Information for Learning Object eXchange (ILOX) framework [Mas10]. This framework organizes multiple metadata specifications in a container that can be handled as a whole. It was developed as part of the IMS Learning Object Discovery & Exchange (IMS LOD) specification that aims to facilitate the discovery and retrieval of learning objects

stored across more than one collection and over a federation of repositories.

3.2.3 Assessment

Assessment specifications address the need of defining common formats and procedures for the exchange of evaluation material among different e-learning tools. Two types of assessment specifications are used: quiz based assessments and text file based assessments.

3.2.3.1 Quiz based assessments

The majority of e-learning systems include the automatic evaluation of quizzes as a feature. Quizzes have the advantage of being generic and usable in any learning domain. The most prominent initiative on the quiz-based evaluation domain is the QTI. The QTI specification defines a standard format for the representation of assessment content and results and their exchange between authoring and delivery systems (e.g. LMS). The specification consists of a data model that defines the structure of the questions (with a set of pre-defined answers, such as multiple choice, multiple response, fill-in-the-blanks and short text questions), the tests, feedback and results; and an XML data binding that defines a language to facilitate the exchange of the materials.

Despite the completeness of the specification and its acceptance by the industry as the de facto standard, the QTI interoperability is still not sufficient [GRACG⁺09]. The most relevant reasons are presented as follows:

Complexity of the specification - if a tool or platform uses a data model with a significant differences from the one proposed in the specification, its capability to interoperate with other platforms is hindered. The QTI Lite specification appeared as an attempt to bridge this complexity gap;

Version instability - the current versions (2.1) is incomplete. The IMS QTI project group is in the process of evolving this specification based on input from market participants. In the meantime, most LMSs and other assessment tools have solved the interoperability problem with ad-hoc solutions that will have a large inertia to adopt future versions of the specification;

LMS support - heterogeneous support from LMS.

The last issue can be further explored. Table 3.3 summarizes the compliance of different e-learning systems with the different versions of QTI. Support for version 1.2 is considerably higher than for the most recent version 2.1.

Table 3.3: QTI compliance.

E-Learning Tools	Import	Export
Angel	-	2.1
ATutor	1.2	1.2, 2.1 (*)
Clix	1.2	1.2
DB Primary	-	2.0
Diploma	1.2, Lite	1.2, 2.1, Lite
Dokeos	1.2	1.2, 2.0
.LRN	1.2	1.2
Moodle	1.2	2.0
OLAT	1.2	1.2
QTITools	2.1	2.1
QuestionMark	1.2	1.2
Respondus	-	1.2
Sakai	1.2	1.2

This heterogeneous support comes from the lack of tools supporting the last version. The lack of use reduces the interest of the format. In addition, version instability is an aggravating factor. An important percentage of the tools supporting QTI 1.2 do not risk to step forward to the following version, as it is not clear if it is going to be endorsed by the IMS GLC consortium in a near future.

There are other formats to describe questions and tests but with less importance. This is the case of GIFT ²² a internet format used by Moodle, HotPatatoes, Aiken, Learnwise and others.

3.2.3.2 Text file based assessments

The most effective types of exercises in any learning domain, both for knowledge acquisition and for student grading, are seldom quizzes. Text file automatic evaluation differs significantly from quiz evaluation based on QTI. In fact, the evaluation of text files requires extra resources and specialized metadata. For this reason QTI [LQ09e] is not adequate for text file automatic evaluation, as would be expected since it was not designed for this purpose. Extensions to learning object specification have to be developed to support text file evaluation. On one hand text file evaluation is too specialized to justify its integration in a general LMS. On the other hand, provided as a service it can be used by many kinds of systems.

²²<http://microformats.org/wiki/gift>

One good example is computer programming assessment. It is hard to imagine learning computer programming without actually programming. An attempt to solve a programming exercise is written in a specific language (a programming language) that cannot be evaluated simply by comparing it with predefined answers, as in quiz evaluation. The data model of QTI was designed for questions with a set of pre-defined answers and cannot handle evaluation domains with specialized requirements, such as programming exercise evaluation. For instance, programming exercises evaluations requires tests cases, program solutions, compilation lines and other specific type of meta-data that cannot be encoded in QTI. QTI supports also long text answers but the specification of their evaluation is outside of its scope. Although long text answers could be used to write the program's source code, there is no way to specify how it should be compiled and executed, which test data should be used and how it should be graded. For these reasons QTI is not adequate for automatic evaluation of programming exercises, although it may be supported for sake of compatibility with some LMS.

The increasing popularity of programming contests worldwide resulted in the creation of several contest management systems. At the sametime Computer Science courses use programming exercises to encourage the practice on programming. The interoperability between these type of systems is becoming a topic of interest in the scientific community. In order to address these interoperability issues several problem formats were developed. The following paragraphs detail four formats: CATS, FreeProblemSet (FPS), Mooshak Exchange Format (MEF) and Peach Exchange Format (PEF). Then, their features are synthesized based on a specific exercises format expressiveness model.

CATS²³ is a format for programming assignments [Kle11]. The format encoded in XML describes the conditions of the problem and a set of files with additional tests, test programs, etc. All these files are wrapped up in a ZIP file to facilitate deployment. A typical XML file contains the description of 1) the problem statement codified in the Simple Text Markup Language (STML) format - a simplified subset of HTML; 2) the format of input and output files and samples; 3) the restrictions on the input format; 4) references for external resources such as tests generators, special checkers, plug-ins and solution programs.

Freeproblemset (FPS)²⁴ is a transport file format for the storage of all information about problems. It aims to provide free problems sets for managers of ACM/ICPC Online Judges by transporting data from one judge to another. The format uses XML to formalize the description of a programming problem. It includes information regarding the problem, the test data, the special judger(optional) and the answer(optional). Currently, the FPS format is supported by several online judge systems including HUSTOJ²⁵, ACM-Server²⁶ and Woj-

²³<http://imcs.dvgu.ru/cats/docs/format.html>

²⁴<http://code.google.com/p/freeproblemset/>

²⁵<http://code.google.com/p/hustoj/>

²⁶<http://code.google.com/p/acm-server/>

land (OJ from Wuhan University)²⁷.

Mooshak²⁸ is a web based competitive learning system originally developed for managing programming contests over the Internet [LS03]. Recently, it was upgraded to expose the evaluation functions as services. These services are expected to integrate in heterogeneous networks of e-Learning types of systems, including the Learning Management Systems (LMS), Integrated Development Environments (IDE) and Learning Objects Repositories (LOR) [LQ09a]. Despite the context where it is used, Mooshak has its own internal format to describe problems called Mooshak Exchange Format (MEF). MEF includes an XML manifest file referring several type of resources such as 1) problem statements (e.g. PDF, HTML); 2) image files; 3) input/output test files; 4) correctors (static and dynamic); and 5) solution programs. The manifest also allows the inclusion of feedback and points associated to each test.

Currently, Mooshak is being used in several Universities worldwide to support learning activity. In the competitive context, it is one of the most used evaluation systems for the IEEE programming contests.

Peach²⁹ is a system for the presentation of assignments, the collection, storage, and automated and/or manual evaluation of work submitted for assignments, and the administration of results. The Peach Exchange Format (PEF) [Ver08] is a specific format for programming task packages used in Peach. Peach task packages are stored in a directory tree with a predefined structure. There are separate subdirectories for: 1) texts subdivided by natural language (task, background information, hints, grading information); 2) metadata (task, author, event, solver, grading and management, etc.); 3) solutions subdivided by programming languages; 4) evaluation data subdivided in cases; and 5) tools (input generator, input validator, output format checker). Currently, Peach is being used by the Eindhoven University of Technology³⁰.

Several approaches can be found in literature [Ver08], [Kle11], [EBC⁺08] to evaluate the expressiveness of programming assignments formats. This sub-section synthesizes the formats described previously according to the model proposed by Verhoeff. The choice of the Verhoeff model over the alternatives is due to its more comprehensive coverage of the required features. This model organizes the programming exercise data in five facets:

Textual information - programming task human-readable texts;

Data files - source files and test data;

Configuration and recommendation parameters - resource limits;

²⁷<http://code.google.com/p/woj-land/>

²⁸<http://mooshak.dcc.fc.up.pt/>

²⁹<http://peach3.nl>

³⁰<http://peach.win.tue.nl/>

Tools - generic and task-specific tools

Metadata - data to foster the exercises discovery among systems.

The **textual information facet** (Table 3.4) accommodates all the information (or pointers to such information) that the exercise author wants to offer to students or contestants about the exercise. It includes, for instance, the exercise challenge, background information, grading information and input/output samples. These texts may be available in several languages and formatted in plain text or other open format standards such as HTML or L^AT_EX. Since this data is encode in flexible text data formats in can be transformed into various (open) presentation formats such as PDF. Other texts (e.g. grading information and samples) should ideally be generated from the evaluation data.

Table 3.4: Textual information facet.

Feature	CATS	FPS	MEF	PEF
Multilingual	-	-	X	X
HTML format	X	X	X	X
L ^A T _E Xformat	-	-	X	-
Image	X	X	X	X
Attach files	X	-	-	X
Description	X	X	X	X
Grading	-	-	-	-
Samples	-	-	-	-

Besides human-readable texts, a programming exercise can also include several other files, in both text or binary format.

Table 3.5: Data files facet.

Feature	CATS	FPS	MEF	PEF
Solution	X	X	X	X
Skeleton	-	-	-	-
Multi-language	X	X	-	X
Tests	X	X	X	X
Test groups	X	-	-	X
Sample tests	-	X	-	-
Grading	X	-	X	X
Feedback	-	-	X	-

The **data files facet** (Table 3.5) covers the following files: program and skeleton source files (ideally with support for multiple programming languages), input/output test data (ideally

with support for grouping and multiple visibility mode), feedback files associated with a specific test case and others files.

The description of a programming exercise can also include parameters related with the submission, compilation and execution of the students' attempts to solve the exercise. These parameters can be organized in terms of **configuration and recommendation** (Table 3.6). The former deals with the configuration of compiler and linkers such as the compilation line of source files and associated parameters. The latter includes recommendations usually expressed in terms of resource limits such as the size and number of lines of a submission, compilation and execution time and memory limits. These recommendations depend on the actual platform used for evaluation runs. Thus, platform information should be associated for a more accuracy control.

Table 3.6: Configuration and recommendation parameters facet.

Feature	CATS	FPS	MEF	PEF
Compiler	-	-	-	-
Executer	-	-	-	-
Memory limit	X	X	-	X
Size limit	-	-	-	-
Time limit	X	X	-	-
Code lines	-	-	-	-

When creating, solving or evaluating an exercise several software tools are needed such as editors, compilers, libraries, linkers, test and feedback generators, input/output format checkers, evaluators, etc. The **tools facet** (Table 3.7) covers the support of the exercise formats either by referencing these tools or by formalizing data that can be used as input for these tools.

Table 3.7: Tools facet.

Feature	CATS	FPS	MEF	PEF
Compiler	-	-	-	X
Test gen.	X	-	-	-
Feedback gen.	-	-	X	-
Skeleton gen.	-	-	-	-
Checker	X	-	-	X
Corrector	-	-	X	-
Library	X	-	X	X

The **metadata facet** (Table 3.8) comprises all the data that provide useful information on the exercise for classification and discovery purposes. There are several types of metadata

that can be included in a programming exercise such as: exercise metadata (exercise type, keywords, difficulty level), authors metadata (name, contact), event metadata (name, type, local, date, number of participants, etc.), solver metadata (platform, operating system, etc.) and management metadata (status of development, version information, etc.).

Table 3.8: Metadata facet.

Feature	CATS	FPS	MEF	PEF
exercise	X	X	X	X
author	X	-	-	X
event	-	X	-	X
keywords	-	-	X	X
license	-	-	-	-
platform	-	-	-	X
management	-	-	-	X

This study confirms the disparity of programming exercise formats highlighting both their differences and their similar features. This heterogeneity hinders the interoperability among the typical systems found on the automatic evaluation of exercises. Rather than attempting to harmonize the various specifications, a pragmatic solution is to provide a service for exercises formats conversion.

3.3 Communication

The share and reuse of learning objects depends not only of the adoption of common formats to describe the content but also of standard mechanisms to share it to different vendors, both of LMS and of repositories. In this section the standards and specifications regarding these two types of e-learning systems are detailed.

3.3.1 Learning Management Systems

An LMS plays an important role in any e-learning environment. Still, the LMS cannot afford to be isolated from other systems in an educational institution. Thus, the potential for interoperability is an important, although frequently overlooked, aspect of an LMS system. The following sub-subsections present the most common strategies for integrating an LMS with other systems usually found in educational institutions namely the Data, the API and the Tool integration strategies. An integration example between a LMS and one of these systems is also presented. Although the focus is the LMS other types of systems can achieve these types of integration.

3.3.1.1 Data Integration

Data integration is the simplest and most popular form of integration in content management. This type of integration uses the import/export features of both systems and relies on the support of common formats as shown in Figure 3.14. For instance, an e-Portfolio system may import data (blog and forum contributions by students, course materials and assignments uploaded by teachers) from LMS to avoid the burden of entering this data manually.

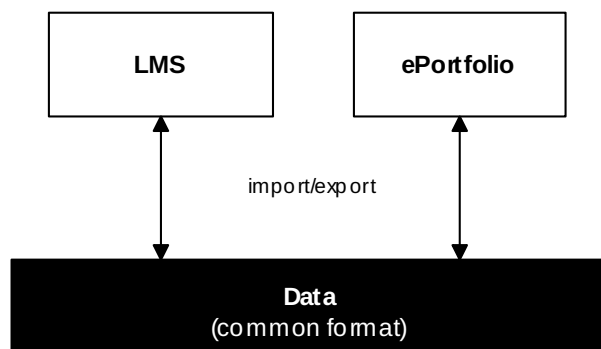


Figure 3.14: Data Integration.

These systems support two types of common formats: generic (e.g. HTML files) and e-Portfolio specific (e.g. Leap2A files). The former are useful since they are widely available, but they lack domain specific semantic data provided by the latter. For instance, if someone adds a post in a Moodle forum it should be included in the Mahara e-Portfolio as a blog post and not as a non-editable HTML artifact. This requires the use of a common e-Portfolio standard so that Mahara (or any other e-Portfolio system) can understand the meaning of the content and decides its final format.

3.3.1.2 API Integration

An API allows client applications to use directly the functions of an e-learning system. These API foster client application development through data encapsulation and behavioural reuse. This clear separation of interfaces specification from their implementation and data formats allows tool vendors to develop new versions without affecting current clients. The major LMS vendors include API to allow developers to extend their predefined features through the creation of plug-ins. Blackboard uses the Building Blocks technology to cover the integration issues with other systems allowing third parties to develop modules using the Building Blocks API. The new Moodle version (v.2.0 released in November 2010) includes several API (Figure 3.15) to enable the development of plug-ins by third parties to access repositories and portfolios such as the Repository API for browsing and retrieving files from external repositories; and the Portfolio API for exporting Moodle content to external

repositories. These two API are based on the File API - a set of core interfaces to allow Moodle to manage access control, store and retrieve files. The new File API aims to enhance file handling by avoiding redundant storage. This feature is achieved since every file in Moodle 2.0 is saved into a file pool (a directory in moodledata) with a file name that is calculated as a SHA1 hash of the file content. If a file is copied (e.g. course cloning) no file duplication happens, just a new record in a special table of files is created.

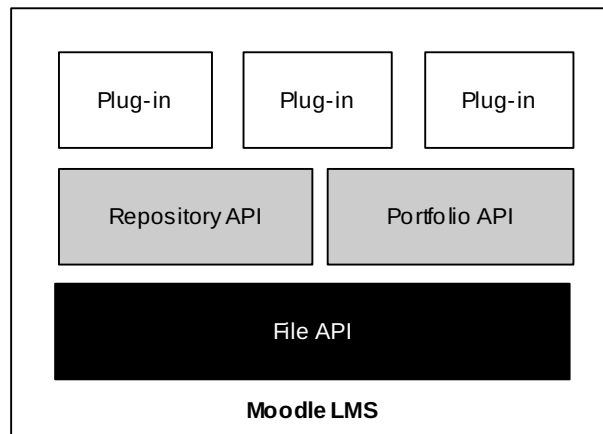


Figure 3.15: API Integration.

In order to ensure a bidirectional communication between a LMS and an e-Portfolio system it is required to use both API to create plug-ins. For instance, in the Moodle LMS, the Mahara support is guaranteed only in one way by the implementation of the Portfolio API. The Portfolio API is a core set of interfaces to publish files from Moodle to external repository systems, mainly e-Portfolio systems. In this approach, the e-Portfolio system appears seamlessly as a folder when students want to save content such as a file, snapshots of forums or blogs and assignments. At time of writing this paper, Moodle 2.0.1 (January 2011) includes in its release package several plug-ins for e-Portfolios such as Mahara, Flickr, Google Docs, Boxnet and supporting different formats such as Leap2A and HTML. Regarding the Repository API the same release package includes support for the repositories Alfresco, Boxnet, Dropbox, Flickr, Google Docs, Merlot and Picasa.

3.3.1.3 Tool Integration

While e-learning frameworks are general approaches for e-learning system integration, several authors proposed service oriented approaches specifically targeted to the LMS. In fact, there are several references in the literature to middleware components for LMSs integration in SOA based e-learning systems. Apostolopoulos [AK03] proposes a middleware component to address the lack of integration of e-learning services. In this approach the e-learning

components are implemented as agents maintained in a local management information base, and can communicate with the agent manager through the SNMP protocol. Costagliola [CCF⁺07] develop an architecture based on a middleware component and use Web Services to integrate different software components and improve interoperability among different systems. The middleware component enables the student learning process traceability since it has been developed to be compliant with SCORM. Al-Smadi [AS10] presents a service-oriented architecture for a generic and flexible assessment system with cross-domain use cases. All these approaches have in common the need of a modification of LMS for each specific vendor, with the implementation of a new module or building block. To the best of the authors' knowledge there are no references in the literature to the use of a common standards supported by the major LMS vendors as a means to integrate the LMS in a service oriented network of learning environments.

A common interoperability standard that is increasingly supported by major LMS vendors is the IMS LTI specification. The IMS LTI provides a uniform standards-based extension point in LMS allowing remote tools and content to be integrated into LMSs. The main goal of the LTI is to standardize the process for building links between learning tools and the LMS. There are several benefits from using this approach: educational institutions, LMS vendors and tool providers by adhering to a clearly defined interface between the LMS and the tool, will decrease costs, increase options for students and instructors when selecting learning applications and also potentiate the use of software as a service (SaaS). The LTI has 3 key concepts as shown in Figure 3.16 [Gil10]: the Tool Provider, the Tool Consumer and the Tool Profile.

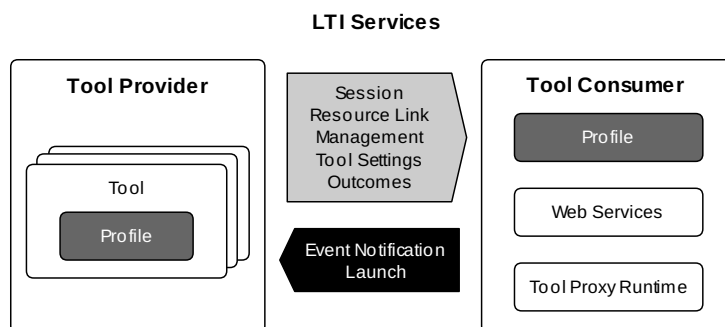


Figure 3.16: IMS Full LTI.

The tool provider is a learning application that runs in a container separate from the LMS. It publishes one or more tools through tool profiles. A tool profile is an XML document describing how a tool integrates with a tool consumer. It contains tool metadata, vendor information, resource and event handlers and menu links. The tool consumer publishes a Tool Consumer Profile (XML descriptor of the Tool Consumer's supported LTI functionality that is read by the Tool Provider during deployment), provides a Tool Proxy Runtime and

exposes the LTI services.

A subset of the full LTI v1.0 specification called IMS Basic LTI exposes a single (but limited) connection between the LMS and the tool provider. For instance, there is no provision for accessing run-time services in the LMS and only one security policy (OAuth protocol³¹) is supported. For instance, to export content from Moodle to Mahara using the Basic LTI the teacher (or LMS administrator) must first configure the tool (Mahara) as a Basic LTI tool in the course structure. When a student selects this tool, Moodle launches a Mahara session for the student. The web interface for this session can either be embed in Moodle's web interface as an iframe or launched in a new browser window.

Recently, IMS launch the Learning Tools Interoperability v1.1 Public Draft that combines Basic LTI and LTI into just Learning Tools Interoperability. Version 1.1 of LTI includes updates and clarifications as well as support for an outcomes service.

3.3.1.4 Comparison of the integration strategies

This subsection presents a comparative study on the e-Portfolio integration strategies in LMS. This study is summarized in Table 3.9 and can be used as a guide in the selection of an integration strategy.

Table 3.9: Comparison of e-Portfolio integration strategies

	Data	API	Tool Integration	
	Integration	Integration	bLTI	fLTI
Technical skills	No	Yes	Yes	Yes
Degree of coupling	No bounding	Tightly	Loosely	Tightly
Security	To implement	To implement	OAuth	OAuth
Batch integration	No	Yes	No	No
Development effort	-	Some	Little	Great
Communication type	Bidirectional	Bidirectional	Uni	Bi
Status (# implementations)	-	Many	Many	Few

Data integration is the best option when the development effort must be kept to a minimum or no one with technical skills (specially programming skills) is available, since the other two strategies require them. This strategy has also the advantage of not coupling the two systems and enabling a bi-directional communication.

API integration is best suited when batch integration is required since the other two strategies involve the use of the GUI of both systems. For instance, if the work of the students of a

³¹OAuth security protocol: <http://oauth.net/>

given set of courses must be copied on a regular basis from the LMS to their portfolios then the API strategies are recommended. The major drawbacks of this approach are the amount of development required and the tight coupling between the LMS and the e-Portfolio system, since special plug-ins must be implemented and API are vendor specific. Finally, this strategy enables bidirectional communication, although the current version of Moodle (2.0.1) does not implement yet the API repository, thus rendering in practice the communication between LMS and Mahara unidirectional.

Tool integration is arguably the best choice in general since it provides a good balance between implementation effort and coupling and security. This is especially true if only unidirectional communication is required and Basic LTI is used. This tool integration flavour is simple to implement and is already supported by most LMS vendors. If bidirectional communication is required then full LTI is needed but in this case the implementation is harder and few LMS vendors support this flavor of the specification. In both cases, tool integration has the added value of providing some basic security features based on the OAuth protocol aiming to secure the message interactions between the Tool Consumer and the Tool Provider.

This comparative study was based both on the available documentation and on the authors experience in using the different strategies to integrate Moodle with other systems, in particular the development of a Moodle plugin using the Repository API [LQ10c] and the basic LTI runtime to link Moodle with other e-learning systems [LQ11e].

3.3.2 Repositories

Repositories are crucial in the e-learning domain and are increasing in number [RSS10]. This growth led many to neglect interoperability issues that are fundamental to share educational resources and to (re)use them on different contexts. Beyond the standardization of content, repositories need to interact with other systems that typically cohabit in the e-learning realm. Examples of these systems are authoring tools, learning management systems, harvesting systems, intelligent tutors, and evaluation engines. In fact, some surveys [RSS10, Fay10, Tea06, Tzi09] concluded that user expectations regarding standardization, content management and interoperability are not completely met by existing repositories. One of the main issues was that current repositories are specialized search engines of LO and not adequate for feeding specialized services. The share and reuse of learning objects depends not only of the adoption of common formats to describe the content but also of standard mechanisms to publish data to repositories but also to search and retrieve data from repositories. The basic functionalities provided by a repository boil down to data push (publication of data from a source into the repository) and data pull (searching/harvesting/gathering of data from the repository). In recent years, several organizations (e.g. IEEE/LTSC, IMS

Global, OKI, LETSI, ADL, CEN) have develop specifications and standards to address these interoperability issues [Fri05]. For the sake of readability only the most prominent specifications [LQ11a] are detailed organized in two facets: data push and data pull.

3.3.2.1 Data push

There are several protocols for publishing learning objects and/or their metadata to digital repositories. A learning object can be sent to a repository by value or by reference. In the former the publishing method embeds the learning object, after encoding, into the message that is sent to the repository. In the latter, the repository embeds a reference (e.g. URL) to the learning object that is being published.

The IMS DRI specification was created by the IMS GLC and provides a functional architecture and reference model for repository interoperability. The IMS DRI recommends common repository functions (e.g. submit, search, download). One of these functions is the submit function for submitting LO to a DRI compliant repository through the transmission of an IMS-compliant Content Package using SOAP Messages with attachments.

The Package Exchange Notification Services (PENS) protocol, developed by the AICC in 2005, supports a notification service for content packages. Using this service a source can announce the location of a package that is available for transport. When an application (e.g. LMS) receives a PENS notification, it can retrieve the package from the URL that is provided. The PENS specification contains an abstract data model and provides a binding to the HTTP protocol.

The Open Knowledge Initiative (O.K.I) created Open Service Interface Definitions (OSID) to enable the submission of assets (learning object and metadata) to a digital repository. The repository OSID includes a JAVA Asset interface that offer methods for adding and deleting records. The SRU Record Update service supports the creation, replacement and deletion of metadata records [McC06]. This specification can be implemented only on metadata resources.

The SWORD (Simple Web-service Offering Repository Deposit) standard allows digital repositories to accept the deposit of any content from multiple sources. SWORD is a profile of the Atom Publishing Protocol (AtomPub - a simple HTTP-based protocol for creating and updating web resources) restricting the scope of depositing resources into scholarly systems.

The Simple Publishing Interface (SPI) specification partly sponsored by the European Committee for Standardization (CEN) Workshop on Learning Technologies, defines a protocol that aims to facilitate the communication between content producing tools and repositories that persistently manage learning resources and metadata [TMT⁺10]. The SPI is an abstract model for publishing metadata and resources. A binding to a technology makes these

methods more concrete and defines how applications can interoperate. SPI has been bound to the Atom Publishing Protocol (APP or AtomPub), compatible with the SWORD profile which is widely used by institutional repositories.

3.3.2.2 Data pull

Learning objects are described by metadata stored in repositories. The latter should support different search/harvesting protocols to expose metadata to users and/or services.

One of the earliest search protocols was the Z39.50. It is a client-server protocol for searching and retrieving information from remote libraries. Z39.50 is a pre-Web technology (work on the Z39.50 protocol began in the 1970s). Since then there have been several efforts to evolve the protocol under the designation ZING (Z39.50 International: Next Generation).

One of the most important is the twin protocols SRU/SRW, which introduces a new communication protocol (HTTP) making the specification more lightweight. Search/Retrieve via URL (SRU) is REST based and enables queries to be expressed in URL query strings; Search/Retrieve Web service (SRW) uses SOAP. Both expect search results to be returned as XML. Queries in SRU and SRW are expressed using the Contextual Query Language (CQL) as a new query language that was based on the semantics of Z39.50. All these standards (SRW, SRU and CQL) are promulgated by the United States Library of Congress.

The IMS DRI also provides a recommendation for a search function. The Search reference model defines the searching of the meta-data associated with content exposed by repositories. It suggests two query languages: XQuery for searching IMS (XML) meta-data format and Z39.50 for searching library information.

The IMS Learning Object Discovery & Exchange (LODE) specification aims to facilitate the discovery and retrieval of learning objects stored across more than one collection. LODE is based on the following assumptions: 1) LO are described by metadata such as LOM or DC; 2) multiple metadata instances might be necessary in order to adequately describe all the aspect of a LO and to create searchable catalogues of LO using the Information for Learning Object eXchange (ILOX) framework; 3) repositories can be searched programmatically using SQI or SRU; 4) large catalogues can be created by harvesting (i.e., mirroring) metadata stored in repositories using protocols such as OAI-PMH.

The Simple Query Interface (SQI) specification, supported by CEN, presents an Application Programming Interface (API) for querying learning object repositories. SQI is neutral in terms of results format and query languages, thus it makes no assumptions about the query language or results format [SMvA⁺05]. OpenSearch - created by A9.com (an Amazon.com company) - is a collection of simple formats for the sharing of search results. The OpenSearch description document format can be used to describe a search engine so that it can be used

by search client applications. The OpenSearch response elements can be used to extend existing syndication formats, such as RSS and Atom, with the extra metadata needed to return search results.

The ProLearn Query Language (PLQL), developed by the PROLEARN "Network of Excellence", is a query language for repositories of learning objects. PLQL is primarily a query interchange format, used by source applications (or PLQL clients) for querying repositories (or PLQL servers). PLQL has been designed with the goal of effectively supporting search over LOM, DC and MPEG-7 metadata. However, PLQL does not assume or require these metadata standards.

Harvesting protocols enable pulling learning objects and metadata from a repository. An example of a metadata harvesting protocol is the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH). Large catalogues can be created by harvesting (i.e., mirroring) metadata stored in repositories using OAI-PMH. To obtain content, the OAI-PMH can be used in combination with a protocol for obtaining content, such as the NISO's OpenURL.

3.4 Summary

This chapter gathers information on e-learning standards organized in two levels: content and communication [JBK05, VA06]. At the content level the focus is on three facets: metadata, content aggregation and assessment. At the communication level the focus is on the study of several ways to communicate with LMSs and LORs. A new level (Frameworks) is added due to its relevance in the last years on the development of new e-learning systems based on SOA. This study groups the existent frameworks in abstract and concrete frameworks and compared them according to their architectural model, impact and maturity, adopted standards and user groups.

For each level, the most prominent standards and specifications and their main contributions in the e-learning field are enumerated based on surveys [QL11c, FCN⁺11]. This information is used in the Part II of this dissertation to choose the most suitable specifications and standards for the domain of automatic evaluation of programming exercises.

Part II

Architecture

Chapter 4

The Ensemble E-Learning Framework

"The hardest part of design ... is keeping features out."

Donald Norman

This chapter presents a proposal for an e-learning framework called the Ensemble E-Learning Framework (EeF). The EeF is a conceptual tool to organize a network of e-learning systems and services based on content and communications specifications. The name "Ensemble"¹ suggests the collaborative work of all the parties in a network to achieve a common goal. The EeF differs from the frameworks presented in section 3.1 in its focus and architectural model.

The EeF is exclusively focused on the teaching-learning process. The frameworks presented in section 3.1 cover areas that go beyond the scope of e-learning, from course to financial management. In this framework the focus is on the coordination of pedagogical services that are typical in everyday life of teachers and students at schools such as the creation, delivery, resolution and evaluation of assignments. This framework emphasizes the use of assessment services to automatically evaluate the attempts of students to solve exercises and to produce relevant feedback on their quality. The need for automatic assessment exists in different domains, for instance 1) an electronic circuit evaluator receive a description of a circuit, injects input signals, simulates the circuit and compares output signals; 2) a diagram evaluator receives a description of a diagram (e.g. UML) - a typed graph - and tries to create a graph homomorphism with a solution; 3) a programming exercise evaluator receives an attempt of a student to solve an exercise and the program is executed against test data,

¹In the music domain, a musical ensemble is a group of people who perform instrumental or vocal music.

and then the output of the program (or return value) is compared with a solution model.

Another distinctive feature of EeF is its architectural model. The EeF uses a model that can be described as a "decentralized orchestration". An implementation of the EeF uses a pivot component that orchestrates the communication with other services but is replicated and deployed for each end user. This novel approach avoids any single-point-of-failure issues that occur in central orchestrations.

In the following sections the framework is described based on three models. Firstly, the architectural model and its components are presented. Then, the interoperable facet of the framework is addressed by presenting its data and integration model.

4.1 Architectural model

The EeF is the basis for the design and implementation of **Ensemble instances** as realizations of the framework for specific domains (e.g. computer programming learning). Each instance can be deployed in several locations denominated as **Ensemble networks**. Several users can interoperate within a network using Web Services. The definition of how these Web services cooperate is typically based on coordination models (e.g. orchestration, choreography). In the EeF architectural model the services coordination is based on a "decentralized orchestration" where central components are replicated for each end user. This is a distinctive feature of this framework. Most e-learning frameworks fall in one of two architectural models: either based on layers of services, or on central communication nodes. Both architectural models present communication issues: layered models present unintended noise between the communication of non-contiguous layers and central communication nodes include a single-point-of-failure since all the communication relies in a central node. With this novel approach it may appear that the replication of components in the execution path would adversely affect performance, however decentralized execution brings performance benefits [CCMN04]:

- there is no centralized coordinator which can be a potential bottleneck;
- distributing the data reduces network traffic and improves transfer time.

Figure 4.1 shows the architectural model of the EeF. On the central axis that is perpendicular to the plan holding the network services resides the central components called axial systems. These central components communicate with services organized at two levels: core services that are crucial for the learning process or secondary services that are used to complement core services in a specialised task.

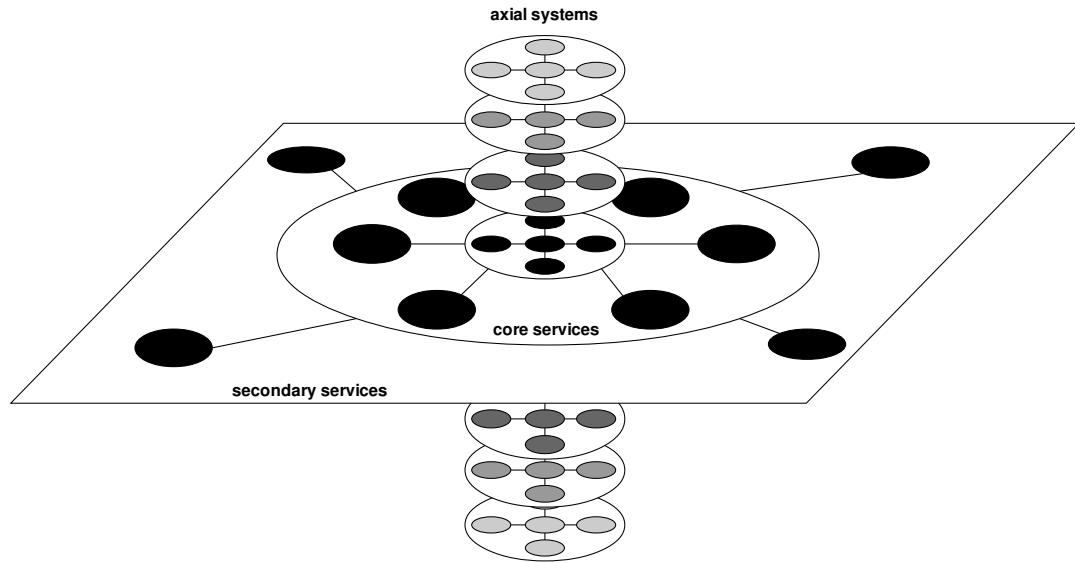


Figure 4.1: EeF architectural model.

The remainder of this section defines in detail the concepts of **axial systems**, **core services** and **secondary services**.

4.1.1 Axial systems

Axial systems are central components in the EeF architectural model. The main features of these systems are:

Centrality - they are able to communicate with all core services;

Locality - they are replicated on the computer of each end user of a specific network;

Interactivity - they mediate the interaction between the users (teachers and students) and the network by means of a user interface.

One of such systems assume a **pivot role**. The role of the pivot component is twofold: orchestration and interface.

The pivot component orchestrates the communication among services and is replicated for each end user. Since it is distributed over each network user this approach prevents the occurrence of any single-point-of-failure issues that might occur.

The pivot component also acts as the graphical interface between users and the network. In the EeF jargon a pivot component is called a Teaching Assistant (TA). A Human TA is a

person who assists a teacher in practical classes. The task of a automated TA is to act as an interface to users (both with teachers and students) and, unlike its human counterpart, to delegate most of its work to others, as it is fundamentally a coordinator of e-learning systems. For instance, in a programming course, an automated TA is used to help students with programming tools (integrated programming environments, compilers, and debuggers), check if they have solved the exercises and provide feedback to help them to overcome their difficulties. This type of tool can also be described as a scaffolding tool since it complements existing tools and was designed to be easily removed once it is no longer needed.

Apart from the TA, other systems reside in this axial area. Using again the programming domain as example, an experimentation system (e.g. IDE) can be used by students for solving programming exercises and may be extended to communicate with other services in the network. In the medical training domain, teachers use simulation models to improve students skills in several medical processes (e.g. birth) and to use medical tools properly, as they would in real world hospitals and clinics.

All these systems need to communicate with other services. The TA may need to interact with an assessment system to submit a student attempt to solve an exercise. A business simulation game may require a repository containing specialized LO describing simulations. Due to size, complexity and security issues, these services should be accessed remotely rather than being installed locally. The next subsection describes them.

4.1.2 Core and Secondary services

An Ensemble instance handles multiple pedagogical learning process. A learning process is a collection of related and structured activities implemented by e-learning services. A typical example of a pedagogical learning process is a classroom assignment. The teacher starts by setting a number of activities in the LMS, including the resolution of a number of relevant exercises in a specific domain obtained from a repository. The learner tries to solve the exercises set by the teacher using an experimentation system that recovers exercise descriptions from the repository. After solving the exercise the learner sends an attempt to an assessment system. The learner may submit repeatedly, integrating the feedback received from the assessment system. In the end the assessment system sends a grade to the LMS gradebook. Most of these services are commonly provided by e-learning systems such as:

Learning Management Systems - to manage and retrieve the exercises to the learners;

Learning Objects Repositories - to persist LO and related meta-information;

Assessment Systems - to evaluate and produce feedback on attempts to solve exercises;

E-Portfolio systems - to organize students achievements.

These types of services are very different in nature. Repositories and Assessment Systems provide truly specialized services. An LMS is not in strict sense a service. It is a system designed to be a complete and generic e-learning environment rather than a service. Nevertheless, since a typical LMS is a component based system, it may be extended to incorporate the features it lacks to communicate with other services.

Secondary services are complementary services that complement the core services in a specific task, although its absence does not alter the execution flow of a learning process. Usually these services do not have graphical interfaces and are more specialized than the core services. An example of this kind of services is an adaptation service. Taking the previous example, an adaptation service could adjust the presentation order in accordance with the effective difficulty of the exercises (not the difficulty stated on the LO) and the needs of a particular student. Other example is a service for handling the conversion between different exercises formats.

4.2 Data Model

The concept of Learning Object (LO) is the cornerstone for producing, sharing and reusing content in e-learning. The most widely used standard for LO is the IMS CP that uses the LOM standard to describe the learning resources included in the package. The QTI specification endows this data model with the capacity for describing questions and test data. Despite its widespread use, this specification is not adequate to specific domains [LQ09c, QL11e]. Recently, IMS GLC proposed the IMS CC that bundles the previous specifications and its main goal is to organize and distribute digital learning content.

The Ensemble data model specification is based on the Common Cartridge specification. This choice is sustained by the experiments of Kurilovas [Kur12] in the deployment of CC packages in an educational context and is also justified by the following features:

Packaging - flexible packaging via URL references to web content;

Communication - collaboration and web 2.0 mash-ups (provisioning of IMS LTI);

Security - content authorization via protected resources;

Content sharing - migration from other data sources (e.g. SCORM 2004);

Extension - augment of data by using new Learning Application Resources (LAO);

Integration - access to learner data using privileges and outcomes (e.g. IMS LIS).

Of all these reasons, the extension facet is the most relevant. The extension can be achieved by adding new LAO resources in the CC package. Each LAO must have a corresponding

resource element in the manifest. Physically, a LAO is a directory in the content package containing a descriptor file, its schema and additional files and subdirectories used exclusively by that LAO. These files held within a LAO directory structure are described as associated content resources. LAO resources differ from web content resources (e.g. HTML and image files) since they require additional processing and interpretation before they can be imported and subsequently used within the target system. Examples of LAO resources predefined in the CC specification are the QTI assessments and discussion forums. Figure 4.2 shows the structure of a CC package with the inclusion of a LAO resource.

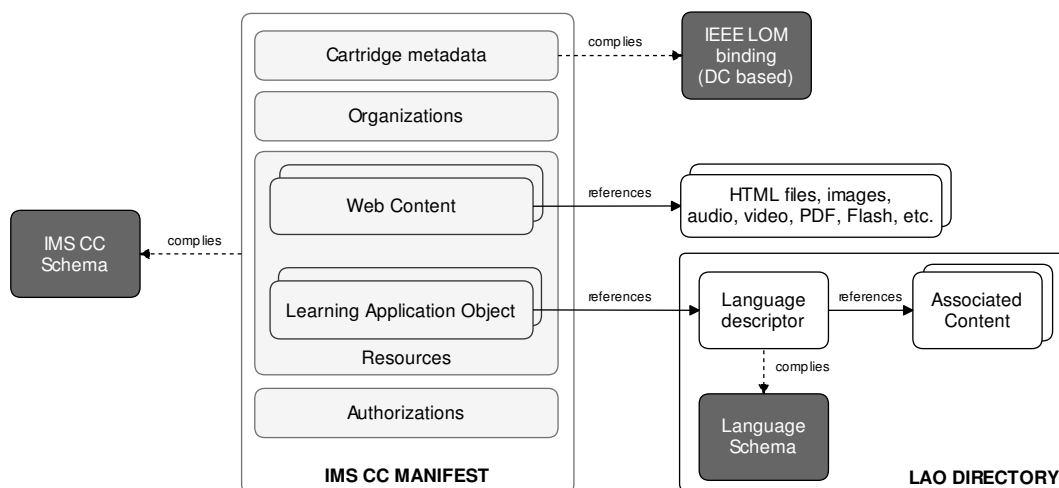


Figure 4.2: Structure of an IMS CC package.

An Ensemble instance must follow this data model. However, it is not mandatory to use the LAO extension mechanism. This should only be used in cases where the specification is not sufficient for a specialized domain. This is the case of the computer programming domain where programming exercises should be described. Current specifications (e.g. LOM, QTI) are insufficient and cannot describe how an exercise should be compiled and executed, which test data should be used or how it should be graded.

4.3 Integration Model

The Ensemble specification also comprises an integration model. This model recommends specifications for the communication between the axial systems and the core services. This section analyses the communication specifications for the interaction of axial systems with three core services typically found in e-learning environments: repositories, assessment systems and learning management systems. The recommendation of the EeF for the assessment is the **Text file evaluation specification** [LQ10g]. The selected specification for the communication with repositories is the **IMS DRI specification**. Finally, for

interacting with the LMS the Ensemble specification recommends the use of the **IMS LTI specification**. Other specifications may exist in an Ensemble instance even if not addressed by this model.

This integration model relies on web services for communication among systems. Web services can be used mostly in two flavours: SOAP and REST. SOAP web services are usually action oriented, mainly when used in Remote Procedure Call (RPC) mode and implemented by an off-the-shelf SOAP engine such as Axis. Web services based on the REST style are object (resource) oriented and implemented directly over the HTTP protocol mostly to put and get resources.

Both specifications have matured in distinct periods and they coexist nowadays. This explains why older specifications such as DRI recommends SOAP, while newest such as LTI are based on REST: SOAP started earlier and now the trend is REST. Figure 4.3 shows the web service types from a directory of 3200 web APIs listed at ProgrammableWeb² (May 2011).

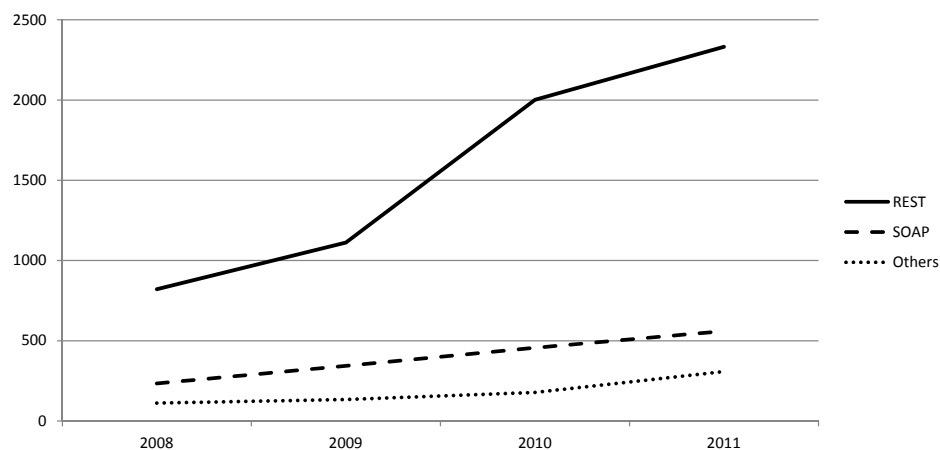


Figure 4.3: Trends on the use of SOAP and REST web services.

Regardless of these trends, the EeF specification does not encourage the use of any flavour in the communication specifications detailed in the following subsections. As far as possible the EeF tries to keep an equidistant position from both flavours.

4.3.1 Text File Evaluation Service Genre

Text file evaluation responds to the shortcomings of the assessment based on questions with predefined answers. Questions with predefined answers are formalized in languages such as

²Official web site: <http://www.programmableweb.com/>

IMS QTI and supported by many e-learning systems. Complex evaluation domains justify the development of specialized evaluators that participate in several business processes as autonomous services. The definition of this specification was inspired by the service definition model of the E-Framework³. The specification consists of an abstract service type (service genre) that describes a text file evaluation service. A service of this genre is responsible for the assessment of a text file with an attempt to solve an exercise described by a LO. The service modelled by the proposed definition receives a text file with an attempt to solve an exercise and produces an evaluation report. The exercise is referenced as a LO available on an interoperable repository. The abstract service supports three functions:

- The **ListCapabilities** function lists all the capabilities supported by a specific evaluator;
- The **EvaluateSubmission** function evaluates a submission to a given exercise, using an available capability;
- The **GetReport** function provides a detailed report of a previous evaluation.

4.3.1.1 The ListCapabilities function

The **ListCapabilities** function informs the client systems of the capabilities of a particular evaluator. Capabilities depend strongly on the evaluation domain. For instance, in a computer programming evaluator the capabilities are related with the features of compilers or interpreters. Each capability has a number of features to describe it and for a programming language they may be the name of the language (e.g. Java), its version (e.g. 1.7) and vendor (e.g. JDK). On an electronic circuit simulator a capability may be a collection of gates that are allowed on a circuit and features may be the names of individual gates. A schematic view of this function is shown in Figure 4.4.

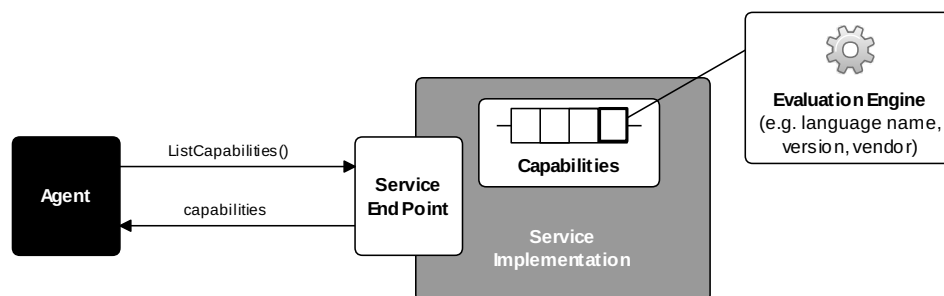


Figure 4.4: The ListCapabilities function.

³Official web site of E-Framework for Education and Research - <http://www.e-framework.org>

In this function, the request does not accept any parameter and the response returns a list of all capabilities of the evaluator. Each capability is described by a list of features, with a name and a value. The format of this listing is outside of the scope of this specification and must be defined by the concrete service definition.

4.3.1.2 The EvaluateSubmission function

The **EvaluateSubmission** function requests the evaluation of an attempt to solve a specific exercise. The request includes an exercise or a reference to an exercise represented as a learning object held in a repository and a text file with a single attempt to solve that particular exercise. The request may include a specific evaluator capability necessary for the proper evaluation of that attempt.

The response returns either a ticket for a later report request or a report on the evaluation. In any event the response will include a ticket to recover the report on a later date. A schematic view of this function is shown in Figure 4.5. The service endpoint provides the interfaces for the requests and responses for the evaluation functionality. Internally the service implementation may include several features (indexing, queuing, transforming, flow control, etc.) needed to provide the defined functionality and a connection with a remote data source holding the objects such as a LOR.

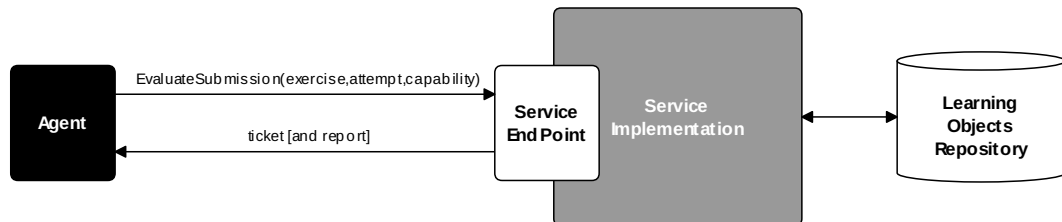


Figure 4.5: The EvaluateSubmission function.

The evaluator returns a report on the evaluation, if it is completed within a predefined time frame. The report should contain detailed information on the assessment rather than a verdict such as passed or failed. The format of the report sent to the client should be designed for using it as input to other systems (e.g. classification systems, feedback systems) and may be, for instance, transformed in the client side based on a XML stylesheet. The specification of the response format is outside the scope of this specification and must be defined by the concrete service definition. Requesting a report using a ticket is supported through another function called **GetReport** detailed in the next sub-subsection.

4.3.1.3 The GetReport function

The **GetReport** function allows a requester to get a report for a specific evaluation. This way the client will be able to filter out parts of the report and to calculate a classification based on its data. A schematic view of this function is shown in Figure 4.6. The request of this function includes a ticket sent previously by the service in response to an evaluation. The response returns an evaluation report.

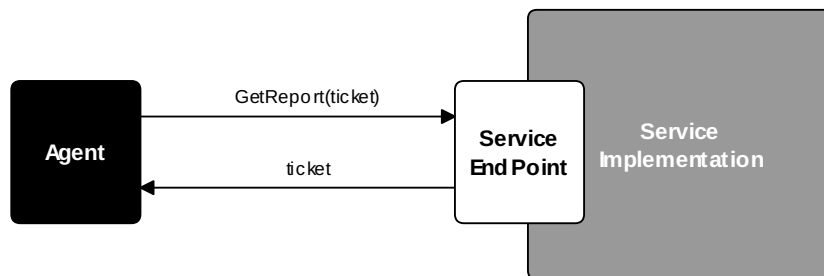


Figure 4.6: The GetReport function.

4.3.2 Digital Repositories Interoperability

The IMS Digital Repositories v1.0 specification⁴, released in 2003, aims to provide recommendations for the interoperation of the most common repository functions (Figure 4.7). In order to use the IMS DRI these recommendations should be implementable across systems to enable them to present a common interface for those functions. These core functions are:

- The **Submit** function defines how an object is moved to a repository from a given network-accessible location and how the object will then be represented in that repository for access. The recommended communication protocol is SOAP with attachments⁵ with the attachments taking the form of one or more IMS-compliant Content Packages;
- The **Search** function defines the searching of the meta-data associated with the content exposed by repositories. Two protocols are suggested: XQuery over SOAP (for learning object repositories) and Z39.50 (for libraries). Searching is performed using the XQuery protocol over XML meta-data, adhering to the IMS Meta-Data Schema;
- The **Request** function allows a client that has located a meta-data record via the **Search** function to request access to the LO described by that meta-data;
- The **Alert** function defines an intermediary aggregation service and envisions that e-mail/SMTP (Simple Mail Transfer Protocol) could provide this functionality.

⁴<http://www.imsglobal.org/digitalrepositories/>

⁵<http://www.w3.org/TR/SOAP-attachments>

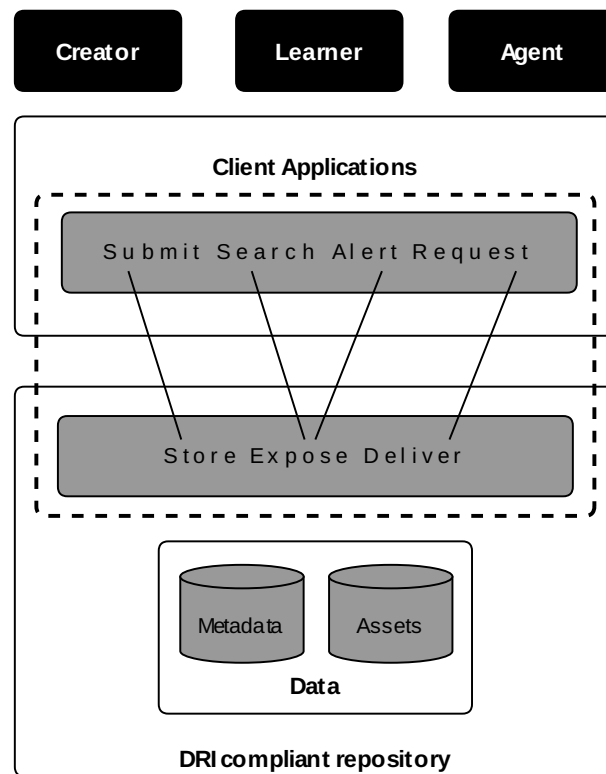


Figure 4.7: The IMS DRI Specification.

The DRI specification also includes a messaging model. This model standardizes general communication between the components defined by the DRI architecture. The basic message has two parts: a message header and a message body. XML bindings are also provided by this specification but further extensions are needed to enhance the communication between the repository and other systems [RSA06, EHR04].

4.3.3 Learning Tools Interoperability

IMS developed the Learning Tools Interoperability v.1.0 (LTI) specification in 2010. This recent specification provides a standard way of integrating rich learning applications - in LTI called Tool Providers (TP) - with platforms like LMSs, portals, or other systems from which applications can be launched - called Tool Consumers (TC). LTI v1.0 defines a formal deployment process whereby the LMS and the application reach an agreement on the run-time services and security policies. In order to accelerate the conformance to this new specification by Tool Consumers the IMS launched also a subset of the full LTI v1.0 specification called IMS Basic LTI. This subset exposes a single (but limited) link between the LMS and the application as shown in Figure 4.8.

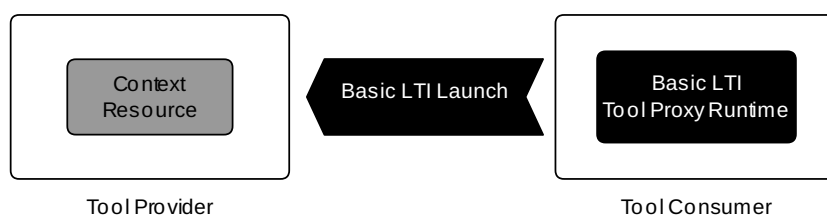


Figure 4.8: The IMS Basic LTI Specification.

Basic LTI focuses on launching Web tool interfaces. It does not include provisions for standardized web service callbacks and only one security policy is supported based on the OAuth protocol⁶. The Tool Proxy Runtime manages the launch request performed by the Tool Consumer. A context resource is sent as HTTP POST for the Tool Provider. The Tool Provider verifies the authenticity of the request using an OAuth key and allows the access to the context data by the Tool Provider user.

In March 2012 IMS launched the LTI v1.1 (final version) merging both specifications (Basic LTI and LTI). This new version includes the support for an outcomes service based on a subset of the IMS Learning Information Services (LIS)⁷ - the LTI Basic Outcomes Service. The LIS specification is the definition of how systems manage the exchange of information that describes people, groups, memberships, courses and outcomes within the context of learning. Figure 4.9 shows how the bidirectionality of the LTI specification is performed.

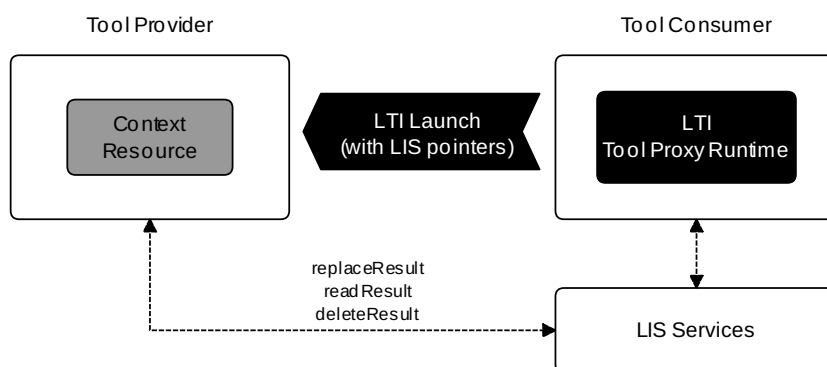


Figure 4.9: The IMS LTI Specification v1.1 - integration with LIS services.

TC provides launch data with LIS pointers to the TP. It is not required for the TC to provide these services. The LIS services could even be provided by a third system such as a Student Information System. Then, the TP calls the LTI Basic Outcomes Service if available. This service receives "Plain Old XML" (POX) messages signed using OAuth.

⁶Official Web site: <http://oauth.net/>

⁷Official Web site: <http://www.imsglobal.org/lis/>

The service supports setting, retrieving and deleting LIS results associated with a particular user/resource combination. The following functions are supported:

- The `replaceResultRequest` function sets a numeric grade (0.0 - 1.0) for a particular result;
- The `readResultRequest` function returns the current grade for a particular result;
- The `deleteResultRequest` function deletes the grade for a particular result.

4.4 Summary

This chapter presents the Ensemble framework as a conceptual tool to organize a network of e-learning systems and services based on content and communications specifications. These specifications were selected based on the analysis presented in chapters 2 and 3 on the current standards and systems in the e-learning realm. The framework presents an abstract data and integration model that should base the implementation of networks in specialized domains with complex evaluations. The implementation of this framework for the computer programming domain is presented in the next chapter. This framework instance is validated in the chapter 9.

Actually, the framework presented was abstracted from the concrete application to programming exercises presented in the following chapter. It is expected that this abstract framework may be applied to other domains, unrelated to the domain that motivated this research. In a strict sense the EeF cannot be called as a framework since it was only "applied" to a single instance. It will be part of the future work, resulting from this dissertation to apply the EeF to other domains and validated it as a framework.

Chapter 5

Specializing Ensemble to computer programming

*"Don't worry if it doesn't work right.
If everything did, you'd be out of a job."*

Mosher's Law of Software Engineering

This chapter presents the specialization of the Ensemble framework for the computer programming domain. Firstly, the overall architecture of this Ensemble instance is presented. Secondly, the data and the integration models are shown. The data model relies on the creation of an interoperability language for describing programming exercises called PExIL. This new language is included as a Learning Application Object (LAO) in an IMS CC package as recommended by the Ensemble specification. The integration model details how systems and services of this Ensemble instance are connected through the extension of the recommended specifications of the EeF. The integration model includes the creation of a new service for the communication with the AS based on the Evaluate service genre and the extension of the IMS DRI and LTI specifications for the integration of LOR and LMS, respectively. Finally, a selection of tools is presented according to the models presented.

5.1 Architecture

This section presents the overall architecture of a network of e-learning systems and services participating in the automatic evaluation of programming exercises. The architecture (Figure 5.1) is composed by the following components:

Teaching Assistant (TA) to interface the users with the network and to mediate the communication among all components;

Integrated Development Environment (IDE) to code the exercises;

Assessment System (AS) to evaluate exercises of students;

Learning Objects Repository (LOR) to store and retrieve exercises;

Learning Management System (LMS) to present the exercises to students;

Conversion System (CS) to convert exercises formats.

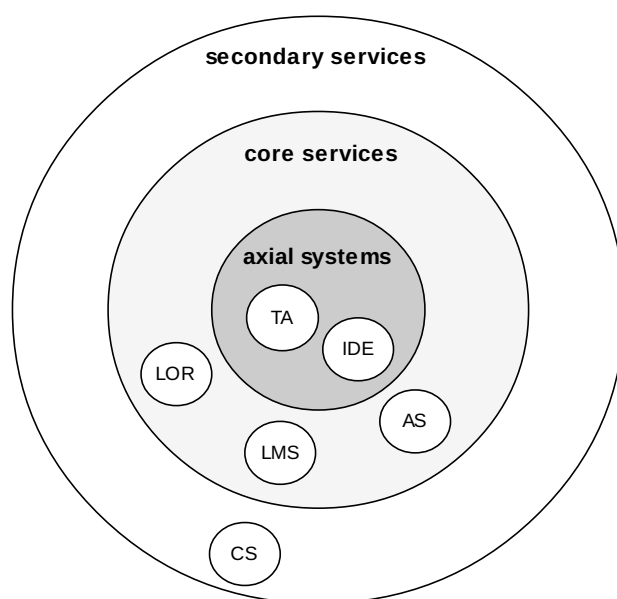


Figure 5.1: Overall architecture of the Ensemble instance.

5.2 Data model

The concept of Learning Object (LO) is fundamental for producing, sharing and reusing content in e-Learning [Fri05]. In essence a LO is a container with educational material and metadata describing it. Since most LOs just present content to students they contain documents in presentation formats such as HTML and PDF, and metadata describing these documents using LOM (or other metadata format). When a LO includes exercises to be automatically evaluated by an e-learning system, it must contain a document with a formal description for each exercise. The QTI specification is an example of a standard for this kind of definitions that is supported by several e-learning systems. However, QTI was designed

for questions with predefined answers and cannot be used for complex evaluation domains such as the programming exercise evaluation [LQ09c]. In particular, the programming exercise domain requires interdependent resources (e.g. test cases, solution programs, exercise description) usually processed by different services in the programming exercise life-cycle. This kind of data cannot be characterized as metadata as they are data effectively needed for evaluation.

The data model of the Ensemble instance extends the IMS CC specification for describing programming exercises. This extension is achieved by adding a new LAO resource in the CC package (Figure 5.2) as recommended by the Ensemble specification.

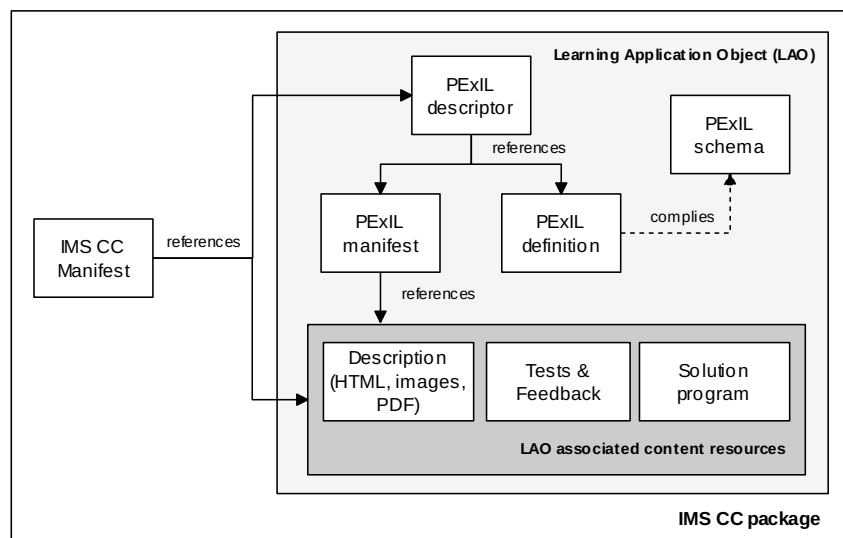


Figure 5.2: Ensemble instance data model.

A LAO resource contains an XML descriptor that serves as the entry point for the target system. The descriptor file is not intended to be displayed within the target system. Rather, it is intended to be processed by the target system upon import of the cartridge. The descriptor file (`pexil.xml`) is associated with a LAO by means of a `file` element. It includes references for two XML documents: `pexildefinition.xml` and `pexilmanifest.xml`. The former describes all data needed for the generation of the evaluation resources. The latter is a manifest with references for all the evaluation resources generated. These resources - called associated content resources - comprise the exercise description, tests and feedback files and the solution program.

This section focuses on the definition of an XML dialect called PExIL (Programming Exercises Interoperability Language). The aim of PExIL is to consolidate all the data required in the programming exercise life-cycle, from when it is created to when it is solved and graded. Then, the XML Schema used to formalize the relevant data of the programming exercise life-

cycle is presented. Finally, the validation of this approach is made through the evaluation of the expressiveness of the PExIL definition. In this evaluation the PExIL definition is used to capture all the constraints of a set of programming exercises stored in a learning objects repository.

5.2.1 The life-cycle of a programming exercise

A programming exercise requires a collection of files (e.g. test cases, solution programs, exercise descriptions, feedback) and special data (e.g. compilation and execution lines). These resources are interdependent and processed in different moments in the life-cycle of the programming exercise. The life cycle comprises several phases:

1. in the **creation phase** the content author should have the means to automatically create some of the resources (assets) related with the programming exercise such as the exercise description and test cases and the possibility to package and distribute them in a standard format across all the compatible systems (e.g. LMS, LOR);
2. in the **selection phase** the teacher must be able to search for a programming exercise based on its metadata from a repository of learning objects and store a reference to it in a learning management system;
3. in the **presentation phase** the student must be able to read the exercise description in its native language and a proper format (e.g. HTML, PDF);
4. in the **solving phase** the learner should have the possibility to use test cases to test his attempt to solve the exercise and to automatically generate new ones;
5. in the **evaluation phase** the evaluation engine should receive specialized metadata to properly evaluate the learner's attempt and return enlightening feedback.

All these phases require a set of inter-dependent resources and specialized metadata whose manual creation would be time-consuming and error-prone.

5.2.2 PExIL

This subsection presents PExIL, an XML dialect that aims to consolidate all the data required in the programming exercise life-cycle. This definition is formalized through the creation of a XML Schema depicted in Figure 5.3.

The following subsections presents the PExIL XML Schema organized in three groups of elements:

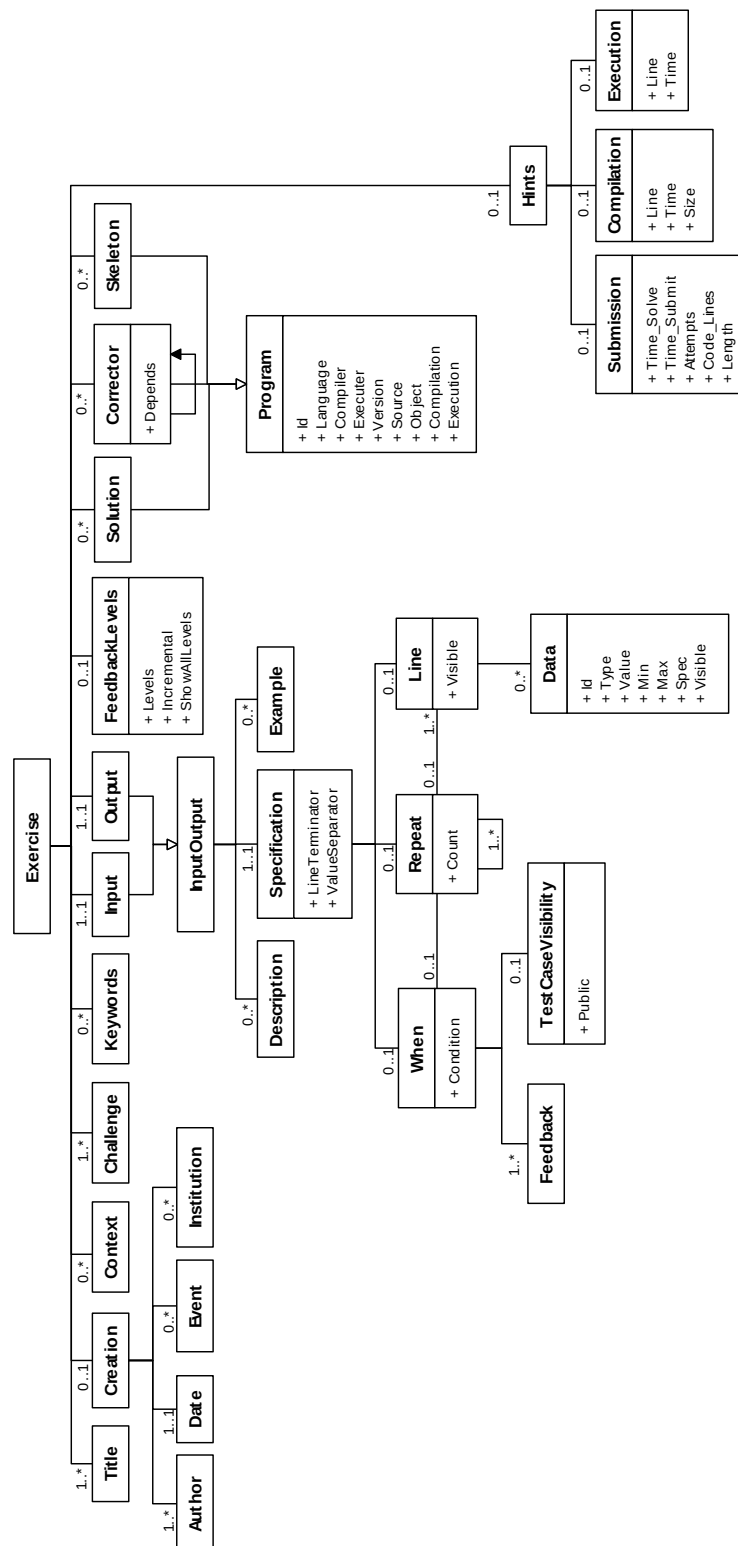


Figure 5.3: PExIL data model.

1. **Textual** – elements with general information about the exercise to be presented to the learner. (e.g. title, date, challenge);
2. **Specification** – elements with a set of restrictions that can be used for generating specialized resources (e.g. test cases, feedback);
3. **Programs** – elements with references to programs as external resources (e.g. solution program, correctors) and metadata about those resources (e.g. compilation, execution line, hints).

5.2.2.1 Textual Elements

Textual elements contain general information about the exercise to be presented to the learner. This type of elements can be used in several phases of the programming exercise life-cycle: in the selection phase as exercise metadata to aid discoverability and to facilitate the interoperability among systems (e.g. LMS, IDE); in the presentation phase as content to be present to the learner (e.g. exercise description); in the resolution phase as skeleton code to be included in the student's project solution. Table 5.1 presents the textual elements of the PExIL schema and identifies the phases where they are involved.

Table 5.1: Textual elements.

Element	Selection	Presentation	Resolution	Evaluation
title	X	X	-	-
creation/authors/author	X	X	-	-
creation/date	X	X	-	-
creation/purpose	X	X	-	-
challenge	-	X	-	-
context	-	X	-	-
skeleton	-	X	X	-

The **title** element represents the title of the programming exercise. This mandatory element uses the `xml:lang` attribute to specify the human language of the element's content. The definition of this element in the XML Schema has the `maxOccurs` attribute set to unbound allowing the same information to be recorded in multiple languages. The **creation** element contains data on the authorship of the exercise and includes the following sub-elements: **authors** with information about the author(s) of the exercise organized by several **author** elements (represented as RDF elements); **date** which includes the date of the generation of the exercise and *purpose* that describes the event for which the exercise was created or the institution where the exercise will be used. The **context** element is an optional field used to contextualize the student with the exercise. The **challenge** element is the actual

description of the exercise. Its content model is defined as mixed content to enable character data to appear between XHTML child-elements. This XML markup language will be used to enrich the formatting of the exercises descriptions. The **skeleton** element refers to a resource containing code to be included in the student's project solution.

5.2.2.2 Specification Elements

The goal of defining programming exercises as learning objects is to use them in systems supporting automatic evaluation. In order to evaluate a programming exercise the learner must submit a program in source code to an Assessment System (AS) that judges it using predefined test cases - a pair of input and output data. In short, the AS compiles and runs the program iteratively using the input data (standard input) and checks if the result (standard output) corresponds to the expected output. Based on these correspondences the AS returns an evaluation report with feedback.

In the PExIL schema, the **input** and **output** top-level elements are used to describe respectively the input and the output test data. These elements include three sub-elements: **description**, **example** and **specification**. The **description** element includes a brief description of the input/output data. The **example** element includes a predefined example of the input/output test data file. Both elements comply with the **specification** element that describes the structure and content of the test data.

Table 5.2: Specification elements.

Element	Selection	Presentation	Resolution	Evaluation
input/specification	-	X	X	X
output/specification	-	X	X	X

This definition can be used in several phases of the programming exercise life-cycle as depicted in Table 5.2: by 1) the content author to automatically generate an input and output test example to be included on the exercise description for presentation purposes; 2) the learner to automatically generate new test cases to validate his attempt; 3) the AS to evaluate a submission using the test cases.

The **specification** element (Figure 5.4) contains two attributes and three top-level elements. The attributes **line_terminator** and **value_separator** define respectively the newline and space characters of the test data. The three top-level elements are: the **line** element which defines a test data row, the **repeat** element which defines an iteration on a set of nested elements controlled by the value of the **count** attribute and the **when** element which evaluates specific conditions for the generation of test cases and respective feedback.

The **line** element defines a data row. Each row contains one or more variables. A variable

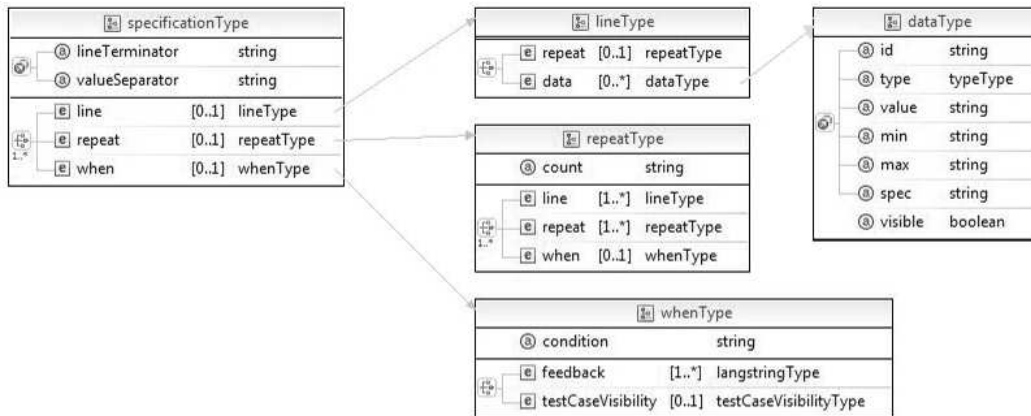


Figure 5.4: The specification element.

in the specification model must have a unique name which is used to refer to values from one or more places in the **specification** element. A variable is represented in the PEXIL schema with the **data** element containing the attributes:

- **id** - defines the name of the variable. To access a variable one must use the **id** attribute preceded by the dollar character ("\$\$") to enable the further resolution and evaluation of XPath expressions while processing the specification model;
- **type** – defines the variable data type (e.g. integer, float, string, enum). In the case of an enumeration the values are specified as a text child node;
- **value** – represents the value to be included in the input/output test file. If filled the variable acts as a constant. Otherwise, the value can be automatically generated based on a set of constraints - the **type**, **min**, **max** or **spec** attributes;
- **min/max** – represents value constraints by defining limits on the values. The semantic of these attributes depends exclusively on the data type: may represent the ranges of a value (integer and float), the minimum/maximum number of characters (string) or a range of values to be selected from an enumeration list;
- **spec** - complementary specification to generate values. For instance, regular expression for generating/matching strings of text, such as particular characters, words, or patterns of characters.

The following XML excerpt shows the specification elements for the input and output test data of an exercise. The exercise challenge is *"given three numbers check if the last number is between the first two."*

The input begins with a single positive integer on a line by itself indicating the number of the cases following (line 2). This line is followed by a blank line (line 3) and there is also a blank line (line 10) between two consecutive inputs. Each line of input contains three float numbers num1, num2 and num3 ranging values between 0 and 1000 (lines 5-9). In line 12 the **when** element forces the generation of a test case that complies with the **condition** attribute value.

Listing 5.1: Example of the input test description.

```

1 <specification line_terminator="\n" value_separator=" ">
2   <line><data id="numTestCases" type="int" value="3"/></line>
3   <line/>
4   <repeat count="$numTestCases">
5     <line>
6       <data id="num1" type="float" min="0" max="1000"/>
7       <data id="num2" type="float" min="0" max="1000"/>
8       <data id="num3" type="float" min="0" max="1000"/>
9     </line>
10    <line/>
11  </repeat>
12  <when condition="$num1>$num2">
13    <feedback xml:lang="en-GB">
14      Numbers can be given in descending order
15    </feedback>
16  </when>
17 </specification>

```

The output test description is shown in Listing 5.2. The output must contain a boolean for each test case separated by a blank line between two consecutive outputs (lines 4 to 7).

Listing 5.2: Example of the output test description.

```

1
2 <specification line_terminator="\n" value_separator=" ">
3   <repeat count="$numTestCases">
4     <line>
5       <data id="result" type="enum" value="1">True False</data>
6     </line>
7     <line/>
8   </repeat>
9 </specification>

```

As said before, the AS is the component responsible for the assessment of an attempt to solve a particular programming exercise posted by the student. The assessment relies on predefined

test cases. Whenever a test case fails a static feedback message (e.g. "Wrong Answer", "Time Limit Exceed", and "Execution Error") associated with the respective test case is generated. The PExIL schema includes a **feedback** element in the **specification** element to complement the static feedback of the evaluator. This element defines a dynamic feedback message to be presented to the student based on the evaluation of an XPath expression included in the **when** attribute. This expression can include references to input and output variables or even dependencies between both. If the expression is evaluated as true then the text child node of the **feedback** element is used as the feedback message.

PExIL supports the concept of incremental feedback to control the presentation of both types of feedback. The **feedbackLevels** element is a top-level child element which defines a set of feedback levels that the exercise supports and when it is shown to the student. Listing 5.3 shows an example of a **feedbackLevels** element.

Listing 5.3: Example of the feedback element.

```

1 <pexil:feedbackLevels
2   levels="simple|count_classifications|test_case_feedback_hint"
3   incremental="2"
4   showAllLevels="false" />

```

The levels attribute may have one or more feedback levels. The existent levels are:

- **Simple** – a feedback message indicating whether the student’s attempt is correct or incorrect (e.g. "Wrong answer!");
- **Count_worst_classification** – a feedback message indicating the worst classification of all the tests (e.g. "1 test with wrong answer");
- **Count_classifications** – a feedback message indicating the classifications of all tests (e.g. "3 tests accepted and 1 test with wrong answer");
- **Test_case_feedback_hint** – a feedback message to be presented to the student based on the evaluation of a condition defined by the content author. This feedback level is pedagogically relevant since the teacher can cover common errors of his students and warn them with useful and contextual feedback (e.g. "Forgot to divide by the number of input elements");
- **Test_case_input_result** – a feedback message including the input data of an unsuccessful test case (e.g. "Unexpected output for the test with the input data: 5 6");
- **Test_case_input_output** – a feedback message with the input and the output data of an unsuccessful test case (e.g. "Unexpected output for the test with the input data: '5 6' and the output data: '5,5' ").

The **incremental** attribute defines the number of times that a given level of feedback is shown. The **showAllLevels** attribute defines if the feedback to be presented to the student should accumulate with previous ones. In the last example were defined three levels of feedback. Based on the **incremental** attribute value the two first students' unsuccessful attempts will receive a simple feedback, the next two a **count.classification** feedback and so on.

5.2.2.3 Program Elements

Program elements contain references to program source files as external resources (e.g. solution program, correctors) and metadata on those resources (e.g. compilation, execution line, hints). These resources are used mostly in the evaluation phase of the programming exercise life-cycle (Table 5.3) to allow the AS to produce an evaluation report of an attempt to solve a programming exercise.

Table 5.3: Program elements.

Element	Selection	Presentation	Resolution	Evaluation
solution	-	-	X	X
corrector	-	-	-	X
hints	X	-	-	X

A program element is defined with the **programType** type depicted in Figure 5.5.

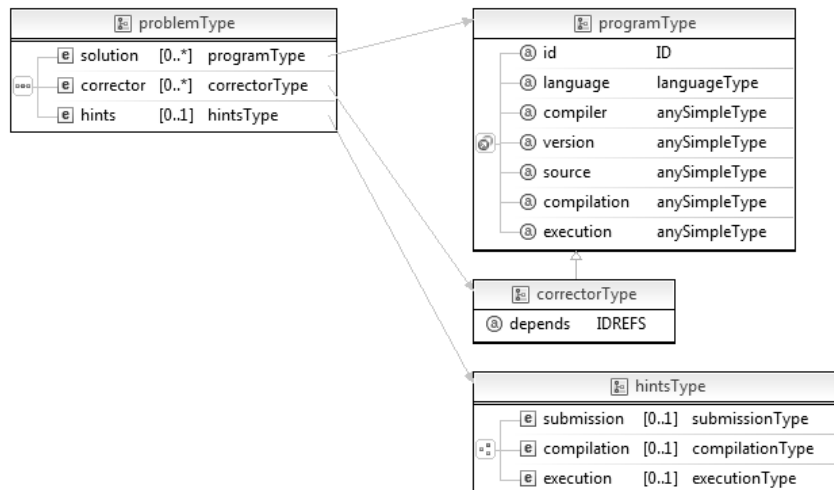


Figure 5.5: The program element.

This type is composed by seven attributes: **id** – an unique resource identifier; **language** – identifies the programming language used to code the resource (e.g. JAVA, C, C#, C++,

PASCAL); **compiler/executer** – defines the name of the compiler/executer; **version** – identifies the version of the compiler; **source/object** – defines the name of the program source/object file; **compilation** – defines a command line to compile the source code; and **execution** – defines a command line to execute the compiled code.

There are two program elements in the PExIL schema: the **solution** and the **corrector** elements. The **solution** element contains a reference to the program solution file. Listing 5.4 shows an example of a *solution* element.

Listing 5.4: Example of the **solution** element.

```

1 <solution
2     id="solution" language="JAVA"
3     compiler="javac" executer="java"
4     version="1.6" source="solution.java" object="solution"
5     compilation="$compiler $source"
6     execution="executer $object" />

```

The **corrector** element is optional and refers to custom programs that change the general evaluation pattern. There are two types of correctors: static and dynamic correctors. The **static corrector** is invoked immediately after compilation, before execution. The corrector can be used, for instance, to compute software metrics on the source code, judging the quality of source code; to perform unit testing on the program; to check the structure of the program's source code. The **dynamic corrector** is invoked after each execution with a test case. It is typically used to deal with non-determinism, for instance, if the output is a set of unordered values, the corrector sort it before comparing it with the expected output. A single programming exercise may use an arbitrary number of correctors. The order in which they are executed is defined by the **depends** attribute extending the **programType** type.

The **hints** element aggregates a set of recommendations for the submission, compilation and execution of exercises. These recommendations can be used by the AS to improve the evaluation and feedback process. The **hints** element is composed by three sub-elements: **submission**, **compilation** and **execution** elements.

The **submission** element defines guidelines for the submission process. It is composed by the following attributes: **time-solve** – time limit to solve the exercise; **time-submit** – time limit to submit the exercise; **attempts** – maximum number of attempts to submit the problem; **code-lines** – maximum number of code lines in the code of the user; **length** – maximum length in the code of the user.

The **compilation** element defines guidelines for the compilation process. It is composed by the following attributes: **time** – time limit to compile the exercise; **size** – maximum size of the execution code.

The `execution` element defines guidelines for the execution process. It is composed by a single attribute: `time` - time limit to execute the exercise.

5.2.3 Evaluating PExIL

In this subsection the expressiveness of PExIL definition is validated. For the evaluation process 24 programming exercises were randomly selected (1% of a total of 2393 exercises) from a repository based on crimsonHex [LQ09a]. These exercises were originally supplied by the administrators of the UVA Online Judge repository¹. The sample exercises were manually converted to PExIL and checked if the specification covered all the necessary constraints. The evaluation results, depicted in the Figure 5.6, shows that in most cases (21 – 88%), PExIL was expressive enough to cover the constraints of the exercise test data. In just one case, a minor change was to be made in the PExIL definition to capture alternative content models.

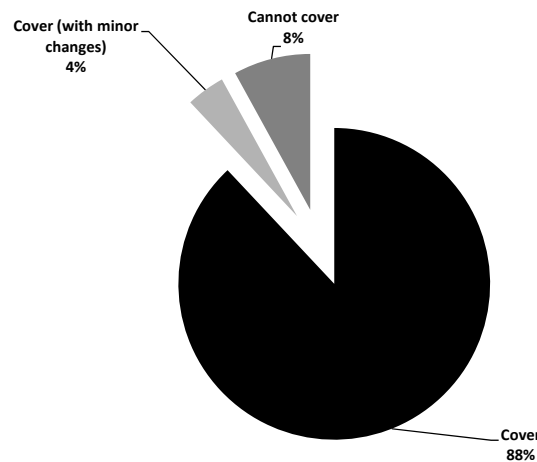


Figure 5.6: Evaluation of PExIL expressiveness.

Finally, two exercises were not completely covered by the PExIL definition. This means that using only the standard data types of PExIL it is possible to define the input and output files, and these definitions can be used to validate them. However, these definitions cannot be used to generate a meaningful set of test data. In these cases the programming exercise author would have to produce test files by some other means (either by hand or using a custom made generator). However, the data types required by these exercises are comparatively rare and do not justify their inclusion in the standard library. Moreover, PExIL does not restrict data types and a generator of exercises can be extended with generators for other data types, if this proves necessary.

¹Official Web site: <http://uva.onlinejudge.org/>

5.3 Integration model

The integration model depicted in Figure 5.7 relies on communication standards recommended by the Ensemble specification. This network model abstracts specific systems and focuses on system types. For instance, it is possible to use in this network any repository as long it supports the IMS CC specification to formalize the description of programming exercises and it implements the IMS DRI specification for communication with other services.

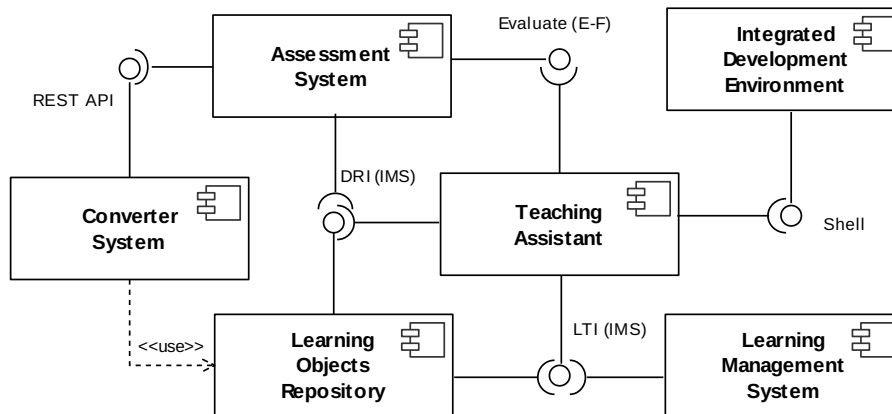


Figure 5.7: Network component diagram.

The next subsections report on the efforts made to integrate the systems and services of this Ensemble instance. These efforts include the creation of a new service for the communication with the AS based on the Evaluate service genre and the extension of the IMS DRI and LTI specifications for the integration of LOR and LMS, respectively.

5.3.1 Digital Repositories Interoperability

This subsection details the extensions made to the DRI specification to accommodate the new domain requirements. These extensions were made at three levels:

Web services - to promote the use of compliant repositories adjusted to different architectural styles;

Core functions - to endow compliant repositories of new capabilities (e.g. management capabilities);

XML binding - to formalize the responses from compliant repositories.

The following sub-subsections detail these extensions.

5.3.1.1 Interface definition

The DRI specification recommends SOAP as the specification protocol for exchanging LO in the implementation of Web Services. The extension adds a new web service flavour called REST. The reason to implement two distinct web service flavours is to promote the use of the repository by supporting different architectural styles. The repository functions are summarized in Table 5.4.

Table 5.4: Core and extension functions of DRI.

Function	SOAP	REST
<i>Register</i>	<i>URL getNextId()</i>	<i>GET /?nextId > URL</i>
Submit	submit(URL loid, LO lo)	PUT URL < LO
Request	LO request(URL loid)	GET URL > LO
<i>RequestAsset</i>	<i>LO requestAsset(URL loid, String asset)</i>	<i>GET URL/asset > ASSET</i>
Search	XML search(XQuery query)	POST / < XQUERY > XML
<i>Report</i>	<i>Report(URL loid, LOReport rep)</i>	<i>PUT URL < LOREPORT</i>
Alert	RSS getUpdates([Integer minutes])	GET /?alert+minutes > RSS
<i>Create</i>	<i>XML Create(URL collection)</i>	<i>PUT URL</i>
<i>Remove</i>	<i>XML Remove(URL collection)</i>	<i>DELETE URL</i>
<i>Status</i>	<i>XML getStatus()</i>	<i>GET /?status > XML</i>

Each function is associated with the corresponding operations in both SOAP and REST web services interfaces. The SOAP interface exposes a method using RPC and the second column of Table 5.4 presents its signature. For the REST interface is shown the HTTP method (GET, POST, or PUT), the requested URL and its input and output, following the Unix syntax of redirection operators. Strings in italic are replaced by values of that type. The lines formatted in italics correspond to the new functions added to the DRI specification to enrich compliant repositories of new capabilities improving the repository communication with other e-learning systems. Next, the Core functions are enumerated and briefly explained.

The **Register** function enables client systems to request a unique ID from a compliant repository. This function is separated from the **Submit** function in order to allow the inclusion of the ID in the meta-data of the LO itself. This ID is an URL that must be used for submitting or retrieving an LO. The producer may use this URL as an ID with the guarantee of its uniqueness and with the advantage of being a network location from where the LO can be downloaded. This action is performed by sending a GET HTTP request to the server. The HTTP response includes in the HTTP Location header the URL returned by the repository.

The **Submit** function uploads an LO to a repository and makes it available for future access.

This operation receives an argument providing an IMS CC compliant file and an URL generated by the **Register** function. This operation validates the LO conformity to the IMS CC Conformance and, in case of success, stores the LO in the internal database of the repository. In order to send a LO to the server through REST, the PUT or the POST HTTP methods can be used. The repository should respond with submission status data compliant with the Response specification language. Client systems should lookup for validation errors in the **error** element of the response specification.

The **Request** and **RequestAsset** functions get a LO or part of it from the repository. The last function is an extension made to the DRI specification. In order to get a LO asset from the server one should use the GET HTTP request. The request URL should be composed by the ID of the LO plus the identification of the asset (e.g. **description**, **solution**, **test**) to retrieve. An asset can be an exercise description (e.g. PDF file, HTML file + images); a solution program or a set of public input/output/feedback files from a test case. For instance, the request URL `http://crimsonHex/lo/easy/12/description` retrieves the description of the exercise 12 stored in the collection **easy** from an instance of the crimsonHex hosted at `http://crimsonHex/lo`.

The **Search** function enables client systems to query the repository using the W3C XQuery language², as recommended by the IMS DRI. This approach gives more flexibility to perform any queries supported by the repository's data. These queries are based on both the LO manifest and its usage reports, and can combine the two document types. From the XQuery point of view the database is a collection of manifest files. For each manifest file there is a nested collection containing the usage reports. Alternatively, it is possible to use a GET request with the searched fields and respective values as part of the URL query string. In the HTTP GET request the query URI uses the query parameters represented as traditional `?name=value[&...]` URL parameters. The supported names in the HTTP query string of the request are: **title** - main title of the programming exercise; **author** - name of the person who created the programming exercise; **lang** - language of the exercise description. The **value** parameter can be a partial string. Queries using the GET method are convenient for simple cases but for complex queries the programmer must resort on the use of a POST request and using XQuery as the query language.

The **Report** function associates a usage report with an existing LO. This function is invoked by the client to submit a final report, summarizing the use of an LO by a single student. This report includes general data on an attempt(s) of a student to solve a programming exercise (e.g. data, number of evaluations, success) and particular data on the characteristics of a student (e.g. gender, age, instructional level). The former is represented as a fixed set of attributes and includes the following data enumerated in Table 5.5.

²XQuery 1.0: An XML Query Language (Second Edition): <http://www.w3.org/TR/xquery/>

Table 5.5: Student's attempt general data.

Attribute	Content	Description
lo-id	URL	reference to LO
data	timestamp	data/time of usage
time	integer (seconds)	resolution time
attempts	integer	number of attempts
success	boolean	success in solving problem

A meta-model was created for representing data to characterize students. This meta-model must be abstract enough to accommodate unexpected requirements and simple enough to provide an efficient implementation. Having this in mind students are represented as a collection of attribute-values pairs, without enforcing the use of any attributes in particular. The attributes for characterizing students are not be fixed by the repository and cannot be assumed to be present (or absent). Table 5.6 shows some of these attributes.

Table 5.6: Student's characteristics particular data.

Attribute	Content	Description
gender	male female	gender of student
age	integer	age of student when (solving to problem)
country	iso-code of country	student's country of residence
language	iso-code of language	student's native language
level	integer	instruction level

For instance, with this data an LMS will be able to dynamically generate presentation orders based on previous uses of LO, instead of fixed presentation orders.

The **Alert** function notifies users of changes in the state of the repository using an RSS feed. With this option an user can have up-to-date information through a feed reader. A **minutes** parameter can be included in the request returning only the repository updates that happened in the last minutes.

The **Create** function adds new collections to the repository. In order to invoke this function in the REST interface the programmer must use the PUT request method of HTTP. The only parameter is the URL of the collection.

The **Remove** function removes an existent collection or learning object. This function uses the DELETE request method of HTTP. The only parameter is an URL identifying the collection or LO.

The **Status** function returns a general status of the repository, including versions of the

components, their capabilities and statistics.

5.3.1.2 XML binding

The responses generated by the repository are formalized by the **Response Specification Schema**. The advantage of this approach is to formalize the responses from compliant repositories facilitating the parsing and validation of the HTTP responses. Figure 5.8 depicts the elements of the new language and their types.

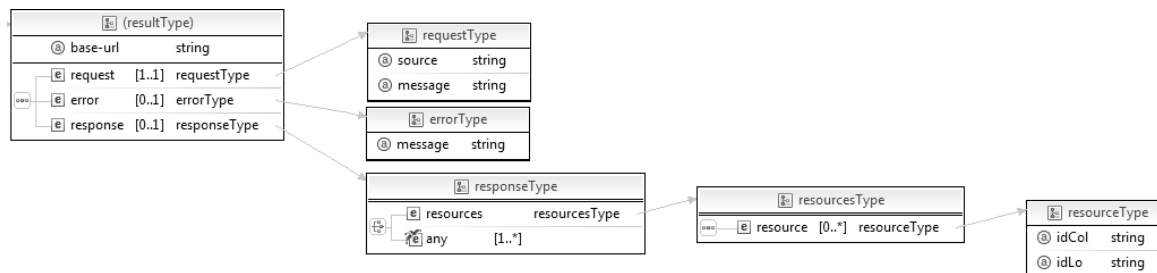


Figure 5.8: Response specification schema.

The schema has two elements: **result** and **rss**. The former is used by all the functions except the **Alert** function that returns a feed compliant with the Really Simple Syndication (RSS) 2.0 specification³. The **result** element contains the following child components:

- **base-url** attribute, defining a base URL for the relative URLs in the response;
- **request** element, containing the request URL and an human readable request message;
- **error** element, containing an error message - client systems should search for this element to verify the existence of errors;
- **response** element, describing a successful execution of the function - it is composed by an human readable response message and, for some functions, by a **resources** element that groups a set of resources defined individually in **resource** elements. A **resource** element contains an identification of the collection absolute path (attribute **idCol**) and an identification of the LO itself (attribute **idLo**).

For the description of the request and response messages please consult the appendix D.

5.3.2 Learning Tools Interoperability

The integration of the LMS with the Teaching Assistant (TA) relies on the LTI specification. The LTI specification recommends REST as the web service flavour for exchanging data

³RSS 2.0 Specification: <http://www.rssboard.org/rss-specification>

between the LMS and external tools. The LTI functions are summarized in Table 5.7.

Table 5.7: LTI functions.

Function	REST	LTI	
		Basic	Full
Launch	POST TA_URL < LTI_PARAMETERS	yes	yes
ReplaceResult	POST LIS_OUTCOMES_URL < LIS_SOURCE_ID + GRADE	no	yes
ReadResult	POST LIS_OUTCOMES_URL < LIS_SOURCE_ID > GRADE	no	yes
DeleteResult	POST LIS_OUTCOMES_URL < LIS_SOURCE_ID	no	yes

At the time of writing this dissertation the majority of the LMS do not support the full LTI specification. Thus, only the communication from the LMS to the TA is supported. The **Launch** function allows the execution of a particular external tool within the LMS. Before the launching, two steps are required: 1) the teacher (or LMS administrator) should configure the TA as an external tool in the LMS control panel by setting the name and the URL of the external tool; 2) the teacher should add an activity into the course structure referring to the external tool. Later on, when a student selects the external tool, the LMS uses the URL to launch the TA through an HTTP POST. This request includes a set of launch parameters (LTI_PARAMETERS) as hidden form fields. Table 5.8 organizes the most important parameters in four groups.

Table 5.8: LTI launch parameters

Groups	Variables	Description
Resource	resource_link_id	Unique identifier of a resource
	resource_link_title	A title for the resource
	resource_link_description	A description for the resource
User	user_id	Unique identifier of a user
	user_image	URI for an image of the user
Context	context_id	Context id of the link being launched
	context_title	A title of the context
	context_label	A label for the context
LIS	lis_person_name_full	Full name of the user
	lis_person_contact_email_primary	E-mail of the user
	lis_outcome_service_url	Unique identifier of the launch
	lis_result_sourceid	Outcomes service URL of the TC

This list can be extended by adding custom parameters. The syntax is `custom_keyname = value`. Three new parameters were added to the launch request: `custom_collection_id -`

defines a link for a collection of exercises in a DRI repository; `custom.sequencing` - defines if the exercises should be solved sequentially; `custom.time_limit` - defines a date/time limit for the solving of all exercises. Listing 5.5 shows a subset of the launch parameters that the LMS (Tool Consumer) sends to the TA (Tool Provider).

Listing 5.5: LTI launch with default parameters.

```

1 resource_link_title = Sum two vectors
2 lis_person_name_full= Pimenta Ana
3 roles = Student
4 context_title = Algorithms and Programming
5 lis_result_sourceid={"data":{"instanceid":"1","userid":"2","launchid":1914382991},"hash":"..."}
6 lis_outcome_service_url=http://crimsonhex.dcc.fc.up.pt:8080/moodle/mod/lti/service.php
7 custom_collection_id = http://crimsonhex.dcc.fc.up.pt:8080/crimsonHex/lo/myCollection/vectors
8 custom_sequencing = true
9 custom_time_limit = 2011-11-07 12:00:00

```

In this example, the TA presents the exercises of the **vectors** collection and students should solve them sequentially till November 7, 2011.

Table 5.7 also refers to three functions included in the IMS LIS Outcomes Service. These functions use the `lis_result_sourceid` parameter included in the launch request that is unique for every combination of `resource_link_id` / `user_id` parameters and identifies a unique row and column within the TC gradebook. After computing a grade, the TA calls the LTI Basic Outcomes Service using the URL stated in the `lis_outcome_service_url` launch parameter. The service supports setting, retrieving and deleting of LIS results associated with a particular user/resource combination (`lis_result_sourceid` parameter). The `replaceResultRequest` function sets a numeric grade (0.0 - 1.0) for a particular result. The `readResultRequest` function returns the current grade for a particular result. The `deleteResultRequest` function deletes the grade for a particular result.

5.3.3 Evaluation service

This subsection presents a specialization of the text file evaluation service genre (subsection 4.3.1) for the computer programming domain [LQ10h, LQF10]. This service models the evaluation of an attempt to solve a programming exercise defined as a learning object and produces a detailed report. This evaluation report includes information to support exercise assessment and grading by client systems. In this service the focus is on the automatic evaluation of programming exercises and the goal is to mark and grade exercises in computer programming courses and contests. By exposing its functions as services, an assessment system of this kind is able to participate in business processes integrating different system types such as repositories, contest management systems and learning management systems.

The remaining of this subsection focuses on the interface definition of the service and in an XML binding to formalize the responses from compliant assessment systems.

5.3.3.1 Interface definition

The definition of the interface of a service formalizes the syntax of requests and responses of its functions. This particular service exposes its functions as SOAP and REST web services. The syntax of function requests in both flavours is summarized in Table 5.9.

Table 5.9: Core functions of the Evaluation Engine.

Function	WS	Syntax
ListCapabilities	SOAP	ERL ListCapabilities()
	REST	GET /evaluate/ > ERL
Evaluate	SOAP	ERL Evaluate (LO, Attempt ,Capability, Language)
	REST	POST /evaluate/\$CID?id=LO&lang=LANG < PROG > ERL
GetReport	SOAP	ERL GetReport(Ticket)
	REST	GET \$Ticket > ERL

The **ListCapabilities** function informs the client systems of the capabilities of a particular assessment system. In a computer programming assessment system the capabilities are related with the programming language compiler or interpreter. Each capability is described by a list of features, with a name and a value. For a programming language they may be the language name (e.g. Java) its version (e.g. 1.5) and vendor (e.g. JDK). In this function the request does not accept any parameter and the response returns a list of all capabilities of the assessment system. Using the REST API this operation is performed by sending a GET HTTP request to the evaluator. The response complies with the Evaluate Response Language (ERL) specification detailed in the next sub-subsection. Table 5.10 shows an example of a request and the respective response for the **ListCapabilities** function.

The **Evaluate** function allows the request of an evaluation for an attempt to solve a specific programming exercise. The request includes a reference to an exercise represented as a LO held in a repository and a single attempt to solve a particular exercise. The request also includes a specific evaluator capability necessary for the evaluation of the attempt. The request may also include a specific ISO 639-1 language (e.g. pt for Portuguese) for the evaluation report. Internally the service implementation may include several features (indexing, queuing, transforming, flow control, etc.) needed to provide the defined functionality and a connection with a remote data source holding the objects such as a LOR. If completed within a predefined time frame, the assessment system returns an evaluation report or a ticket for a later report request.

Table 5.10: REST request and response for the **ListCapabilities** function.

Request	Response
GET http://eval.domain.org/evaluate > ERL	<pre> <message date="2001-12-31T12:00:03"> <request date="2001-12-31T12:00:03"/> <reply date="2001-12-31T12:00:05"> <capabilities> <capability id="MyService.C"> <feature name="Compiler" value="gcc"/> <feature name="Compile" value="/usr/bin/gcc -lm \"\$file"/> ... </capability> </capabilities> </reply> </message> </pre>

Using the REST API this operation is performed by sending a POST HTTP request to the server. The respective response complies with the ERL specification and includes an evaluation report or a ticket to later recover. Both request and response are depicted in the Table 5.11.

Table 5.11: REST request and response for the **Evaluate** function.

Request	Response
POST http://eval.domain.org/evaluate/java1.6? id=http://lor.domain.org/lo/123 &lang=pt < PROG > ERL	<pre> <message date="2001-12-31T12:00:03"> <request date="2001-12-31T12:00:03"> <capability="MyService.C"/> <learningObject=http://lor.domain.org/lo/123/> <program> <![CDATA[... program code here ...]]> </program> </request> <reply date="2001-12-31T12:00:05"> <token id="https://eval.domain.org/report/123"/> </reply> </message> </pre>

The HTTP request parameter **id** is a reference to a LO with the programming exercise. The **lang** attribute defines the natural language for the report. The **PROG** is an attempt to solve it. The **ERL** is the content of the HTTP response to the above request. It includes a ticket and may include an evaluation report. The **id** attribute of the **token** element can be used to recover the report on a later date.

The **GetReport** function allows a requester to retrieve a report for a specific evaluation using a ticket. The request includes a ticket previously received in response to an evaluation. The report included in this response may be transformed in the client side based on a XML stylesheet. This way the client will be able to filter out parts of the report and to calculate a classification based on its data. The evaluation report does not compute a grade, points or classification, nor produces a feedback for any particular scenario. Table 5.12 shows the HTTP request and response of the **GetReport** function.

Table 5.12: REST request and response for the **GetReport** function.

Request	Response
GET https://eval.domain.org/report/123/xpto > ERL	<pre> <message date="2001-12-31T12:00:00"> ... <report date="2001-12-31T12:00:00" evaluationServer="https://eval.domain.org/"> <capability id="MyService.C" /> <exercise href="http://lor.domain.org/lo/123"> A very simple Problem </exercise> <tests> <test executionTime="100" mode="program"> <input> <![CDATA[/home/.../tests/T1/in-1.txt]]> </input> <expectedOutput>4</expectedOutput> <obtainedOutput>4</obtainedOutput> <outputDifferances></outputDifferances> <classify>Memory Limit Exceeded</classify> <mark>0</mark> <feedback/> <environmentValues> <environmentValue name="memory" value="12kb" /> </environmentValues> </test> </tests> </report> ... </message> </pre>

5.3.3.2 XML binding

All these functions respond with an XML document complying with the Evaluation Response Language (ERL). This language is formalized in XML Schema and covers the definition of

the response messages for the three assessment system functions. The specification includes two main elements: **request** and **reply**. The former echoes the request function and its parameters as received by the evaluation service. It contains a different sub-element according to the function type. The structure of the **reply** element depicted in Figure 5.9 contains the output to that request.

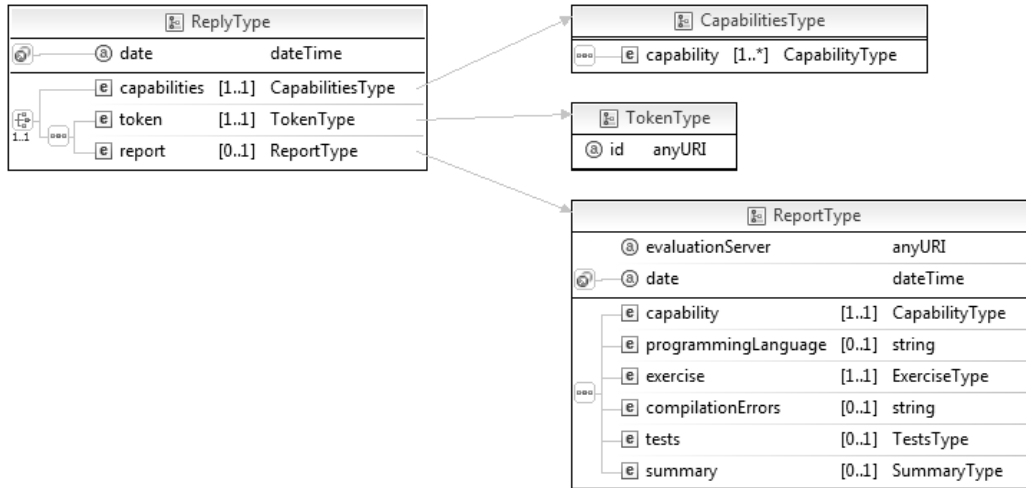


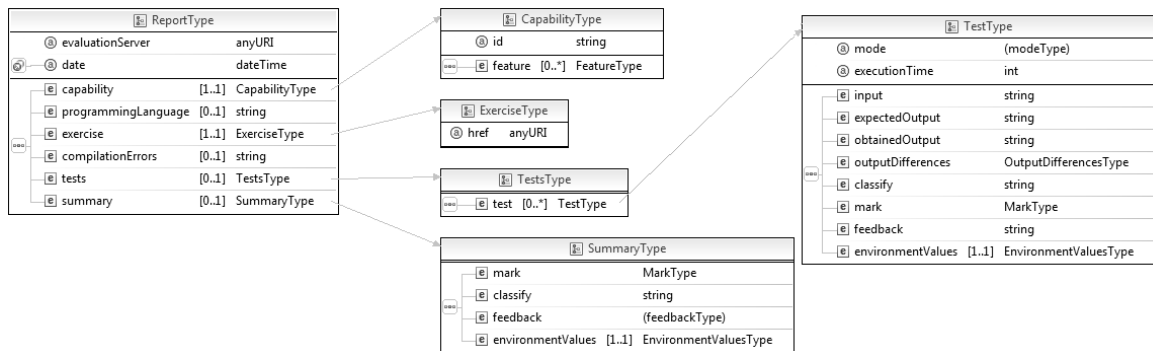
Figure 5.9: The **reply** type on the ERL specification.

The **reply** element includes the possible responses of the service such as the **capabilities** element for the **ListCapabilities** function and the **token** and **report** elements for the **Evaluate** function. The former has several **capability** sub-elements each with several **feature** elements to describe it. The latter element includes a **token** element which holds a ticket to recover a report and the optional **report** element with the effective evaluation data.

The structure of the element **report** depicted in Figure 5.10 contains the raw data sent to the client and can be used as input to other systems (e.g. classification systems, feedback systems). It has a single mandatory **evaluationServer** attribute representing the URL of the assessment system.

The **report** element includes the following sequence of sub-elements: **capability** - a specific evaluator capability used to evaluate this attempt; **programmingLanguage** - the language used to code the solution; **exercise** - a reference to the Learning Object and the title of the exercise; **compilationErrors** - compilation error messages of the code of the used; **tests** - contains a set of tests used for the evaluation of the submitted attempt. Each **test** element represents a test case describing resources supplied to evaluate the submitted program; **summary** - the synthesis of the assessment.

As shown in Figure 5.10, each test corresponds to a single test case that can be repeated to create a test set. The submitted program is executed once for each test element, receiving as

Figure 5.10: The **report** type on the ERL specification.

input the content of the **input** element. The resulting output, stored in the **obtainedOutput** element, is compared to the expected output contained in the **expectedOutput** element. The **outputDifferences** element describes the differences between the two previous elements using the syntax of the Unix **diff** command. The **test** element contains also data for grading and correcting programs. This element includes a **mark** element to assign a mark for a successful execution. The client may compute a grade for the submission as the sum of the marks of successful executions. The **feedback** element contains feedback for an unsuccessful execution. The environment values are a list of property-value pairs that may be supplied by the execution environment. For instance, if the execution environment is able to report the memory usage of a program execution then this data is recorded in this element.

5.3.4 Workflow

The previous subsections report on the efforts made to integrate the systems and services of an Ensemble instance for the computer programming domain. These efforts included the creation and extension of communication specifications such as:

- the extension of the IMS DRI specification for the integration of LOR;
- the extension of the IMS LTI specification for the integration of LMS;
- the creation of an evaluation service for the communication with the AS.

For each specification the interface definition and response bindings were presented. This subsection summarizes all the interactions among systems and services based on these specifications and represented in the UML sequence diagram depicted in Figure 5.11.

The workflow presented in Figure 5.11 starts by the selection of an LTI activity by the student. This activity was previously configured by the teacher by selecting a collection of exercises. After selection, the LMS launches the TA through the **Launch** function of

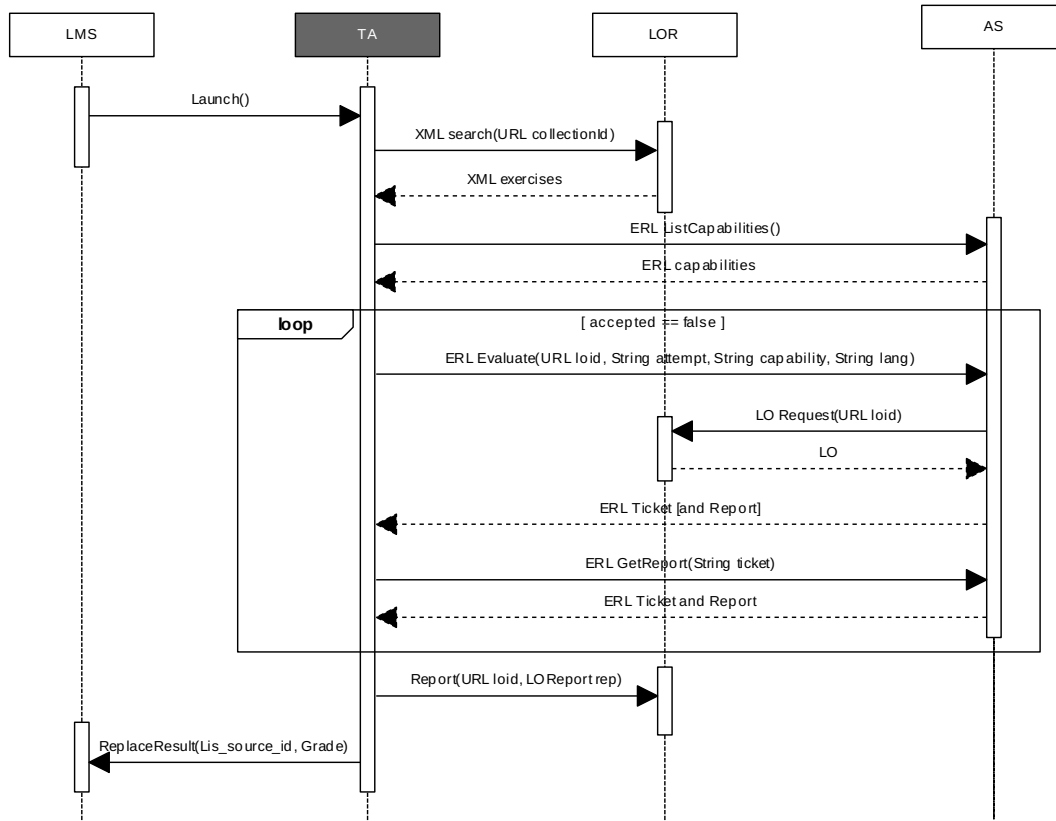


Figure 5.11: Sequence diagram of the Ensemble instance.

the LTI specification passing the user and course parameters. In order to present the programming exercises to the student the TA forwards the search to the LOR using the **Search** DRI function with the given collection URL. Meanwhile the TA gets from the AS the evaluation capabilities for later evaluation using the **ListCapabilities** function. In general each student is able to make several submissions for the same exercise and an activity may include several exercises. Each evaluation starts with an **Evaluate** request from the TA to the AS, sending a program and referring an exercise and a programming language. The AS retrieves the LO from the repository to have access to test cases, special correctors and other metadata. The AS responds with a ticket and an evaluation report, if the evaluation is completed within a certain time frame. The TA may retrieve the evaluation report using the **GetReport** function with the ticket as argument. When the student ends the session, the TA sends an usage report to the LOR using the DRI **report** function and a grade back to the LMS using the LTI **replaceResult** function.

5.4 Tools selection

The data and integration model of this Ensemble instance relies on content and communication standards as recommended by the Ensemble specification. These interoperability effort abstracts specific systems and focus on system types. This approach has facilitated the selection of tools for deployment purposes. The next paragraphs discuss the selection of each type of system or service. Figure 5.12 shows the tools selected for networks using this Ensemble instance.

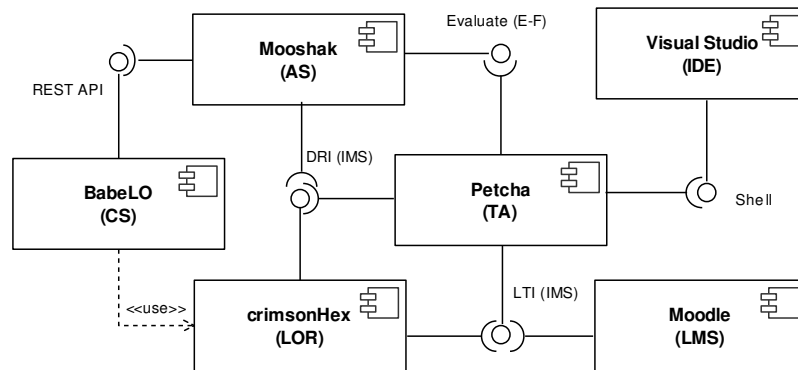


Figure 5.12: The deployment architecture of the EeF.

On the LMS side the choice fell on Moodle since it is a popular and open source LMS, arguably the most popular LMS nowadays [DW09, CF07]. This LMS has made efforts to support interoperability with other e-learning systems at two levels: content (e.g. IMS CP, SCORM, IMS CC) and communication (e.g. IMS LTI). Also successfully tests were made with Sakai LMS on this network evidencing the interoperable characteristics of the proposed approach.

The LOR system selected was CrimsonHex (see chapter 6) - a software for the creation of repositories of programming exercises. The exercises are described as learning objects and complying with the IMS CC specification. The repository also adheres to the IMS DRI specification to communicate with other systems. Other software for repositories were analysed (e.g. Flori, HarvestRoad Hive, IntraLibrary) but none of them met the domain requirements for the content and communication interoperability and most of them follow a commercial development model.

The AS system selected was Mooshak [LS03]. Mooshak is an open source system for managing programming contests on the Web including automatic judging of submitted programs. This was the logical choice after the survey (section 2.4) made to 15 assessment systems. One of the most important reasons for its selection was the support of web services.

The IDE system selected was Visual Studio Express for C# assignments. Successful tests

were made also with the Eclipse IDE for JAVA assignments on this network.

The TA system selected was Petcha (see chapter 8). Petcha has a two-fold goal: to coordinate the systems and services of this network and to interface with users, both teachers and students. Given the specificity of this role no other similar systems were found.

The CS system selected was BabeLO (see chapter 7). This system converts formats of programming exercises among systems. At the time of writing this dissertation no other system was found with these characteristics.

5.5 Summary

This chapter presents a specialization of the abstract framework detailed in the previous chapter for a new domain - computer programming. The presentation includes the overall architecture of the Ensemble instance followed by the description of the data and the integration models. The data model relies on the PExIL specification - an interoperability language for describing programming exercises - included as a LAO resource in an IMS CC package as recommended by the Ensemble specification. The integration model detailed how systems and services of this Ensemble instance are connected through the extension of the recommended specifications of the EeF. The integration model includes the creation of a new service for the communication with the AS based on the Evaluate service genre and the extension of the IMS DRI and LTI specifications for the integration of LOR and LMS, respectively. Then, a workflow of a network based on this Ensemble instance was presented showing how systems and services interact. Finally, a selection of tools adjusted to the models was presented. This process was straightforward evidencing the interoperability features of the framework. Three of these systems and services that integrate this network were created from scratch and will be detailed in the following chapters, more precisely, the crimsonHex repository, the BabeLO converter and the Petcha Teaching Assistant. The acceptability of this Ensemble instance is validated in chapter 9 through an experiment in a pedagogical environment.

Part III

Implementation

Chapter 6

Learning objects repository

*"They were urged on by the delirium of trying to reach the books in the
Crimson Hexagon: books whose format is smaller than usual,
all-powerful, illustrated and magical."*

Jorge Luis Borges in "The Library of Babel"

This chapter presents the design and implementation of a service oriented repository of learning objects (LOs) called crimsonHex. This repository supports new definitions of learning objects for specialized domains and this feature is illustrated with the definition of programming exercises as learning objects and its validation by the repository. The repository is also fully compliant with existing communication standards and extensions were made by adding new functions, formalizing message interchange and providing a REST interface. To validate the interoperability features of the repository a plug-in for Moodle was developed.

6.1 Architecture

The repository was modelled using a book library as metaphor: LOs may be seen as books that the learner (the reader) accesses through the LMS. As in a book library, the repository has a catalogue that provides efficient search. After selecting a LO the learner receives an actual copy of it, as supplied by its author, not just a pointer to another service. As LOs are a kind of e-books the repository will have an unlimited number of copies to lent, unlike in a physical book library. Libraries also keep records of requisitions. It is therefore very simple for a librarian to know which are the most popular books, those that were never

requested and the books that readers take longer to read. This kind of information will be interesting for the next generation of e-learning systems. Instead of using fixed presentation orders of LOs, as they do today, they will dynamically determine presentation orders based on the information about their previous use. To accommodate these features the repository provides the option to record information on the use of the LO and provides statistics on this data. Based on these thoughts the following design goals were defined:

1. The repository must be simple and effective. Simplicity is the best way to promote the reliability and efficiency of the repository. This principle led to identify a core component with a minimal set of features efficiently implemented. Complementary features are delegated to helper applications that connect to the core component;
2. The repository must be reusable in other contexts, with generic features available through users or applications interfaces;
3. The repository must comply existing standards for content (e.g. IEEE LOM, IMS CP, SCORM, IMS CC) and for communication (IMS DRI).

The architecture of crimsonHex is depicted in Figure 6.1.

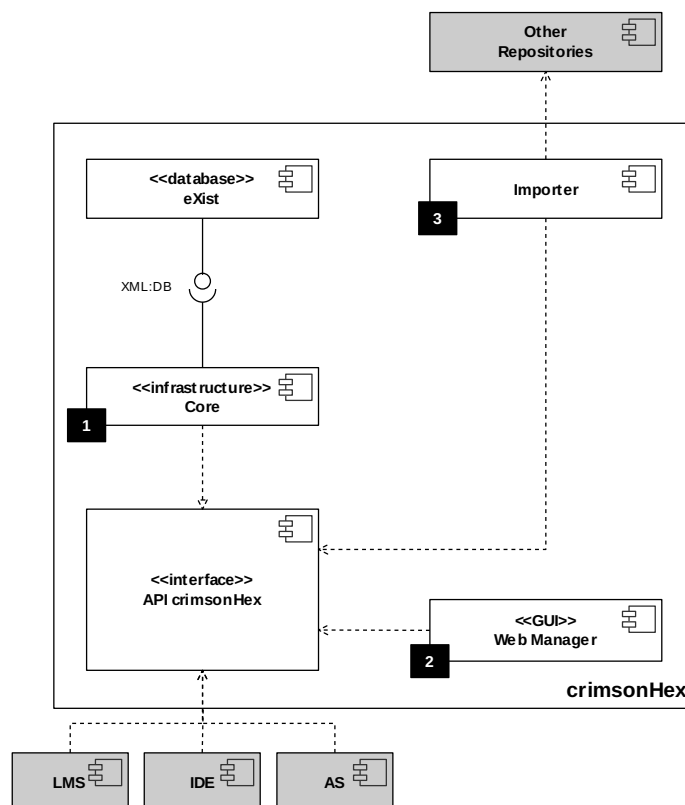


Figure 6.1: UML components diagram of the crimsonHex repository.

The crimsonHex repository includes the following components:

1. **the Core** exposes the main features of the repository, both to external services, such as the LMS and the AS, and to internal components - the Web Manager and the Importer;
2. **the Web Manager** allows the searching, previewing, uploading and downloading of LOs and related usage data;
3. **the Importer** populates the repository with content from existing legacy repositories, while converting it to LOs.

The core operations of the repository are uploading and downloading LO - ZIP archives - which are inherently simple operations that can be implemented almost directly over the transport protocol. Other features may need a more elaborate implementation but do not require the same reliability and efficiency of the core features. These features are relegated to auxiliary components, connected to the central component using the crimsonHex API. Other e-learning systems can be plugged into the repository also using this API.

The crimsonHex interoperability is based on two models: data and communication.

The **data model** of crimsonHex relies on the representation of programming exercises as learning objects. A LO containing a programming exercise must include metadata to allow its use by different types of specialized e-learning services. The existing LO standards are insufficient for that purpose, which led to the development of a new language to describe programming exercises called PExIL. This language is presented in detail in subsection 5.2.2. A PExIL descriptor is included in an IMS CC package as a LAO resource. This is the proper place to include new extensions to the manifest since a LAO resource comprises several files connected through a single descriptor.

The **communication model** of the repository defines the interaction between the repository and the other e-learning systems. The model relies on an API used both internally and externally. Internally the API links the main components of the repository. Externally the API exposes the functions of the repository to third party systems. In order to promote the integration with other e-learning systems, the API of the repository adheres and extends the IMS DRI specification. The specification recommends a set of functions introduced in subsection 5.3.1 and detailed in appendix D.

6.2 Implementation details

This section details the design and implementation of the Core component of crimsonHex on the Tomcat servlet container. The following subsections detail the development of the

four main facets of the Core - storage, validation, interface and security.

6.2.1 Storage

Searching LOs in the repository is based on queries on their XML manifests. Since manifests are XML documents with complex schemata two types of databases systems with XML support were studied: XML enabled relational databases and Native XML Databases (NXD). XML enabled relational databases are traditional databases with XML import/export features. They do not internally store data in XML format hence they need extra processing to support querying using XQuery. Since queries in this standard are a DRI recommendation, this type of storage is not the best option. In contrast, NXD uses the XML document as fundamental unit of (logical) storage, making it more suitable for data schemata difficult to fit in the relational model. Moreover, using XML documents as storage units enables the following standards: XPath for simple queries on document or collections of documents; XQuery for queries requiring transformational scaffolding; SOAP, REST, WebDAV, XmlRpc and Atom for application interface; XML:DB API (or XAPI) as a standard interface to access XML datastores; XSLT to transform documents or query-results retrieved from the database.

Several open source NXD, including SEDNA, OZONE, XIndice and eXist were analysed. Only eXist implements the complete list of the features enumerated above, which led to its select as the storage component of crimsonHex. It has also two important features that worth mentioning: support for collections, to structure the database in groups of related documents and automatic indexes to speed up the database access [Mei02].

6.2.2 Validation

CrimsonHex is a repository of specialized learning objects. To support this multi typed content the repository must have a flexible LO metadata validation feature. The eXist NXD supports implicit validation on insertion of XML documents in the database but this feature could not be used for several reasons: LO are not XML documents (are ZIP files containing an XML manifest); manifest validation may involve many XML Schema Definition (XSD) files that are not efficiently handled by eXist; and manifest validation may combine XSD and Schematron validation and this last is not fully supported by eXist.

All LOs stored in crimsonHex must comply with the IMS CC that specifies its structure and content. This specification also requires the XSD validation of their manifests. For particular domains it is possible to configure specialized validations in crimsonHex by supplying a Java class implementing a specific interface. These validations extend those of the IMS CC and may introduce new schemata, even using different type definition languages such as Schematron.

Validations are configured for each collection of documents. Thus, different types of specialized LO may coexist in a single instance of crimsonHex. As mentioned before, IMS CC main schema imports many other schemata (more than 30) that must be downloaded from the Internet. This requirement has a bad impact on the performance of the `submit` function. To accelerate this function a cache was implemented. A newly stored schema has a time to live of 1 hour. Outdated schemata are reloaded from their original Internet location using a conditional HTTP request that downloads it only if it has effectively changed.

Despite the expressiveness of XML Schema, there are several situations where it is not possible to validate a document with only this language. One example is the values dependence where it is important to guarantee that a value is higher/lower than another (e.g. value of the `imsmd:minimumversion` element is less than the value of the `imsmd:maximumversion` element).

To check this type of constraints a XML Schema cannot be used. There are, at least, three options: combine with others schema languages; write code in a programming language to express the additional constraints; use an XSLT/XPath stylesheet. The first is the chosen one since in order to maintain the solutions based in XML technologies and, if possible, in a single schema document. There are several alternative schema languages, such as RELAX, TREX and Schematron. A good candidate to this “second level of validation” is Schematron. The previous example could be validated as a separate file with the rule included in Listing 6.1:

Listing 6.1: Example of a Schematron rule.

```

1 <schema xmlns="http://www.ascc.net/xml/schematron" >
2   <pattern name="version validation" >
3     <rule context="//imsmd:requirement" >
4       <assert test="imsmd:minimumversion <= imsmd:maximumversion">ERROR</assert>
5     </rule>
6   </pattern>
7 </schema>

```

Schematron validation can be used in conjunction with a XML schema validation using two approaches: as separate files using pipeline validation languages (e.g. DSDL, Schemachine) or as a unique file embedding Schematron rules in the XML Schema. In order to simplify the file version management the second option was selected and the Schematron rules were embedded within the `appinfo` elements in the XSD document. However, a W3C XML Schema processor does not validate constraints expressed by the embedded Schematron rules. They need to be extracted from the source schema and concatenated into a new Schematron document. To address this issue a stylesheet (`Schematron-Generator.xsl`) was created to extract embedded Schematron rules from a W3C XML Schema document and merge them into a complete schema. This approach is depicted in Figure 6.2.

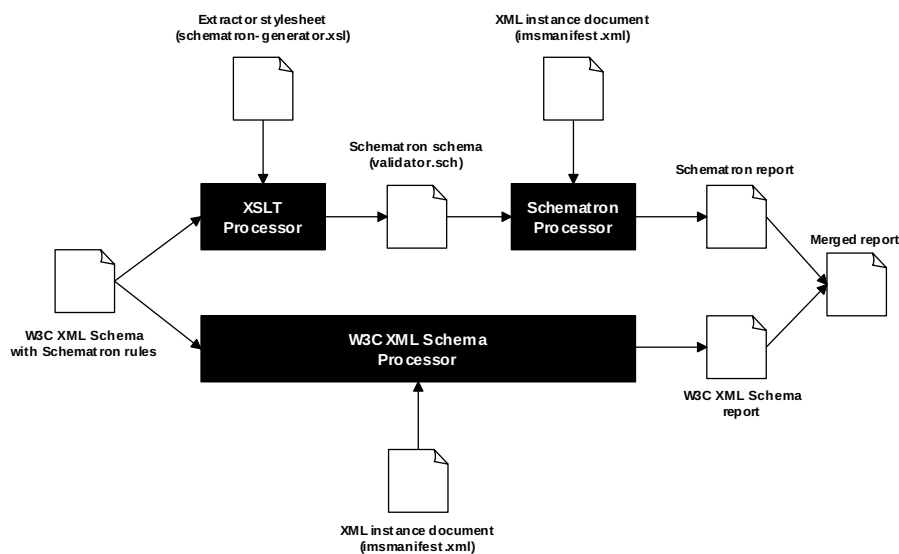


Figure 6.2: Validation of XML files by a W3C XML Schema with Schematron rules [Rob02].

Since Schematron rules are built using XPath and XSLT functions, the Schematron processor is based on a XSLT processor. To perform this validation an implementation of a Java API for the Schematron language¹ was used that organizes the Schematron processing in two steps, as shown in Figure 6.3.

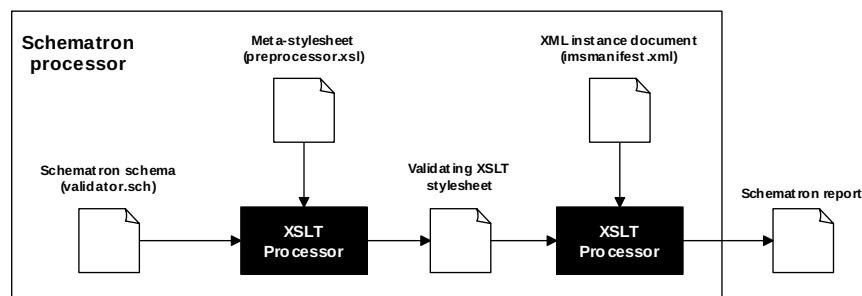


Figure 6.3: Schematron processing [Rob02].

The Schematron schema is transformed into a validating XSLT stylesheet by a meta-stylesheet provided by the API. The validating stylesheet is then used on the XML instance document and the result will be a report based on rules/assertions of the Schematron schema. By combining these two validation languages, many of the constraints that previously had to be checked in the application code can now be abstracted to the schema. However it should be noticed that in time critical applications the overhead of processing the embedded Schematron rules may be unaffordable.

¹Official Web site: <http://www2.informatik.hu-berlin.de/~obecker/SchematronAPI/>

6.2.3 Security

Following the design principles of simplicity and efficiency the management of users and access control in the Core was avoided. This decision does not preclude the security of this component since these features can be controlled in the communication layer. Since both web services flavours use HTTP as transport protocol the channel can be secured using Secure Sockets Layer (SSL) (i.e. HTTPS). This ensures the integrity and confidentiality of assets in LO. The authentication and authorization rely on the verification of client certificates provided by SSL. In practice, to implement this approach it is necessary to configure the servlet container (e.g. Tomcat) to support HTTPS requests with authorized certificates. Nevertheless, managing certificates is a comparatively complex procedure thus a set of auxiliary functions is provided in the core that act as a mini Certificate Authority (CA). These functions are used for managing and signing client certificates and their implementation is based on the Java Security APIs.

6.2.4 User Interface

This subsection presents the crimsonHex user interface. Firstly the strategy to design the user interface is presented. Then, the main tasks, namely browsing, authoring and searching are described. The design of this user interface (Figure 6.4) was based on the identification of task and usage profiles, task objects and task actions. The task profiles are:

Archivist - a person responsible for a set of activities related with the collection management, such as: creation of collections, assigning of learners and reviewers to collections;

Author - a person that develops and submits LO to the repository. The submission of LO will be enforced to comply with controlled vocabularies defined in meta-data standards (IEEE Learning Object Metadata - LOM) and possible extensions. This class of users will contribute with new learning objects and receive peer reviews from specialists;

Reviewer - a person that controls the quality of the repository by validating the submitted LO;

Consumer - a person that browses (part) of the repository and has limited access to its content (LO, usage reports, reviews, comments).

Users will have different usage profiles. On one hand, many will be novice or first-time users, especially among authors and consumers. On the other hand, some users, especially reviewers and archivists, will use crimsonHex frequently, tending to become experts in its use. After the identification of users and usage profiles the tasks they need to perform on this interface were identified. The LO and collections of LOs are task objects, each

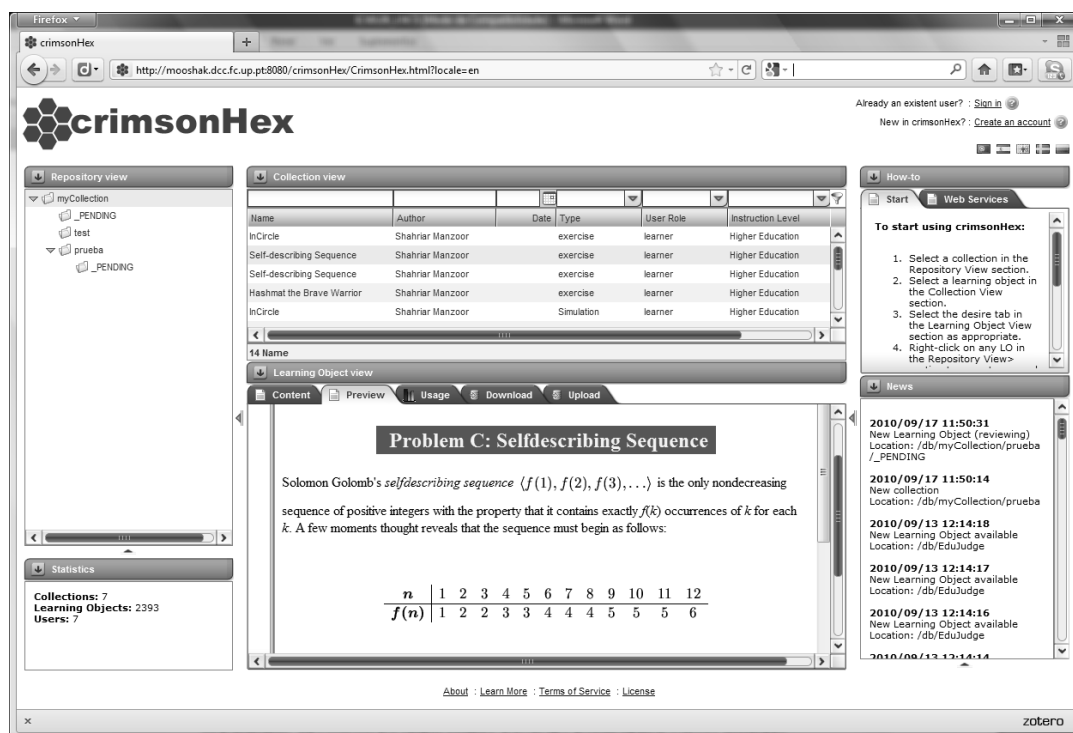


Figure 6.4: crimsonHex WebManager.

with a number of associated task actions, depending on user profiles. Task actions over LO include: viewing, reviewing, downloading, and commenting. Task actions on collections include creating, removing and authoring/uploading (LO to that collection). Based on the previous identifications a screen layout was defined - a single screen (Figure 6.4) with specific areas for task object selection and task actions. Task object selection is needed by all users, although the selectable content depends on the user's profile, thus it can be implemented by a common tree-based control. Different task actions require specific forms or panels that also share a common control on the user interface. Since the number task actions is comparatively small a tabbed control was chosen to aggregate them. The tab configuration shown to users depend both on their profile and on the current task object selection. As a rule, all available task actions have an associated tab, thus helping novice users to recognise which are the available actions. However, some of these task actions can be executed directly over selected task objects, without requiring additional data. In general, these task actions are meant for frequent users and will be bound to contextual menus on the tree-control, as well as to accelerator keys.

Figure 6.4 shows the user interface layout of the repository with two main areas: selection on the left side and action on middle. In the selection area the user navigates through the repository structure to select task objects. In the action area the user executes task actions on the selected task objects. Secondary areas in this layout are the header, used

for authentication and registration, and the right side, used for news and statistics. The remainder of this section details the design of task actions available in the main areas. Selected LO can be viewed in different perspectives, such as, Content, Usage and Review. As would be expected, each perspective is assigned to a different tab on the action area.

The Content tab shows the resources and meta-data of the selected LO using XSLT transformations on its manifest file. Different stylesheets can be select to configure content presentation, including:

Resources – lists resources and their meta-data with support for viewing/downloading individual resources;

Meta-data – shows all global meta-data, including LOM and extensions;

LOM - show LOM meta-data grouped in main categories (General, Lifecycle, Technical, Educational, Rights);

Extension – shows meta-data related with the extension schemata.

The Usage tab presents statistics on the use of a selected LO. This feature is related to the report function that associates usage reports to an existing LO. This function is invoked by a consumer of the repository services (typically an LMS), summarising an episode of using a LO with a particular student. The aim of this function is to provide the LMS with the ability to dynamically generate presentation orders based on previous uses of LO, instead of using fixed presentation orders. This report includes both general data on the student's attempt to solve the programming exercise (e.g. data, number of evaluations, success) and particular data on the student's characteristics (e.g. gender, age, instructional level).

The Review tab assists in the review process of a LO. Before validation the LO is not available to general users of the repository. The availability of the LO depends on this validation. If the LO is not accepted the reviewer could justify the rejection and/or supply comments to the author of the LO. These comments may lead to new versions that must be submitted as a new LO.

The Web Manager component was developed using an Ajax framework called Google Web Toolkit (GWT) to enable the implementation of the single screen design resulting from the last section. GWT is an open source Java software development framework that allows a rapid development of AJAX applications in Java. When the application is deployed, the GWT cross-compiler translates Java classes of the GUI to JavaScript files and guarantees cross-browser portability. The framework also supports asynchronous remote procedure calls. This way, tasks that require significant computational resources (e.g., complex searching within the repository) can be triggered asynchronously, increasing the user interface's

responsiveness. The complex controls required by the selection and action areas are provided by SmartGWT, a GWT API's for SmartClient, a Rich Internet Application (RIA) system.

The Web Manager component is organised in two main packages: the back-end (server) and the front-end (client). The back-end includes all the service implementations triggered by the user interface. These implementations rely on the gateway class for managing the communication with the Core of the repository. A single class implementing the Gateway design pattern concentrates the interaction with the core component. To interact with other DRI compliant repositories only this class will have to be re-implemented.

6.2.5 Tests

Reliability is one of the main concerns regarding the Core component of crimsonHex. The JUnit² was adopted as the automated unit testing framework since crimsonHex is implemented in Java and this tool is support by Eclipse, the Integrated Development Environment (IDE) used in this project. Apart from the unit tests, a tool was created for automatic generation of random requests to the repository, following the communication model. The goal of this tool is two folded: to look for bugs in unpredicted sequences of requests and to stress-test the repository. The tool generates a random sequence of Core functions' invocations and records then in the Core's log file (through a Java-based logging utility called log4j³). Errors generated by these request sequences are recorded by the Core in the same log files. After each test the log file is manually inspected looking for function sequences that originated errors. This approach was essential to discover errors that otherwise would only be detected in production. Efficiency and scalability are two other main concerns in the development of crimsonHex. In order to test performance the test tool was used to compare execution times of the main functions in the two supported web services interfaces: SOAP and REST. For the experiment the same PC was used for client and server purposes. The PC was an Asus M70VSeries with Windows Vista Home Premium (32 bits), Intel(R) Core(TM)2 Duo P8400 @2.26GHz and 4GB RAM. Were used 100 LOs on the experiment ranging in storage size from 2MB to 5MB. Each function has been repeated 10 times. Average function execution times for the set of functions are shown in Table 6.1.

Table 6.1: Average function execution times per interface (in seconds).

	Submit	Retrieve	Search
SOAP	4.53	1.57	2.23
REST	2.11	0.44	0.93

These figures show that the DRI extension, based on REST, is twice as efficient as the

²Official Web site: <http://www.junit.org/>

³Official Web site: <http://logging.apache.org/log4j/1.2/>

standard SOAP interface. These results were expected since the REST interface does not have to marshal request messages. In both interfaces submit times are significantly higher than the other functions due to the cost of the validation process. Scalability has other important issue. Scalability is bound by the database limits. The eXist NXD supports a maximum of 2^{31} documents and theoretically, documents can be arbitrary large depending on relevant file system limits, e.g. the max size of a file in the file system. To test the scalability of eXist some queries were made with ever increasing data volumes. The experiment shows linear scalability of eXist's indexing, storage and querying architecture.

6.3 Case Study: using crimsonHex as a LMS plug-in

This section presents the creation of a plug-in for the crimsonHex repository. To evaluate the interoperability features of the crimsonHex repository this plug-in is integrated with Moodle, arguably the most popular LMS nowadays. The development of this plug-in was straightforward. In terms of programming effort half a day was spent to produce approximately 100 new lines of code. This quick and simple integration benefited from the new interoperability features of the repository. Currently, Moodle 2.2 includes support for different types of repositories. Several APIs are available to enable the development of plug-ins by third parties, including:

File API for managing internal repositories;

Repository API for browsing and retrieving files from external repositories;

Portfolio API for exporting Moodle content to external repositories.

The Repository API was chosen for testing the integration features of the crimsonHex repository in Moodle. The goal of this particular API is to support the development of plug-ins to import content from external repositories. The Repository API is organized in two parts: Administration, for administrators to configure their repositories and File picker, for teachers to interact with the available repositories. To create a plug-in for Moodle using the Repository API one must implement a set of related files. For instance, the steps to create the crimsonHex plug-in for Moodle are the following:

1. to create a folder for the plug-in (moodle/repository/crimsonHex);
2. to add to the plug-in folder the files `repository.class.php` – sub-classing a standard API class and overriding its default methods and `icon.png` – providing the icon displayed in the file picker;

3. to create the language file `repository_crimsonHex.php` and add it to the folder `moodle/repository/lang/en_utf8/`.

The `repository.class.php` is responsible for handling the communication between Moodle and all repository servers of that type. In this case the repository type is crimsonHex but other types are being developed for other types of repository, such as Merlot, YouTube, Flickr and DSpace. For Moodle, each repository is just a hierarchy of nodes. This allows Moodle to construct a standard browse interface. The repository server must provide: a URI to download each node (e.g. a LO) and a list of the nodes (e.g. LO and collections) under a given node (e.g. collection). In addition to these requirements, a repository can optionally support authentication, provide additional metadata for each node (mime type, size, dates, related files, etc.), describe a search facility or even provide copyright and usage rules.

As explained before, the Repository API has two parts – Administration and File Picker – each with its own graphical user interface (GUI). The Figure 6.5 shows the file picker GUI of the crimsonHex plug-in that will be used by the teacher to pick up the suitable exercises for the class.

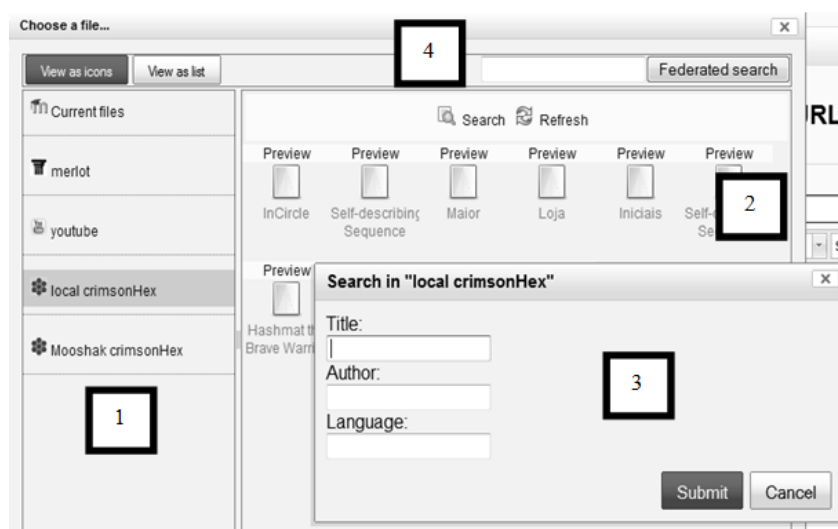


Figure 6.5: crimsonHex plugin interface.

In the left panel are listed the available repositories as defined by the administrator. Two crimsonHex repository instances are marked with label 1. Label 2 marks the default listing of the selected repository. Pressing the “Preview” link presents a preview of the respective LO. Pressing the “Search” link pops-up a simple search form, marked as 3 in Figure 6.5. Federated search in all available crimsonHex repositories uses the text box marked as 4.

Each feature of the plug-in is implemented by a method in the `repository.class.php` file. A typical method includes: a repository invocation (SOAP or REST), the parsing of

its response (using the PHP `simplexml_load_string` function to parse the XML data), a selection of the pertinent data (using XPath) and an iteration over the new results (for instance, populating an array with the relevant data). Listing 6.2 shows an excerpt of the overridden `search` function.

Listing 6.2: Example of the search function of the crimsonHex plugin.

```

1 private function _search($queryString) {
2     $list = array();
3     $c = new curl();
4     $content=$c->get($this->options['url'] . $queryString);
5     $xml = simplexml_load_string($content);
6     $result = $xml->xpath("//resource");
7     foreach ($result as $entry) {
8         $attr = $entry->attributes();
9         $list[] = array(
10             'title'=>(string)$entry,
11             'thumbnail'=>$OUTPUT->icon_url(path),
12             'date'=>'',
13             'size'=>'',
14             'source'=>$attr['url'].$attr['idCol']
15             .$attr['idLo']);
16     } return $list; }

```

6.4 Summary

This chapter presents the design and implementation of a service oriented repository of learning objects called crimsonHex. This repository supports the definition of programming exercises as learning objects complying several specifications (IMS CC). The repository is also fully compliant with existing communication standards (IMS DRI) and extensions were made by adding new functions, formalizing message interchange and providing a REST interface. To validate the interoperability features of the repository a plug-in for Moodle was developed that is expected to be included in its next release (version 2.3) of this LMS.

The improved interoperability of crimsonHex is expected to support the development of new e-learning tools requiring greater integration with repositories. The repository plug-in will facilitate the use of crimsonHex by Moodle users. In its current status crimsonHex is available for download at the following URL: <http://ensemble.dcc.fc.up.pt/crimsonHex>. The plug-in for accessing crimsonHex repositories from Moodle is also available for download at the following URL: <http://ensemble.dcc.fc.up.pt/ChMoodlePlugin>.

Chapter 7

Programming exercises converter

"And the whole earth was of one language, and of one speech"

Genesis 11:1

In the last two decades there was a proliferation of programming exercise formats. The study included in section 3.2.3.2 confirms the disparity of programming exercise formats highlighting both their differences and their similar features. This heterogeneity hinders the interoperability among the typical systems found on the automatic evaluation of exercises. Rather than attempting to harmonize the various specifications, a pragmatic solution is to provide a service for exercise format conversion.

BabeLO is a programming exercise converter providing services to a network of heterogeneous e-learning systems such as contest management systems, programming exercise authoring tools, evaluation engines and repositories of learning objects. Its main feature is the use of a pivotal format to achieve greater extensibility. This approach simplifies the extension to other formats, just requiring the conversion to and from the pivotal format. This chapter starts by presenting this approach and the pivotal data format used. Then, the abstract service definition, its components and web service interface are presented. Finally, a report on the use of BabeLO in two concrete scenarios is included: to relocate exercises to a different repository and to use an assessment system in a network of heterogeneous systems.

7.1 Exercise format conversion

Data conversion is the conversion of computer data from one format to another. Data conversion can typically occur based on two approaches: **direct** or **pivotal** [Tsu10].

In the direct conversion the converter receives the input format and apply transformations according to the output format. One good example is the transcoder¹ developed by JISC Centre for Educational Technology and Interoperability Standards (JISC CETIS) that enables the conversion between e-learning content packages. It addresses conversions between the most common e-learning content formats such as IMS CP, SCORM and IMS CC.

The pivotal conversion is based on an intermediate format allowing any source format to be converted to its target. Compared with the previous approach, this pivotal encoding approach provides several advantages such as manageability. A data format converter would have to support a huge number of mappings for all the permutations of the data formats supported. Using a intermediate format scales down this number since only one mapping is needed for each format supported. Pivotal conversion is often used in several areas. For instance, Office applications use the OpenDocument file format as a pivot for the conversion between office file formats. Despite its use, the pivotal approach is also subject of criticism such as the augment of noise due to the propagation of the translation errors and inaccuracy or lost of data due to the conversion between formats that are conceptually different [Tsu10].

7.1.1 Approach

Based on the current conversion approaches the choice was on the use of the pivotal approach for the exercise formats conversion. As depicted in Figure 7.1 using a pivot format reduces

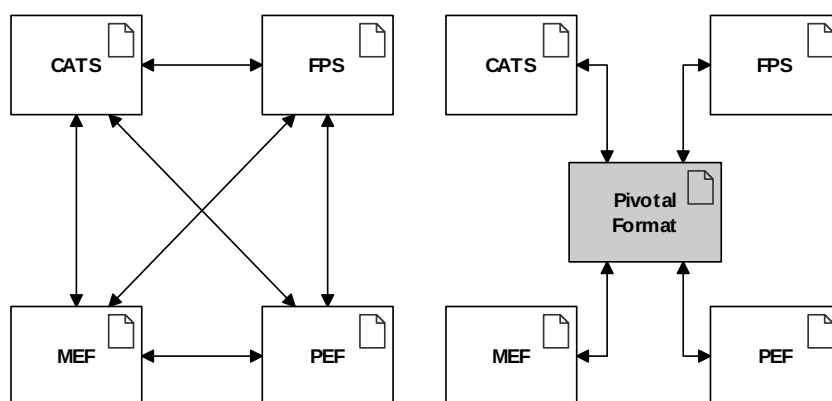


Figure 7.1: Exercise formats conversion using both approaches.

drastically the number of permutations needed from $n \times (n - 1)$ to $n \times 2$, where n is the number of formats supported. Since one of the design requirements of the converter service is its extensibility it is important to simplify the support for new formats.

¹<http://purl.oclc.org/NET/transcoder>

7.1.2 Pivot format

PExIL is a XML dialect that aims to consolidate all the data required in the programming exercise life-cycle [QL11e]. The expressiveness of PExIL was validated according to the multi-facet model proposed by Verhoeff. Table 7.1 shows the PExIL elements coverage.

Table 7.1: PExIL coverage based on the Verhoeff model.

Verhoeff / PExIL	G1	G2	G3
Text	X	X	-
Data files	X	X	X
Parameters	-	-	X
Tools	-	-	X
Metadata	X	-	X

Textual elements group (G1) in PExIL can be used in several phases of the programming exercise life-cycle: in the selection phase as exercise metadata to aid discoverability and to facilitate the interoperability among systems (e.g. LMS, IDE); in the presentation phase as content to be present to the learner (e.g. exercise description); in the resolution phase as skeleton code to be included in the student's project solution.

Specification elements group (G2) in PExIL can be used in several phases of the programming exercise life-cycle: by 1) the content author to automatically generate an input and output test example to be included on the exercise description for presentation purposes; 2) the learner to automatically generate new test cases to validate his attempt; 3) the Evaluation Engine to evaluate a submission using the test cases.

Program elements group (G3) contain references to program source files as external resources (e.g. solution program, correctors) and metadata about those resources (e.g. compilation, execution line, hints). These resources are used mostly in the evaluation phase of the programming exercise life-cycle to allow the Evaluation Engine to produce an evaluation report of a students' attempt to solve an exercise.

This analysis asserts the total coverage of PExIL elements based on the Verhoeff model and guarantees PExIL as a good candidate to act as a pivot format for a conversion service of programming exercises formats. More details about PExIL can be found in subsection 5.2.2.

7.1.3 Abstract functions

This section presents the generic capabilities of a converter service expressed in terms of their behaviours, without prescribing how to make them operational. A service of this genre

is responsible for the conversion of programming exercises formats. It supports the following functions:

- The **GetFormats** function provides the requester with a list of all the formats supported by the service. In order to support a format the service must implement the format conversion from and to the pivotal format. In this function, the request may not have parameters or have one representing the input format. In the former the response returns a list of all formats of the converter. In the latter the response includes only the formats that can be converted from the input format given as a parameter. In the response, each format is described by its name and a list of formats that can be used as outputs and its corresponding URL paths. This will allow client systems to automate the conversion request based on the available formats returned by this function.
- The **Convert** function performs the conversion of a given programming exercise from an input format to an output format. The function includes three parameters: the format of the exercise to convert, the conversion output format and a reference (URL based) of the exercise to convert. The function returns an archive with its contents complying the output format.
- The **ConvertSets** function converts a set of programming exercises from an input format to an output format. This function is especially useful since one may not always want to convert just an exercise. In a more competitive environment a contest manager may want to feed a new contest based on a problem set of a existing programming contest. In a more pedagogical context a teacher may want to use in the classroom a set of problems from an external source. In both cases the request parameters are similar to the previous function. The function returns an archive with a set of exercises complying the given output format.
- The **ConvertFromSet** function converts a single programming exercise from a set of exercises described with the input format to a single exercise in the output format. The exercise to convert is given by its position within the set as an additional parameter. The function returns an archive with a single exercise complying the output format.
- The **ConvertToSet** function converts a single programming exercise complying with the input format to a single collection in the output format. The function returns an archive containing a collection with only one exercise in the given output format.

7.2 The BabeLO service

BabeLO is a service for the conversion between different programming exercises formats. Figure 7.2 shows the architecture of BabeLO described by the UML component diagram.

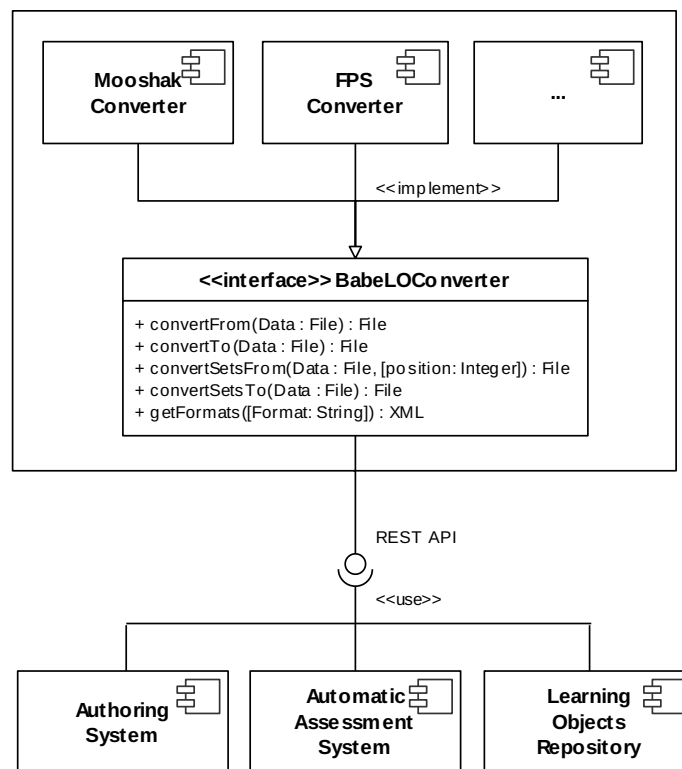


Figure 7.2: BabeLO architecture.

In order to allow client systems to use the conversion features of BabeLO a set of core functions is exposed as services using a REST web service interface. The REST implementation uses Jersey² - the reference implementation of JAX-RS (The Java API for RESTful Web Services). In Table 7.2 each function is associated with the corresponding operation in the REST flavour.

Table 7.2: BabeLO REST API.

Functions	REST API
GetFormats	/convert[/inFormat] > BRL
Convert	convert/inFormat/outFormat/refEx > Archive
ConvertSets	convertsets/inFormat/outFormat/refEx > Archive
ConvertToSet	convertttoSet/inFormat/outFormat/refEx > Archive
ConvertFromSet	convertfromset/inFormat/outFormat/position/refEx > Archive

The **GetFormats** function returns a list of the formats supported by the service. The response is formalized using the BabeLO Response Language (BRL). Listing 7.1 shows the correspondent response of the above request.

²Official Web site: <http://jersey.java.net>

Listing 7.1: An example of a valid BRL instance.

```

1 <babelo>
2   <from name="cats" href="convert/cats">
3     <to name="fps" href="convert/cats/fps" />
4     <to name="mooshak" href="convert/cats/mooshak" />
5   </from>
6   <from name="fps" href="convert/fps">
7     <to name="cats" href="convert/fps/cats" />
8     <to name="mooshak" href="convert/fps/mooshak" />
9   </from>
10  <from name="mooshak" href="convert/mooshak">
11    <to name="cats" href="convert/mooshak/cats" />
12    <to name="fps" href="convert/mooshak/fps" />
13  </from>
14 </babelo>

```

The **Convert** function converts an exercise from an input format to an output format. The function includes three parameters: **inFormat** - the format of the exercise to convert; **outFormat** - the conversion output format and **refEx** - reference (URL based) of the exercise to convert. The function returns an archive with its contents complying the output format. Figure 7.3 shows a conversion between two formats: Mooshak and FPS. BabeLO uses the **convertFrom()** method of **MooshakConverter** to transform the programming exercise from mooshak format to the PEXiL format. Then, the **convertTo()** method of the **FPSConverter** is used to transform the exercise from the PEXiL format to the FPS format.

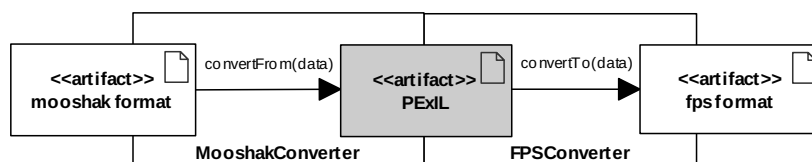


Figure 7.3: The Convert function.

Being an extensible converter BabeLO can be augmented with other classes implementing a specific Java interface defining the **convertFrom()** and **convertTo()** methods mentioned above, as well as other similar methods to handle collections.

7.3 Evaluation results

This section evaluates the effectiveness and efficiency of BabeLO on two uses: to relocate exercises to a different repository and to use an AS in a network of heterogeneous systems.

7.3.1 Case study 1: repositories exchange

In the first case study the BabeLO service is used to create a repository collection based on a problem set of an existing programming contest. Figure 7.4 depicts the interconnection of BabeLO with two other components to achieve this purpose.

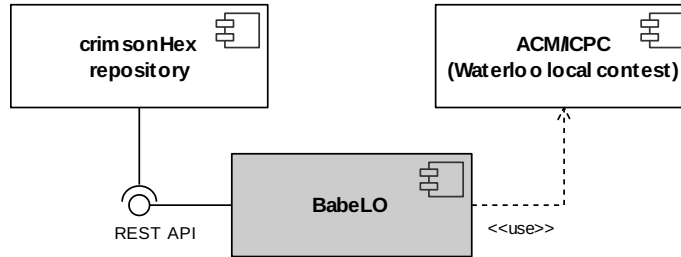


Figure 7.4: Case study 1 - exchanging exercises between repositories

In order to measure the efficiency of the BabeLO Service two repositories were used: an installation of crimsonHex [LQ09a] - a repository of programming exercises - and the ACM/ICPC contest system of the University Waterloo. The exercises were copied from the Waterloo repository to crimsonHex installation. The former supports the IMS CC format for describing the exercises. The later uses the FPS format to describe the contest problem sets.

The time to convert collections with different numbers of exercises was measured and compared with a standard download, i.e using the time of a HTTP GET as benchmark. The average size of the exercises used in this experiment was 12KB and time was measured in milliseconds. The PC where the tests occurred was an Asus M70VSeries with Windows Vista Home Premium (32 bits), Intel(R) Core(TM)2 Duo P8400 @2.26GHz and 4GB RAM. Table 7.3 summarizes the results and presents the overheads introduced by BabeLO.

Table 7.3: Download time (ms) and overhead of BabeLO

# ex.	Direct		BabeLO		Over head
	Coll.	P.Ex.	Coll.	P.Ex.	
1	7	7.00	78	78.01	10.11
10	22	2.23	693	69.34	30.53
100	145	1.45	5,970	59.73	40.23
1000	1,395	1.40	56,348	56.34	39.43

The main conclusion is that BabeLO introduces an overhead of a factor of 10 when converting an exercise. This overhead increases with the size of collections when taking as benchmark the direct download of a collection. This fact is understandable since BabeLO has to process

each exercise, one-by-one, and takes little advantage of collections. Analysing the columns for direct download one notices that, although the time for downloading a collection increases with the size of the collection, the average time per exercise reduces significantly. This is due to the fact that a collection download avoids the time of establishing a connection to the repository for each exercise, which is a significant part of the download. Analysing the columns for the BabeLO download one notices something similar; the time per exercise is reduced but it is not as expressive as in the direct download. This is due to the fact that the conversion time is a larger share of the overall time and saving the time of establishing connection has less impact.

At first sight an overhead factor of 10 may seem impracticable for on-the-fly format conversion. It should be noted that exercises only need to be downloaded from a remote site for the first time they are used. As in any HTTP based system a cache can and should be used to improve overall efficiency [FT00].

7.3.2 Case study 2: automatic assessment

The effectiveness of BabeLO was also validated with its inclusion on a network of heterogeneous systems³ with automatic assessment. The architecture depicted by UML components diagram in Figure 7.5 is composed by several systems and services such as LOR to store/retrieve exercises and AS to evaluate students' exercises.

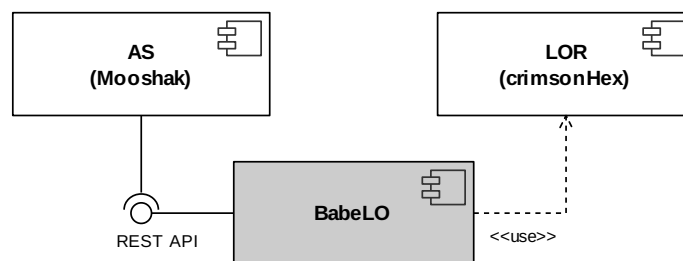


Figure 7.5: Case study 2 - automatic assessment.

The first time that the user send an attempt to the AS, the engine uses the BabeLO service to convert the exercise from its original format to the Mooshak internal format and then it caches the BabeLO response for further use. After that the AS evaluates the student's attempt and returns an evaluation report. The student may submit repeatedly, integrating the feedback received from the AS. In the end, the AS feeds the gradebook of the student, typically included in a LMS. The efficiency of this network was validated in a classroom experiment with several assignments comprising 18 exercises. From a extensive list of results the number of times that the AS accessed to BabeLO (18) comparing it with the number of

³This network is used in the Ensemble evaluation in chapter 10

attempts submitted by students to solve the exercises (819). These results show two things: the AS cache mechanism is working as expected and the BabeLO service made successfully conversions since no more requests were made.

7.4 Summary

This chapter presents the design and implementation of a service for converting programming exercise formats called BabeLO. The chapter starts by presenting the pivotal approach used to perform the conversion and the pivotal data format (PExIL). Then, the abstract service definition, its components and web service interface are presented. Finally, a report on the use of BabeLO in two concrete scenarios is included: to relocate exercises to a different repository and to use an assessment system in a network of heterogeneous systems.

This service was design to be integrated in networks of heterogeneous e-learning systems including contest management systems, programming exercise authoring tools, assessment systems and repositories of learning objects. In its current status BabeLO is available for download at the following URL: <http://ensemble.dcc.fc.up.pt/BabeLO>.

Chapter 8

Programming Teaching Assistant

"If, what is learned is not put into practice, the student is like a cow that does not yield milk; a fruit lacking in taste, a book bereft of wisdom."

in Atharvaveda

A teaching assistant (TA) is a person who assists a teacher, typically in practical classes. The task of a TA in a programming course is usually to help students in solving exercises and assignments. They must help students to use programming tools (e.g. integrated programming environments, compilers, debuggers), check if they have solved the exercises and provide feedback to help them to overcome their difficulties. Unfortunately, the number of TAs is frequently insufficient for the number of students enrolled in introductory programming courses and they are only available on computer labs and on a certain timetable. The role of an automatic TA specialized in programming exercises is that of a tool for bridging between the teacher and the students, and providing them with the best systems for each task.

This chapter presents a tool called Petcha - an acronym of Programming Exercises TeaCHing Assistant - that acts as an automated TA in computer programming courses. Petcha meets this objective by helping both teachers to author programming exercises and students to solve them. It also coordinates a network of heterogeneous systems, integrating automatic assessment systems, learning management systems, learning object repositories and integrated development environments.

Petcha can be described as a scaffolding tool since it complements existing tools and was designed to be easily removed once it is no longer needed. For instance, rather than providing its own environment for solving exercises Petcha promotes the use of existing Integrated Development Environments (IDEs) and different IDEs can be used with Petcha,

such as Eclipse or Visual Studio. More than just a scaffolding tool, Petcha is also a pivot component on a network integrating other e-learning systems. Unlike a human TA Petcha delegates most of its work to others, as it is fundamentally a coordinator of e-learning systems. These e-learning systems are used for: 1) automatic evaluation of programs and feedback generation; 2) authoring and storing of programming exercises as learning objects; 3) managing instruction and learning activities. A proper integration of these tools sets up the necessary foundations for the practice of solving programming exercises and has a great impact on the acquisition of programming skills.

8.1 Use Cases

As happens with a human TA, Petcha needs to interact both with teachers and students. Thus, these two use cases provide an overview of Petcha features. Figure 8.1 shows an UML Use Cases diagram for both uses.

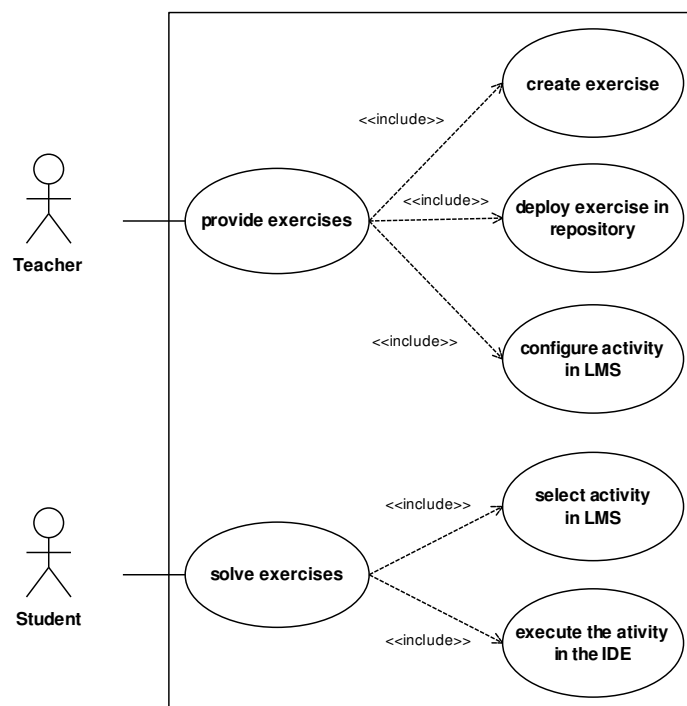


Figure 8.1: Petcha use cases.

Although complementary, these two tasks share a number of requirements. Both teachers and students need to: code and test programs in an IDE; send and retrieve programming exercises from a repository; check program code against test cases. Thus, although the graphical user interface of both user profiles shown in Figure 8.2 is apparently very different,

they actually share many Petcha internal functions. The following subsections present both use cases in more detail.

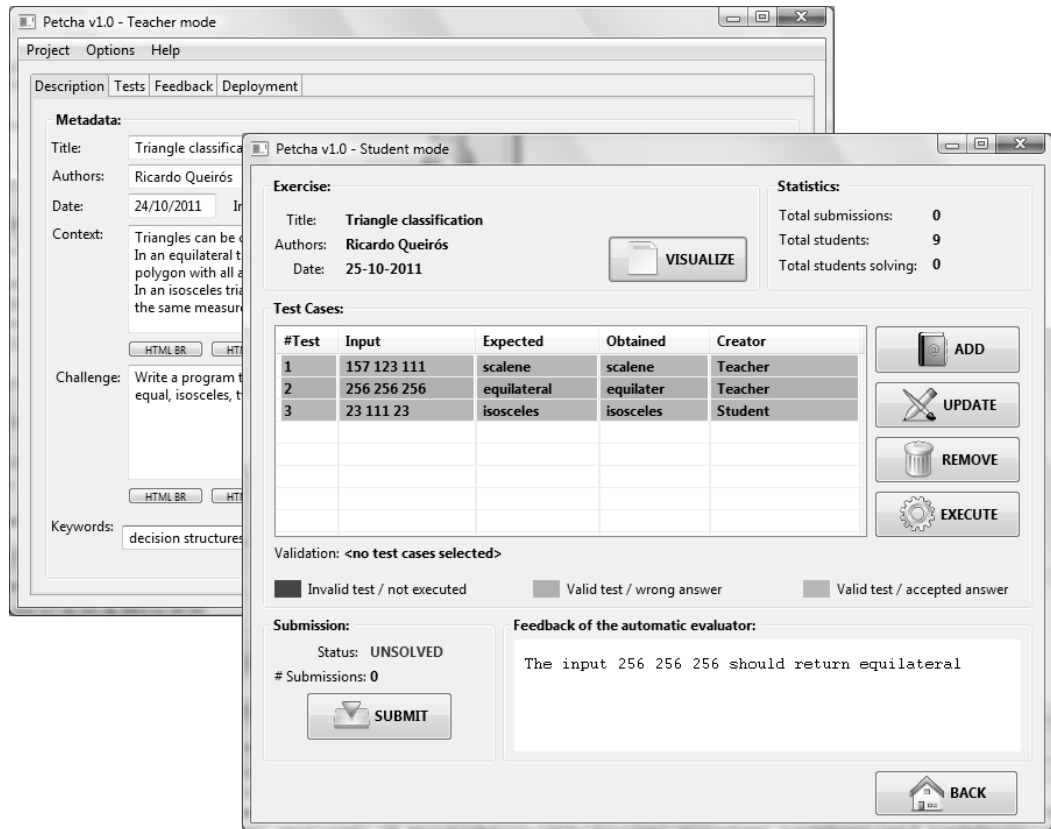


Figure 8.2: The GUI of Petcha with teacher and student modes.

8.1.1 Teacher

To author and deploy a programming exercise in Petcha teachers must perform the following three tasks:

- *Create programming exercises.* In the authoring task, teachers automatically create most of the resources related with programming exercises such as expositive resources (e.g. exercise description) and evaluation resources (e.g. test cases, correctors, feedback files). The upper left window of Figure 8.2 shows an example where the teacher describes an exercise and sets related metadata. Other tabs in this window are used for defining tests, assigning feedback to error patterns and publishing the exercise. All the resources defined in these tabs are encoded using PExIL [QL11e]. The aim of PExIL is to consolidate all the data required in the programming exercise utilization, from creation to evaluation, covering also solving, the grading and feedback. The PExIL

definition supports the concept of incremental feedback to control the appearance of both types of feedback upon a submission of a student's attempt.

- *Deploy programming exercises in a repository.* In the deployment task, teachers can package and publish programming exercises in repositories. The packaging subtask consists on the selection of a package format and its generation. By default, Petcha supports the IMS Common Cartridge (IMS CC) specification as the package format. The generation of an IMS CC package is performed in two steps. First the manifest is generated from a valid PExIL instance and all the resources are assembled in a ZIP package. After that, teachers must publish the package on a repository. In order to be an eligible the repository must adhere to content (IMS CC) and communication specifications (IMS Digital Repositories Interoperability – IMS DRI).
- *Configure programming activity in LMS.* For this task teachers search in repositories for suitable programming exercises, group them in a collection and store a reference to the collection in a LMS as a Learning Tools Interoperability (LTI) activity.

8.1.2 Student

To solve programming exercises using Petcha students performs the following two tasks:

- *Select an activity in the LMS.* In this task students select an activity defined by the teacher in the LMS. This selection triggers an LTI launch of Petcha. The launch includes student's contextual information that can be used for presentation purposes (e.g. personalize the Petcha frontend) or for sequencing purposes (e.g. assign an exercises sequence model). Petcha is launched as a Java Web Start (JAWS) application on the computer of the student enabling the interaction of Petcha with the IDE through shell commands.
- *Perform an activity using the IDE and Petcha.* When a student starts solving an exercise Petcha automatically creates a project on the IDE of the student¹. Then the student reads the exercise description in Petcha's GUI and solves it on the IDE. The student should test the code locally by executing the test cases provide by the teacher and is encouraged to create new ones. If new test cases are created, a validation step is performed to verify that they meet the specification defined by the teacher in the authoring phase. The right window on Figure 8.2 shows an example where the code of the student did not pass all the local tests (two provided by the teacher and one created by the student). Even so, the student decided to submit the code to the assessment system and received a report evaluation with a feedback message indicating an input

¹Currently Petcha supports two IDEs - Eclipse and Visual Studio Express - but Petcha can be easily extended to support other IDEs.

data that generated a wrong answer. The student may submit repeatedly, integrating the feedback received from the AS. In the end of this cycle, Petcha reports the exercise usage data back to the repository and sends a grade back to the LMS gradebook.

8.2 Design

The design of Petcha is described by the UML class diagram shown in Figure 8.3. This diagram models Petcha statically showing the relations among classes.

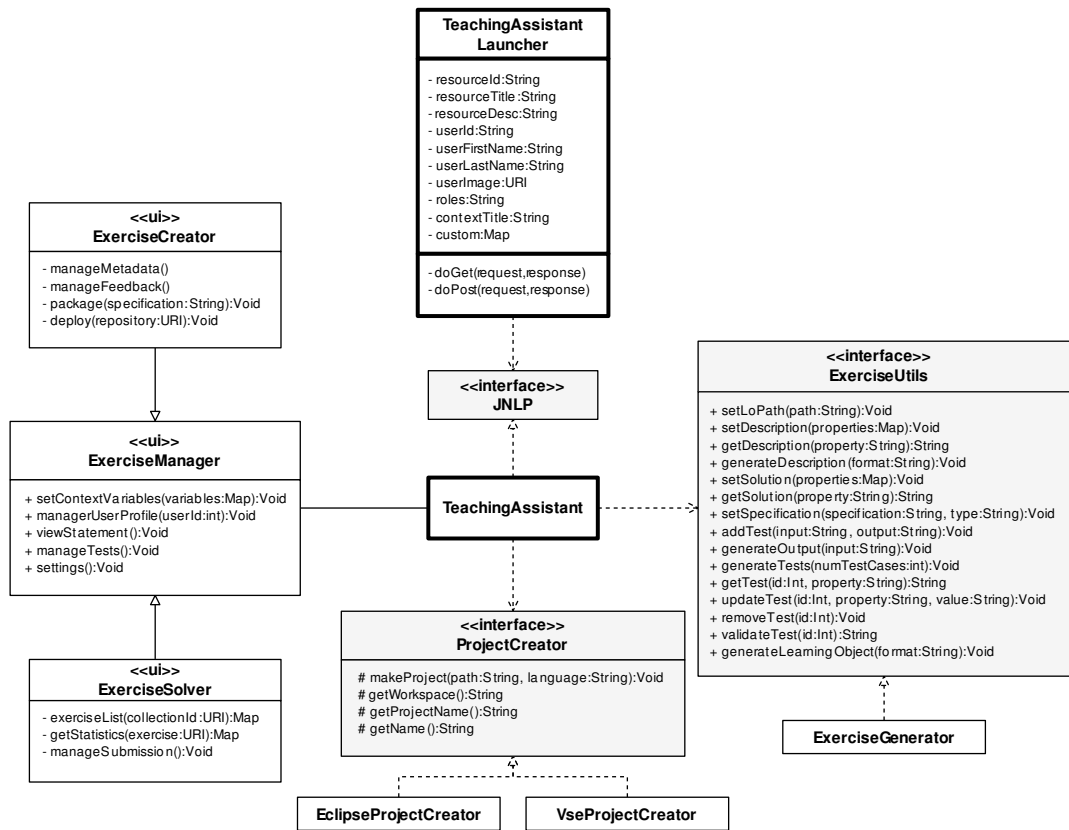


Figure 8.3: The UML class diagram of Petcha.

The Petcha component relies on two different processes (running on different JVMs): **TeachingAssistantLauncher** and **TeachingAssistant**. These active classes (with a thicker border in the diagram) initiate and control the flow of the activity in two different moments. The former act as an adapter for handling the HTTP requests and launching the Teaching Assistant (Petcha). The latter is the main process where Petcha effectively runs.

8.2.1 Class `TeachingAssistantLauncher`

Petcha is launched as a Java Web Start (JAWS)² application and interacts with the students IDE by using system shell commands on the students' computer. Since the LTI specification supports only web-based Tool Providers there is a need to create a web service that injects LTI variables on Petcha. A JAVA servlet fulfils this role and dynamically generates a JNLP file "on-the-fly" based on the LTI variables.

The `TeachingAssistantLauncher` class is the entry point for Petcha acting as a JAVA servlet that generates dynamically a JNLP file based on the HTTP request initiated by the user. The JNLP specifies how to launch JAWS applications. A JAWS application is a framework that allows users to start JAVA application software directly from the Internet using a web browser. After the generation, the servlet redirects the request to the generated JNLP file in order to launch the JAWS program (Petcha). These variables can later be used, for instance, to personalize the front-end of the tool provider. Two user profiles can perform this request: teachers select a new or an existent exercise in the repository for authoring purposes and students select an activity (one or several exercises) in the LMS for solving purposes. A teacher perform an HTTP GET request. The servlet class `TeachingAssistantLauncher` receives this request in the `doGet` method. Then, it automatically generates the JNLP file with the inclusion of an exercise identifier in the case of the exercise's authoring is based on an existent exercise. In this request no LTI variables are used. A student perform an HTTP POST request that is handled by the servlet using the `doPost` method. This request automatically includes several LMS context variables based on the LTI specification. The servlet feeds their internal properties (e.g. `resourceId`, `resourceTitle`) using the implicit setter methods. These properties correspond to the LTI variables. Then the correspondent getter methods are used to inject the LTI variables in the JNLP file. Listing 8.1 shows an excerpt of the generated JNLP file.

The `resources` element is used to specify all the resources, such as Java class files and native libraries which are part of the application. A `JAR` element specifies a Java Archive (JAR) file that is part of the application's classpath. The JAR file is loaded into the JVM using a `ClassLoader` object. The JAR file contains Java classes and other resources, such as icons and configuration files (available through the `getResource` mechanism). The `j2se` element specifies what Java 2 SE Runtime Environment (JRE) versions are supported by the application. The `application-desc` element indicates that the JNLP file is launching an application (as opposed to an applet). The `application` element has an optional attribute called `main-class` which can be used to specify the name of the application's main executable class (`TeachingAssistant` class). This element includes several `argument` elements representing each LTI variable injected by the `TeachingAssistantLauncher` class.

²Official Web site: <http://www.oracle.com/technetwork/java/javase/javawebstart/index.html>

Listing 8.1: POST data sent by the Tool Consumer.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <jnlp spec="1.0+" codebase="$codeBase">
3    <information>
4      <title>Pivot Component</title>
5      <homepage href="$codeBase"/>
6      <icon href="resources/icon.png" />
7      <description>
8        PETCHA – fostering the practice programming exercises solving.
9      </description>
10   </information>
11   <security><all-permissions /></security>
12   <resources>
13     <j2se version="1.5+" />
14     <jar href="resources/Pivot_Component.jar" />
15     ...
16   </resources>
17   <application-desc main-class="Pivot">
18     <argument>$colId</argument>
19     <argument>$studentFirstName</argument>
20     ...
21   </application-desc>
22 </jnlp>

```

8.2.2 Class TeachingAssistant

The **TeachingAssistant** class is the entry point for the main process where Petcha effectively runs. Its main objectives are to manage the graphical user interface both for teachers and students, the programming exercises and the integrated development environments. These objectives are achieved by using the following classes respectively: **ExerciseManager**, **ExerciseUtils** and **ProjectCreator**. These classes are detailed in the next sub-subsections.

8.2.2.1 Class ExerciseManager

The graphical user interface (GUI) on Petcha is based on the Standard Widget Toolkit (SWT)³. This toolkit aims to provide portable user-interfaces across the operating systems. The **ExerciseManager** class manages the Petcha's GUI. A new instance of this superclass is initialized by the **TeachingAssistant** class based on the user profile. Two classes extend this superclass for each user profile (creating and solving exercises): **ExerciseCreator** - provides

³<http://www.eclipse.org/swt/>

the user interface for the creation, packaging and deployment of programming exercises by teachers and `ExerciseSolver` - provides the user interface for the testing and submission of students' attempts to solve programming exercises. Both classes inherit methods for common features of Petcha such as statement and tests visualization but redefines others regarding the user profile tasks. An example is the `manageMetadata` method from the `ExerciseCreator` class that handles the metadata form to be fulfilled by the teacher while creating the programming exercise.

8.2.2.2 Interface ProjectCreator

One of the most important tasks of both user profiles is the exercise coding: the teacher needs to code the program solution and the student needs to code an attempt to solve it. Both requires the use of an IDE. Thus, the `TeachingAssistant` class uses the `ProjectCreator` interface to abstract the use of specific IDEs on Petcha. This interface must be implemented to provide support for other IDEs. Table 8.1 enumerates the four required methods.

Table 8.1: `ProjectCreator` interface methods.

Method	Description
<code>makeProject(path:String, lang:String)</code>	Creates a project on the IDE. The <code>path</code> argument includes the location and name of the project. The <code>lang</code> argument is the programming language for the project.
<code>getWorkspace()</code>	Returns a string with the project workspace location.
<code>getProjectName()</code>	Returns a string with the name of the project.
<code>getName()</code>	Returns a string with the IDE name.

On implementing a `ProjectCreator` class one must pay attention to the `makeProject` method. A project contains source code and related files for building a program in a specific programming language. Thus, a set of predefined files and directories need to be generated for the project creation.

8.2.2.3 Interface ExerciseUtils

The `ExerciseUtils` interface defines a set of abstract methods for the generation of resources that compose a programming exercise. The representation of a programming exercise relies on the PExIL specification. The `ExerciseGenerator` class (implements the interface `ExerciseUtils`) uses the Java Architecture for XML Binding (JAXB) to map Java classes to XML representations. In other words, JAXB provides the ability to marshal Java objects into XML and the reverse.

The generation of a programming exercise as a learning object is straightforward. Petcha uses as input a valid PExIL instance and a program solution file and generates:

1. an exercise description in a given format and language;
2. a set of test cases and feedback files;
3. a PExIL descriptor;
4. a valid IMS CC manifest file.

Then, a validation step is performed to verify that the generated tests cases meet the specification presented on the PExIL instance and the manifest complies with the IMS CC schema. Table 8.2 lists for each resource type the corresponding methods of the `ExerciseUtils` interface.

Table 8.2: *ExerciseUtils* interface methods.

Resource	Method
Description (1)	<code>setDescription(properties:Map)</code> <code>getDescription(property:String)</code> <code>updateDescription(property:String, value:String)</code> <code>generateDescription(format:String, language:String)</code>
Tests & Feedback (2)	<code>setSpecification(specification:String, type:String)</code> <code>addTest(input:String, output:String)</code> <code>generateOutput(input:String)</code> <code>generateTests(numTestCases:int)</code> <code>getTest(id:Int, property:String)</code> <code>updateTest(id:Int, property:String, value:String)</code> <code>removeTest(id:Int)</code> <code>validateTest(id:Int)</code>
PExIL (3)	<code>generatePexil()</code>
LO (4)	<code>setLoPath(path:String)</code> <code>generateManifest(format:String)</code> <code>packageLo(format:String)</code>

The following paragraphs detail these four groups of methods.

1) Exercise statement generation

For the generation of an exercise description (Figure 8.4) it is necessary to obtain the format and the natural language of the exercise description. The former is given by the generator

tool and the latter is obtained from the total number of occurrences of the `xml:lang` attribute in the `title` element of the PExIL instance.

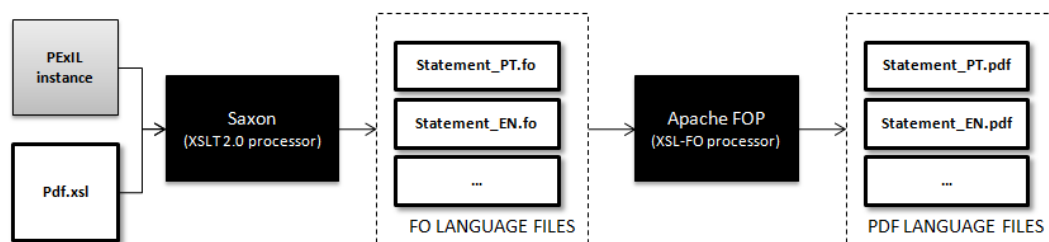


Figure 8.4: Generation of the exercise descriptions.

The generator tool receives as input a valid PExIL instance and a respective XSLT 2.0 file and uses the Saxon XSLT 2.0 processor combined with the `xml:result-document` element to generate a set of XSL Formatting Objects (FO)⁴ files corresponding to the human languages values founded in the `xml:lang` attribute. W3C XSL-FO is a markup language for XML document formatting which is most often used to generate PDFs. Listing 8.2 shows an excerpt of the `Pdf.xsl` file. This stylesheet generates the FO files based on the textual elements of a PExIL instance.

Listing 8.2: Exercise statement generation.

```

1 <xsl:template match="pexil:title">
2   <xsl:variable name="uri" select="concat('desc',@xml:lang,'.fo')"/>
3   <xsl:result-document href="resources/{\ $uri}">
4     <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
5       <!--apply templates over the textual elements --> ...
6     </fo:root>
7   </xsl:result-document>
8 </xsl:template>
  
```

In the next step, the FO files are used as input to the Apache FOP formatter – an open-source and partial implementation of the W3C XSL-FO 1.0 standard - generating for each FO file the corresponding PDF file. The use of the PExIL definition to generate exercise descriptions does not end here since it is included in the LO itself making it possible, at any time of the LO life-cycle, to regenerate the exercise description in different formats or natural languages. Figure 8.5 shows a typical exercise in PDF format.

The description also includes a description and an example of a test case. In the case of the absence of the `input/description` and `input/example` the generator relies on the `specification` element to generate the test data and include it in the exercise description.

⁴XSL Version 2.0 - W3C Working Draft: <http://www.w3.org/TR/xslfo20/>

Problem C: Selfdescribing Sequence

Solomon Golomb's *selfdescribing sequence* $(f(1), f(2), f(3), \dots)$ is the only nondecreasing sequence of positive integers with the property that it contains exactly $f(k)$ occurrences of k for each k . A few moments thought reveals that the sequence must begin as follows:

n	1	2	3	4	5	6	7	8	9	10	11	12
$f(n)$	1	2	2	3	3	4	4	4	5	5	5	6

In this problem you are expected to write a program that calculates the value of $f(n)$ given the value of n .

Input

The input may contain multiple test cases. Each test case occupies a separate line and contains an integer n ($1 \leq n \leq 2,000,000,000$). The input terminates with a test case containing a value 0 for n and this case must not be processed.

Output

For each test case in the input output the value of $f(n)$ on a separate line.

Sample Input

```
100
9999
123456
1000000000
0
```

Sample Output

```
21
356
1684
438744
```

Figure 8.5: An example of an exercise description.

2) Test cases and feedback generation

The generation of test cases and feedback relies on the specification element of the PExIL definition. The generator tool can be parametrized with a specific number of test files to generate. By default, the tool calculates this parameter as the number of test cases based on the total number of variables and the number of feedback messages. In the former, the number of test cases is given by the formula 2^n where the base represents the number of range limits of a variable and the exponent the total number of variables. Testing the range limits of a variable is justified since their values are usually not tested by students, thus with a high risk of failure. In the latter, the tool generates a test case for each feedback message found. The generation will depend on the successful evaluation of the XPath expression included in the condition attribute of the **when** element. Listing 8.3 clarifies how the generator calculates the number of test cases.

Suppose that the generator tool is parametrized to generate 10 test cases. Using the previous example the estimation of the number of test cases and its respective input values is demonstrated in the Table 8.3.

Listing 8.3: Example of the generation of test cases.

```

1 <repeat count="$numTestCases">
2   <line>
3     <data id="n1" type="float" min="0" max="1000"/>
4     <data id="n2" type="float" min="0" max="1000"/>
5     <data id="n3" type="float" min="0" max="1000"/>
6   </line>
7   <line/>
8 </repeat>
9 <when condition="$n1>$n2">
10  <feedback xml:lang="en-GB">
11    Numbers can be given in descending order
12  </feedback>
13 </when>

```

Table 8.3: Generation of input data of test cases (R=random).

Var	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
n1	0	0	0	0	1000	1000	1000	1000	Min=n2+1	R
n2	0	0	1000	1000	0	0	1000	1000	n2	R
n3	0	1000	0	1000	0	1000	0	1000	R	R

The test values are: eight tests to cover the range limits of all variables ($2^3 = 8$) and one test to represent the constraint included in the feedback message. Note that this test case will be executed only if the expression included in the condition attribute was not covered in the previous eight test cases; the remaining tests are generated randomly. Also note that the creator of the programming exercise can statically define new test cases and use the PExIL definition for validation purposes.

3) PExIL descriptor

The generation of a programming exercise as a LO in the IMS CC format implies the inclusion of the exercise as a LAO resource. A LAO resource contains a XML descriptor that serves as the entry point for accessing the information about a LAO required to import it into the target system. The descriptor file is not intended to be displayed within the target system. Rather, it is intended to be processed by the target system upon import of the cartridge. The descriptor file (`pexil.xml`) is associated with a LAO by means of a `file` element. It includes references for two XML documents: `pexildefinition.xml` - a serialized copy of a PExIL definition and `pexilmanifest.xml` - a manifest with references for all the resources generated. The former describes all data needed for the generation of the evaluation resources. The latter is a manifest with references for all the evaluation resources generated.

Figure 8.6 shows the PExIL files included in the LO package.

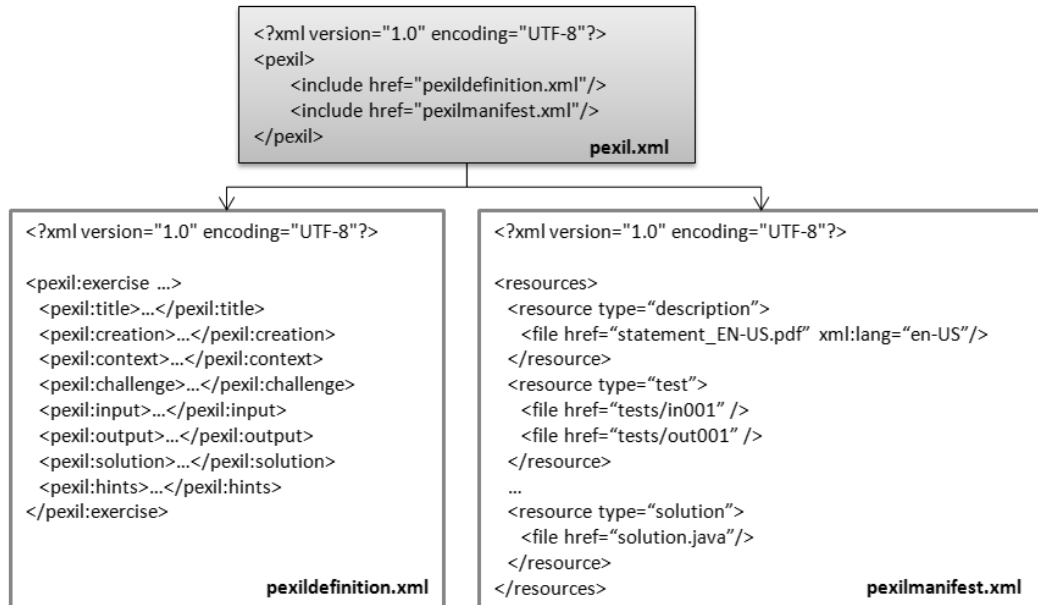


Figure 8.6: PExIL files.

4) Manifest generation An IMS CC learning object assembles resources and metadata into a distribution medium, typically a file archive in ZIP format, with its content described by a manifest file named `imsmanifest.xml` in the root level. The main sections of the manifest are: metadata which includes a description of the package, and resources which contains a list of references to other resources in the archive and dependency among them.

In this example (Figure 8.7) the resources section starts with a LAO resource (1) pointing to the PEXIL descriptor. This file is responsible for the automatic generation of all the other files included in the package (with the exception of the solution program and images). The description of the exercise is included on the manifest as a WCR resource (2). This type of resources can be automatically rendered by the browser without any additional processing. The program solution (3) is associated with metadata since this resource should not be made visible in player mode to the students and will be used only to regenerate test cases and in the evaluation phase of the programming life-cycle. The test cases are defined with a pair of input and output files (and feedback files) as resource objects (4 and 5). Finally, the BLTI link is included as a LAO resource (6). This link points to a XML file that includes all the data needed to integrate the cartridge in a LMS-web application communication. Despite its inclusion, this link will not be used since the teacher will configure the access to Petcha as an LTI activity rather than an LTI resource.

This XML file contains information to create a link in a Tool Consumer (e.g. LMS). Upon the user's click on the LMS, the execution flow passes to a Tool Provider along with contextual information about the user and Consumer.



Figure 8.7: Structure of the IMS CC manifest file.

The metadata section of the IMS CC manifest comprises a hierarchy of several IEEE LOM elements organized in several categories (e.g. general, lifecycle, technical, educational). In order to achieve interoperability a binding of the textual elements of the PExIL definition and the correspondent IEEE LOM elements was defined. The generator tool uses this binding to generate the LOM elements through a template pattern. Table 8.4 presents a binding of the PExIL textual elements and the corresponding LOM elements which will be used by the generator tool to feed the IMS CC manifest.

One may note some redundancy between the PExIL and the IMS CC manifest since both reference the same files. This happens since the exercise must be independent from the chosen export format. For instance, this allows the use of exercise as input for a format converter service such as BabeLO detailed in chapter 7. By defining this set of metadata at the LOM side, e-learning systems continue to use the metadata included in the IMS CC manifest to search for programming exercises, rather than using a specialized XML dialect

Table 8.4: Binding PExIL to IEEE LOM.

Type	Schema	Element Path
Title	LOM	//lomcc:general/lomcc:title
	PExIL	/exercise/title
Date	LOM	//lomcc:contribute[lom:role='Author']/lom:date
	PExIL	/exercise/creation/date
Author	LOM	//lomcc:contribute[lom:role='Author']/lom:entity
	PExIL	/exercise/creation/authors/author/v:VCard/v:fn
Event	LOM	//lomcc:general/lomcc:coverage
	PExIL	/exercise/creation/event

such as PExIL. In order to validate the IMS CC cartridges previously generated was used the IMS validator⁵. This service validates cartridges for conformance with the IMS Common Cartridge v1.0 and/or v1.1 specification. In the validation process the IMS CC Validator test the whole cartridge (or just the XML manifest) verifying the following type of constraints:

Static - the parameters (e.g. file names) are fixed in the profile (e.g. `imsmanifest.xml` must exist at the root of the package);

Dynamic - the parameters are taken from an instance document in the package (e.g. `href` attribute of a `resource` element must point to a QTI file)

Conditional - the constraint depends on a condition (e.g. if parameter `contenttype` is `question` then the `href` attribute must point to a QTI file).

The cartridges generated from PExIL instances using the methodology presented in the previous sub-section passed all tests performed by the validator.

8.3 Summary

This chapter presents the design and implementation of a tool that acts as an automated TA in computer programming courses called Petcha. Petcha helps both teachers to author programming exercises and students to solve them. It also coordinates a network of heterogeneous systems typically found in the computer programming domain. The chapter focus on the implementation details related with the design of Petcha and the automatic generation of programming exercises as learning objects. The validation of the usability and usefulness of Petcha as an automated TA is presented in chapter 9 where all the network is evaluated and

⁵Official Web site: <http://validator.imsglobal.org/>

Petcha is the front-end to its users. Petcha was recently updated to support the IMS LTI 1.1. With this support Petcha can deliver student grades to the gradebook of the student typically found in the school LMS. Petcha is available for download at the following URL: <http://ensemble.dcc.fc.up.pt/Petcha>.

Part IV

Validation

Chapter 9

Ensemble evaluation

"A successful [software] tool is one that was used to do something undreamed of by its author."

S. C. Johnson

This chapter evaluates the Ensemble instance presented in chapter 5. The Nielsen's model [Nie94] is used to evaluate the acceptability of Petcha as the user interface of the network. For this evaluation an experiment was conducted with undergraduate students and their teachers. This chapter starts by presenting the evaluation model followed in the experiment based on the heuristics of Nielsen. Then, the design of the experiment is described. The description includes the methodology followed, the educational context and infrastructure where the experiment occurs and the instruments used to collect the data. Finally, the evaluation results are presented and analysed.

9.1 Evaluation Model

According to Nielsen [Nie94] the acceptability of a system is defined as the combination of social and practical acceptability. The former determines the success/failure of the system, since the more the system is socially acceptable the greater is the number of people using it. The latter relates factors such as usefulness, cost, reliability and interoperability with existing systems. An adaptation to the Nielsen's model [Nie94] is depicted in Figure 9.1.

The **usefulness** factor relates the utility and usability offered by the system. Utility is the capacity of the system to achieve a desired goal. As the system perform more tasks, more utility he has. Usability is defined by Nielsen as a qualitative attribute that estimates how

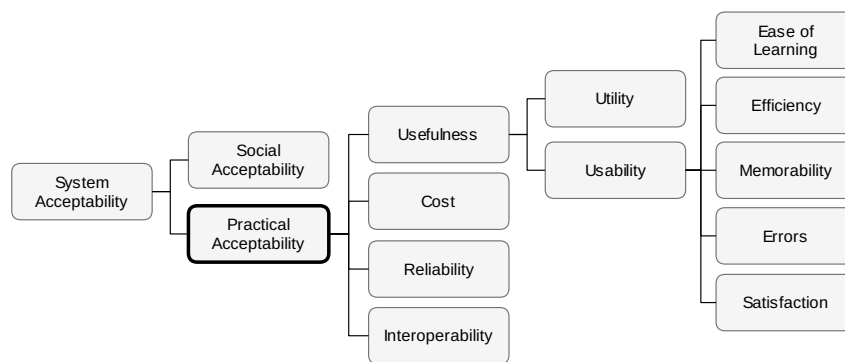


Figure 9.1: System acceptability. Adapted from Nielsen [Nie94].

easy is to use an user interface. He mentions five characteristics involved in the concept of usability: *ease of learning* - the system should be easy to learn so that the user can start doing some work with the system; *efficiency* - the system should be efficient to use, so after the user learns the system, a high level of productivity is possible; *memorability* - the system should be easy to remember so that the casual user is able to return to the system after a period without using it, without requiring to learn it all over again; *errors* - the system should prevent the user from committing errors as should deal with them gracefully and minimizing the chance of occurring catastrophic errors; *satisfaction* - the system should be pleasant to use so that users become subjectively satisfied when using. The **cost** factor was not considered since Petcha is a free (and open source) software. The **reliability** is the ability of a system or component to perform its required functions under stated conditions for a specified period of time. **Interoperability** is the ability of two or more systems or components to exchange information and to use the information that has been exchanged.

9.2 Experiment Design

The experiment took place at the Escola Superior de Estudos Industriais e de Gestão (ESEIG) - a school of the Polytechnic Institute of Porto - during the months of October and November of 2011. The participants were students from the first-year of the course Algorithms and Programming and their teachers. This course is offered to the degree in Mechanical Engineering and aims to introduce students to programming concepts.

9.2.1 Methodology

The experiment methodology followed the experimental research method [Onc11]. For this purpose experimental and control groups (two classes from the same course) were

settled. The first class (the experimental group) had 21 students and the second class (the control group) had 19 students. Students of both groups have similar characteristics such as the gender and previous background. For instance, the gender in both groups were well distributed, respectively 62% and 57% of females in both groups. The conditions of the experiment were also equal for both classes (e.g. syllabus, teaching times, teacher, labs, technical means).

Although it could be assumed that the population of the classes were almost randomly formed, strictly the experiment should be called a *quasi-experiment*. A completely randomised design was not possible due to operational reasons. Based on this type of design, a *static group comparison design* [BG07] was followed where students of class A used Ensemble (the experimental group) and students of class B did not use it (the control group).

The course has an average enrolment of 40 students per year divided in two classes. The course is organized in two lectures of one hour each and one lab session of 4 hours per week. The experiment occurred in 6 lab sessions. In each lab session both groups (a total of 40 students) had 3 exercises to solve. In the experimental group the teacher only intervenes to solve operational issues related to the use of Ensemble and does not give any feedback to students regarding the exercises. Prior to the experiment, teacher and students were prepared for the experiment. A summary of the experiment schedule is presented in Table 9.1.

Table 9.1: The ESEIG experiment schedule

Phases	Date	Description
Preparation	03-09-11	Install and configuration of the network
	10-09-11	Presentation of Ensemble to the teacher
	12-09-11	Creation of C# exercises covering the course program
	19-09-11	Preliminary tests
	26-09-11	Presentation of Ensemble to the students
Experiment	03-10-11	Module nº1 - Variables and arithmetic operations
	10-10-11	Module nº2 - Conditional structures
	17-10-11	Module nº3 - Cyclic structures
	07-11-11	Module nº4 - Vectors manipulation
	14-11-11	Module nº5 - Matrix manipulation
	21-11-11	Module nº6 - Search and Sort algorithms

9.2.2 Infrastructure

The network deployed integrates systems and services selected in section 5.4. A list with the actual systems, versions used and respective requirements is presented in Table 9.2.

Table 9.2: Network selected systems

System	Version	Requirements
Petcha	1.0	Windows/Linux + JAVA 1.6
Moodle	1.9	Windows/Linux + XAMP 1.7.7
CrimsonHex	0.8	Windows/Linux + XAMP 1.7.7
Mooshak	1.6a2	Linux + Apache + TCL
Visual Studio	10.0	Windows XP (or above)
BabeLO	1.0	Windows/Linux + XAMP 1.7.7

To model the physical deployment of these systems and services an UML deployment diagram is depicted in Figure 9.2. This type of diagram shows what hardware components ("nodes") exist, what software components ("artifacts") run on each node and how they are connected. The nodes appear as boxes and the artifacts allocated to each node appear as rectangles within the boxes.

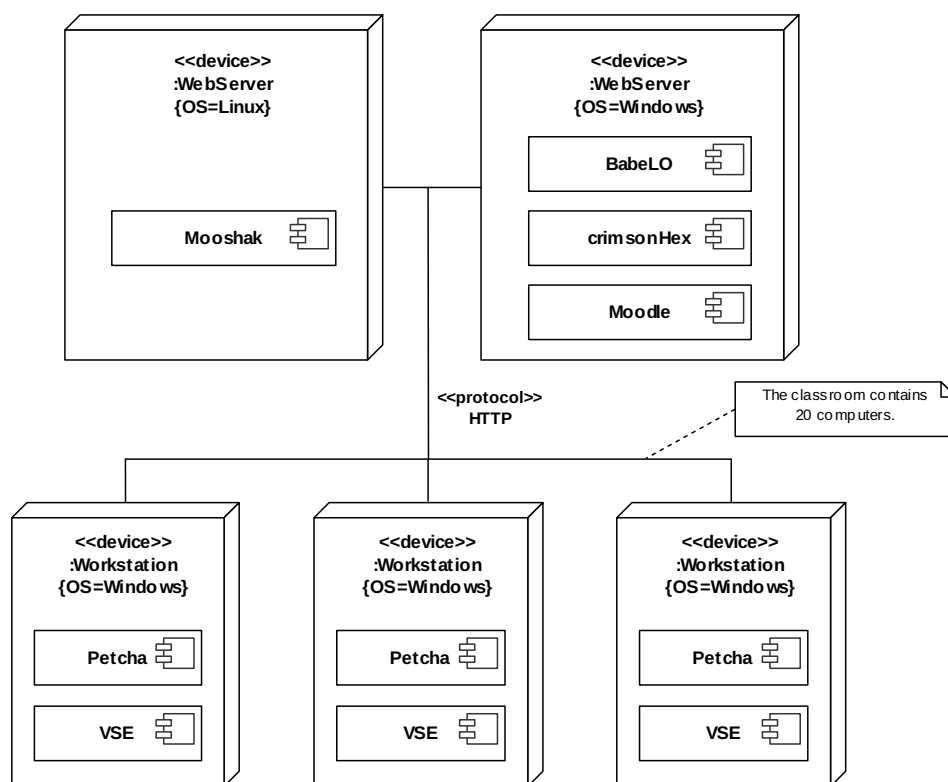


Figure 9.2: UML deployment diagram for the Ensemble instance.

The core and secondary services were installed in two machines located in a servers room at ESEIG. The servers have the following characteristics: Packard Bell - IXTREME I8875

PT, Windows 7 Home Premium, Intel Core i7 (3.40 GHz, 8 MB), 8 GB DDR3 SDRAM, 1 TB Serial ATA. Both servers use different operating systems: one runs the Linux Fedora OS 10 and was used to install Mooshak; the second server uses the Windows 7 Professional and housed the remaining services (Moodle, crimsonHex and BabeLO).

The axial systems (Petcha and VSE) were installed in 20 computers in the computers lab where the experiment occurs. The computers have the following characteristics: Asus M70VSeries, Windows 7 Home Premium edition Intel(R) Core(TM)2 Duo P8400 @2.26GHz (32 bits), 4GB RAM and 600GB of hard disc. The communication between the lab computers and the servers relies on a physical network with a bandwidth of 100 Mbit/s (Fast Ethernet).

9.2.3 Instruments and Data collection

The instruments used for collecting data on the experiment were the following: surveys (session & final survey), service logs, students' attendance logs and grades.

The **surveys** were fulfilled and collected on-line using Google Forms¹. Two types of surveys were presented to students: session and final survey (Appendix B). The former was filled in by both groups of students after each lab session. The questionnaire² includes questions on the number of solved exercises and the feedback impact. It had an average of 38 responses per session (the equivalent to 95% of the total of students). The latter was presented to the experimental group at the end of the experiment. The questionnaire includes questions on the Petcha usefulness and reliability. The final survey was completed by all the students from the experimental group.

The **service logs** were used to attest the accuracy of the experimental group questionnaires responses. The data collected in the surveys of the experimental class was checked against the logs of Petcha and other systems in the network. An average discrepancy of 4.6% between these two sets of values was found.

The **student attendance** and **student outcomes** (programming module and semester grades) were collected through the Academic Management System used at ESEIG. The data was exported to a spreadsheet to simplify the data processing.

9.3 Results and discussion

In this section the Ensemble network is evaluated based on three Nielsen heuristics: usefulness, reliability and interoperability.

¹<http://www.google.com/google-d-s/forms/>

²<http://goo.gl/AlhsL>

9.3.1 Usefulness

Usefulness combines the utility and usability of a system. Utility is the capacity of the system to achieve a desired goal. Usability is defined by Nielsen as a qualitative attribute that estimates how easy is to use an user interface.

Petcha was evaluated according to the Nielsen's model using a heuristic evaluation methodology. A heuristic evaluation is an inspection method for computer software that helps to identify usability problems in the user interface (UI) design. Jakob Nielsen's heuristics (Appendix A) are arguably the most used usability heuristics for user interface design. Based on these heuristics a questionnaire (Appendix B) with 41 questions was designed in Google Forms. The aim of the questionnaire is to identify usability problems in the UI design of Petcha. Two profiles of users answered this questionnaire in the end of the experiment: 21 students (experimental group) and 4 teachers (although only one was the teacher of the students that participated in the experiment).

Figure 9.3 shows the results obtained grouped by the Nielsen's heuristics. The data collected are shown in graphs. In the chart graphs the heuristics are sorted in ascending order of user satisfaction.

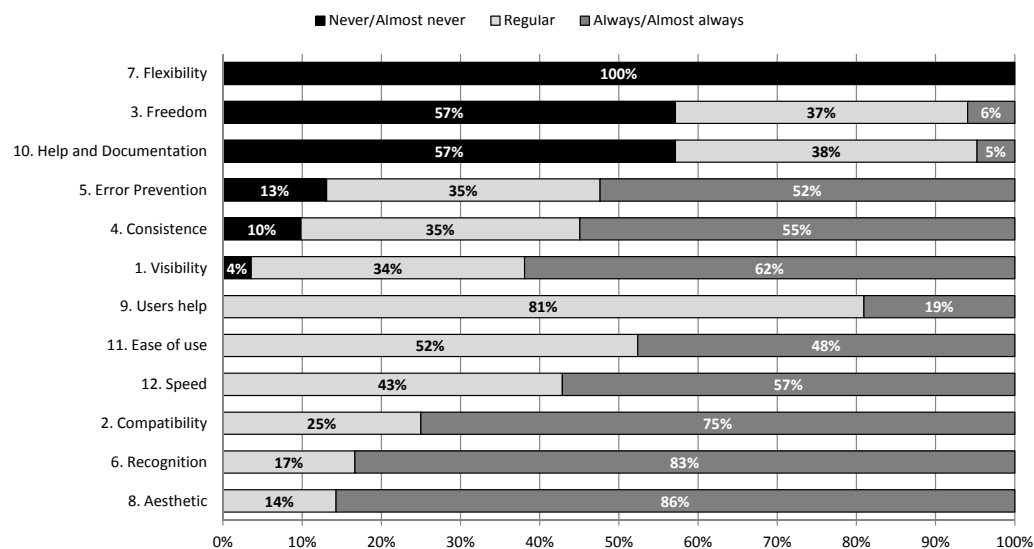


Figure 9.3: Results of each heuristic in the student's profile.

The results highlight deficiencies in three areas: Flexibility, User control and freedom, Help and documentation. In regard to the flexibility of the system, respondents considered that the system do not allow the personalization of the interface as shown in Figure 9.4, more precisely, the activation/deactivation of certain functions and the configuration of the screens. The possibility of use of accelerator keys to speed up the interaction with the TA is also a handicap of Petcha.

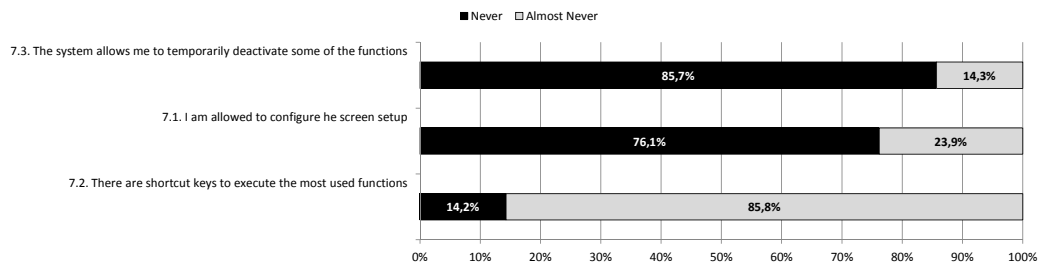


Figure 9.4: Evaluation of Petcha's flexibility.

The User control and freedom is the second worst facet (Figure 9.5). Most of the complaints focused on the inability to cancel or roll back mistakes made to a previous and safe state.

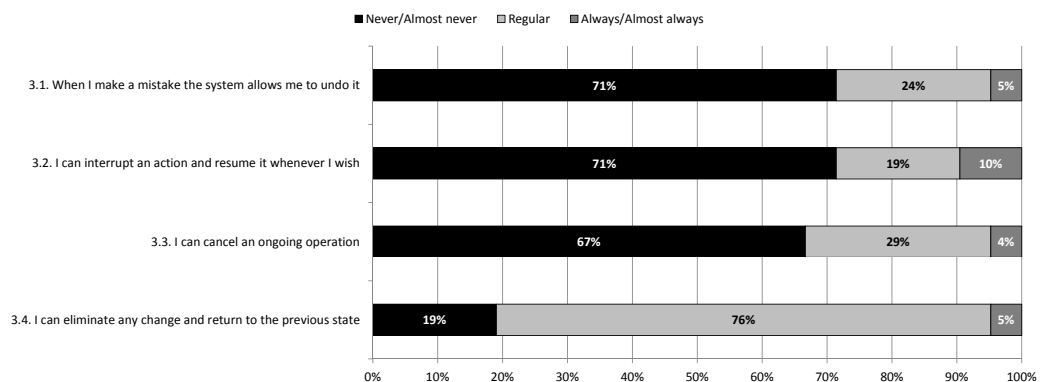


Figure 9.5: Evaluation of Petcha's freedom.

The Help and documentation is another heuristic with negative values as shown in Figure 9.6. The respondents state that it is difficult to find help and documentation in Petcha.

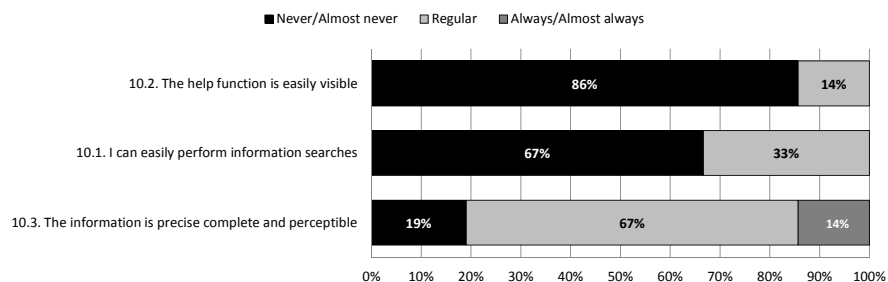


Figure 9.6: Results of each heuristic in the student profile.

The respondents reveal that the error messages are sometimes unclear and inadequate in Petcha. The respondents also state that the documentation is scarce and is hard to find it.

The final classification of Petcha is shown in Figure 9.7.

One can conclude that the majority of students classified Petcha as a good tool according

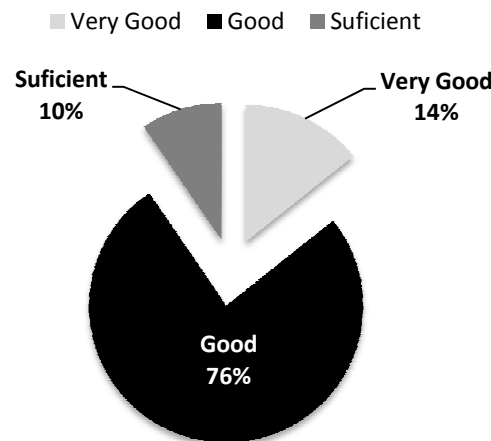


Figure 9.7: Classification of Pectha by students.

to the parameters evaluated.

Regarding the teacher profile, Figure 9.8 shows the results obtained.

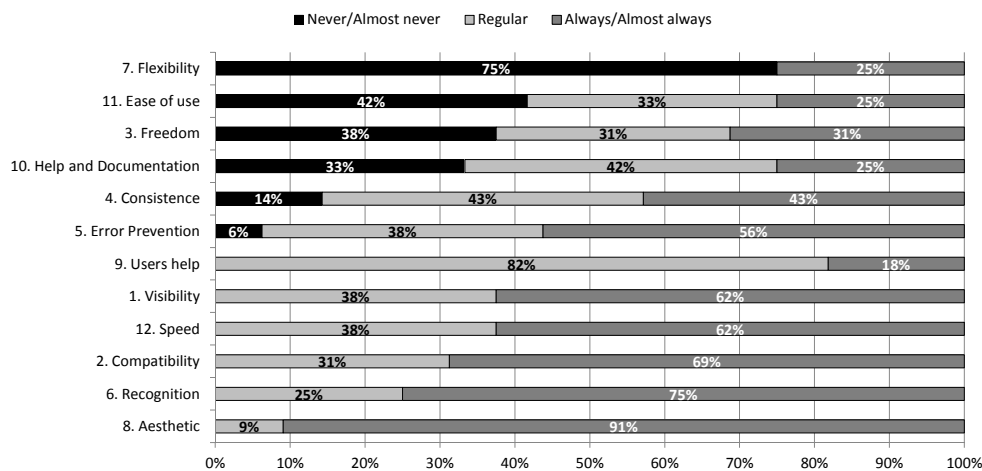


Figure 9.8: Results of each heuristic in the teacher profile.

The results are similar to those of students. In one hand the aesthetic factor was the one with higher values of satisfaction. The respondents considered that the information contained on the screen is only what is needed and the system is aesthetically pleasing on the factors: color, brightness, etc. In the other hand the flexibility, freedom and documentation heuristics had some of the lowest values. However another heuristic had a low value: the Ease of Use. Figure 9.9 shows the results associated to this heuristic.

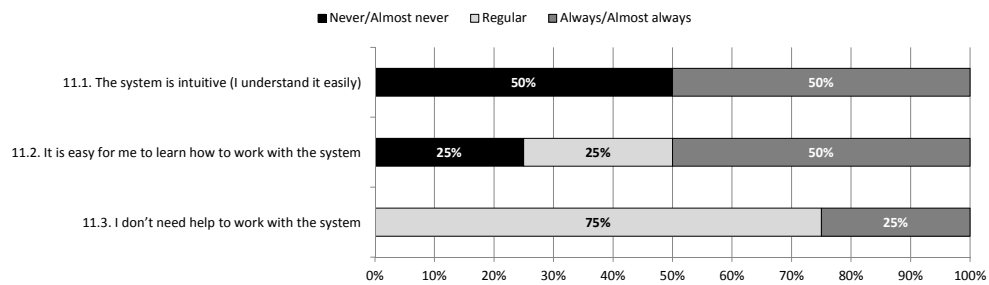


Figure 9.9: Results of the Ease of Use heuristic in the teacher profile.

9.3.2 Reliability

The reliability of a system is the ability of a system or component to perform its required functions under stated conditions for a specified period of time. In the final survey a reliability section was added in order to check on what tasks students and teachers had the difficulties. Figure 9.10 shows the results of the survey in this facet.

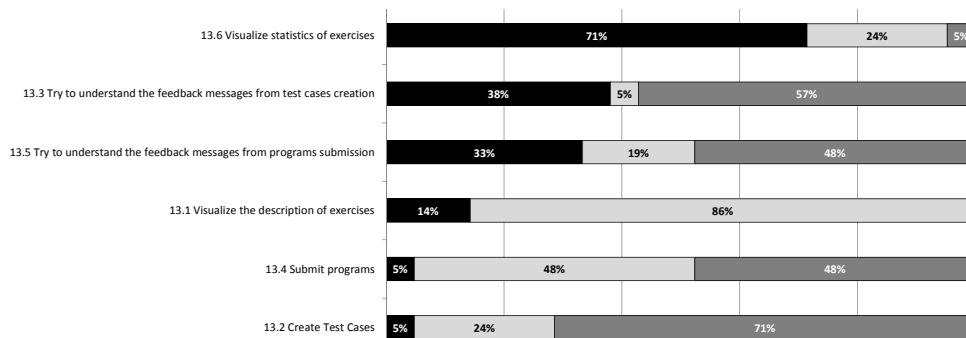


Figure 9.10: Reliability of Petcha.

Students reported difficulties in understanding error messages related to programs submission. Some users have left comments and suggestions, and the following were the most common:

- the message wording should be more friendly;
- the feedback should be more complete;
- the help system is insufficient for novice users;
- the exercise statistics should be more complete (e.g. time to solve, students rankings).

The comments and suggestions of the teachers were the following:

- the creation of exercises is not intuitive;

- the evaluation based on test cases should be more flexible in the comparison between the outputs generated by the student solution with the accepted outputs;

9.3.3 Interoperability

In the design of the experiment a network was deployed based on the Ensemble instance presented in chapter 5. This network is composed by several systems and services that need to interoperate to achieve a common goal. In this subsection the interoperability of this network is validated. Table 9.3 summarizes the communication between pairs of systems. These results were gathered from 6 lab sessions. In each session a class of 21 students had 3 exercises to solve. Based on these figures were computed the following expected values: the expected number of accesses to the system that is given by multiplying the number of students with the number of assignments ($21 * 6 = 126$); and the expected number of accesses to the exercises by all the students that is given by multiplying the number of students with the number of assignments and with the number of exercises by assignment ($21 * 6 * 3 = 378$).

Table 9.3: Statistical data on interoperability of the network components

Observation point	Expected	Real
# accesses to the system (LMS to TA)	126	135
# exercises requested to the repository (TA to LOR)	378	345
# exercises that students try to solve (TA to IDE)	378	342
# submissions (TA to AS)	378	819
# exercises requested to the repository (AS to LOR)	18	18
# exercises in which the students got feedback (AS to TA)	378	810

The first line of the table indicates that the system worked well since only nine extra sessions were used. These extra values were mainly due to the accidental closing of the application by students.

The number of exercises requested by the TA (Petcha) measures the number of times that students got an exercise statement from the repository. This action triggers an automatic request from Petcha to the repository. From the collected data it can be observed that not all exercises were actually read by all the students. There are two possible justifications: either the students did not have time to read all the available statements or some students may have given up after reading the exercise title. In any event students did not report any difficulty in accessing to problems

The third line of Table 9.3 is the number of exercises that students tried to solve. It is assumed that a student tried to solve an exercise when (s)he ran locally a set of tests to

validate the code. The real number is less than the expected and less than the number of exercise statement accesses. Most likely some students read an exercise statement, but did not have time to code a solution, or run the tests, or simply given up on solving it. The number of submissions is the number of requests for evaluation received by the AS. On average each student made approximately two submissions per exercise.

The number of exercises requests reflects the need of the AS to obtained the full LO from to repository given its reference. Since the AS has a cache mechanism the expected and real values are identical thus showing that the AS cache feature is working as expected and is accelerating the evaluation process.

The number of exercises in which the students got feedback should be similar to those of the number of submissions. Since they are almost identical (a difference of 9) one can conclude that the communication among the two systems (TA and AS) works well.

9.4 Summary

This chapter presents an acceptability evaluation on an Ensemble network for the computer programming domain. To carry out this evaluation an experiment was conducted in a higher education school. This evaluation focuses on three facets: usefulness, reliability and interoperability.

For the usefulness facet, a questionnaire based on the Nielsen's heuristics was filled in by students in the end of that experiment. On one hand, the results showed that the aesthetic was the heuristic with higher values of satisfaction. The respondents claim that the minimalist design is one of the strongest points of Petcha (the front-end of the Ensemble network). On the other hand, the results reveal deficiencies in three areas: flexibility, freedom and documentation. In the flexibility heuristic, the respondents felt difficulties in personalize the user interface and speed up the execution of frequent actions (e.g. definition of short-cuts). The freedom heuristic was another area where students were not satisfied since Petcha does not allow, for instance, the undo/redo of actions. Finally, students complained of the unavailability of supporting documentation while using Petcha. In the reliability facet students stated difficulties in understanding error messages and teachers complained about the lack of intuitiveness in the creation of exercises. For the interoperability facet, the communication between pairs of systems was analysed during the experiment by comparing the expected values with the real values (those obtained through service logs). The results show that the proposed e-learning framework is useful in practice. The figures collected during the experiment are within reasonable bound from the expected values. These results show that the network is stable enough to handle a significant number of students, and exercises and can be used in a classroom setting.

Chapter 10

Thesis validation

"Individually, we are one drop. Together, we are an ocean."

Ryunosuke Satoro

A set of observable assertions was stated at the beginning of this dissertation. These assertions are related with the impact of an Ensemble network in the teaching-learning process for domains with complex evaluation. This chapter validates these assertions with the use of an Ensemble network for the computer programming domain. The assertions are organized in four groups of hypothesis: exercises solving, feedback, attendance and grades. Then, each group of hypothesis is validated using statistical methods.

10.1 Hypothesis

Based on the goals formulated in the beginning of this dissertation a set of observable and testable assertions were defined related to the impact of the Ensemble network. These hypothesis are organized in four groups presented in Table 10.1.

Hypothesis validations relied on data collected in the experiment already described in the previous chapter. The data collection sources are surveys, service logs and grade-books. The following subsection validates these data. The remainder subsections seek to validate each group of hypothesis using statistical methods.

Table 10.1: Hypothesis on Ensemble usage.

Group	#	List of hypothesis.
Exercises solving	H1.1	Students start to solve more exercises
	H1.2	Students complete more exercises
	H1.3	Students effectively solve more exercises
	H1.4	Students maintain or improve the number of exercises started over time
	H1.5	Students maintain or improve the number of exercises completed over time
	H1.6	Students maintain or improve the number of exercises effectively solved over time
Feedback	H2.1	Students receive more feedback
	H2.2	Students receive more feedback to effectively overcome their difficulties
	H2.3	Students increasingly receive more feedback over time
	H2.4	Students increasingly receive more feedback that effectively overcome their difficulties over time
	H2.5	The automatic feedback provided is at least as good as the human feedback
Attendance	H3.1	Practical classes have more attendance
	H3.2	Practical classes maintain or improve attendance over time
Grades	H4.1	Students get better grades in the programming module
	H4.2	Students get better grades in the course

10.2 Data collection and validation

After each lab session both classes were surveyed on the number of solved exercises and the feedback impact. Table 10.2 aggregates the answers given by students. Data collected in the experimental class surveys was checked against the logs of Petcha and other systems in the network to attest the truthfulness of the survey responses. An average discrepancy of 4.6% between these two sets of values was found. This value suggests that students filled the survey carefully.

Firstly, statistical tests called Shapiro–Wilk were made to test if the samples came from a normally distributed population. After this validation, statistical hypothesis tests were used to assess whether the means of these two groups are statistically different from each other. The *t-test* is commonly used to verify if differences in observations can be explained by chance with a certain significance level p , typically 0.05 or 0.01. For instance, using the

Table 10.2: Survey results averages.

Assertions	Experimental group	Control group
Exercises started	89%	81%
Exercises complete	83%	74%
Exercises effectively solved	82%	66%
Exercises with feedback	59%	62%
Feedback helpfulness	55%	62%

.05 significance level means that one will accept as a real difference only those that occurred by chance only 5 times in 100 (i.e. in 95% of cases not due to chance). The statistical tests on the two sets of data (survey data and logs data) were satisfactory since the probability of the differences being the result of chance is higher than 5% ($p = 0.312 \Rightarrow p > 0.05$). Hence we cannot distinguish between the two averages since their difference most likely occurred by chance.

10.3 Results and discussion

This section includes for each group of hypothesis a subsection with the statistical results and their analysis. To prove the usefulness of the Ensemble network the analysis is made by student participation. A *participation* is defined as the performance of a single student in a single class. The reason for using participations and not simply count the exercises solved by students has to do with the fact that both classes have different numbers of students that would lead to misleading results.

10.3.1 Exercises solving

The first group of hypothesis is related with exercise solving. More precisely, it is hypothesized that in practical classes students that use the Ensemble network will start, complete and effectively solve more exercises and they will maintain or improve their performance over time. Figure 10.1 shows the average and standard deviation of exercises initiated, finished and effectively solved by student participation in both groups.

Figure 10.1 shows that the experimental group performed better than the control group because the mean score is higher and suggests that the use of the Ensemble network (the treatment) is effective. It must be ruled out that the differences in the mean between the experimental and control groups occurred by chance. A significance test determines whether the amount of difference between two groups is due to chance or due to the treatment.

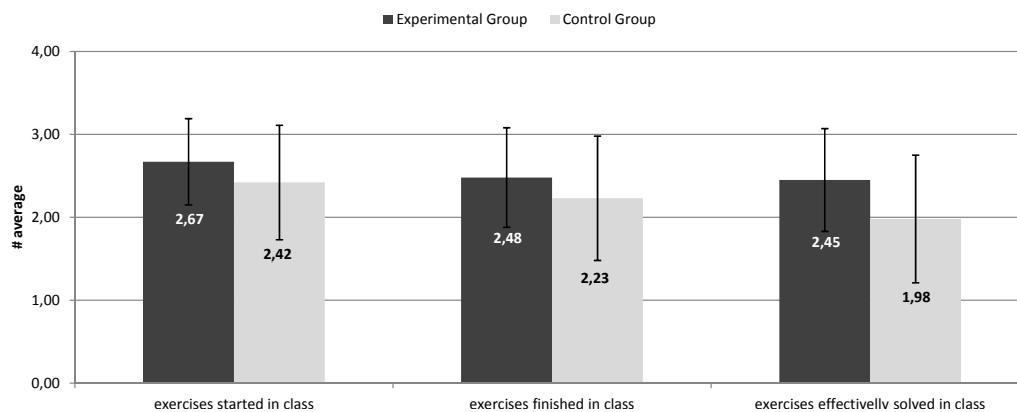


Figure 10.1: Average and standard deviation of exercises solving by student participation.

Table 10.3 shows the mean (M) and the standard deviation (SD) on student participation (n) in the two groups (experimental and control). It includes also a statistical hypothesis test called t-test to accept or reject the null hypothesis¹.

Table 10.3: Statistics on student participation.

Assertions	Experimental			Control			T-Test	
	n	M	SD	n	M	SD	T	p
Exercises started	116	2.67	0.52	88	2.42	0.69	2.857	0.004852
Exercises finished	116	2.48	0.60	88	2.23	0.75	2.617	0.009722
Exercises solved	116	2.45	0.62	88	1.98	0.77	4.678	0.000006

According to the results in Table 10.3, students who used the Ensemble network had a better performance than those who did not use it. The results for all the questions are significantly different since $p < 0.05$ and $p < 0.01$. This indicates that the difference is not due to chance but to the treatment (the use of the Ensemble network) and validates Hypothesis 1.1, 1.2 and 1.3.

Also the standard deviation in the experimental group is smaller than the one in the control group. This fact indicates that the number of exercises started/completed/solved is less dispersed in the experimental group suggesting a more homogeneous student performance when comparing with the control group.

Another goal in this experiment is to verify if the use of Ensemble influences the student motivation. Figure 10.2 depicts the evolution of the number of the exercises started in class.

Projections are also included in the chart using trendlines. Trendlines graphically display

¹The null hypothesis is accepted when has no statistical significance the difference between both groups.

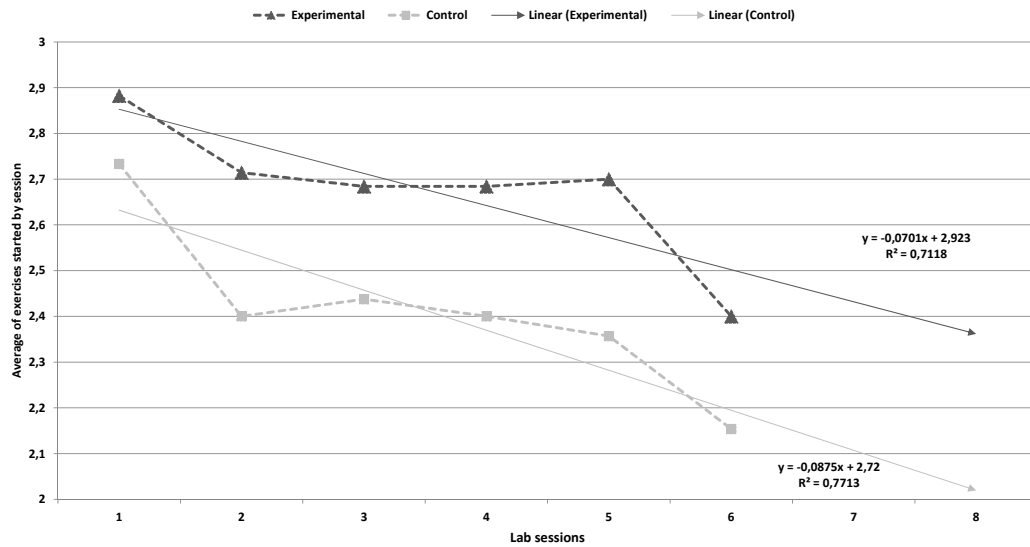


Figure 10.2: Evolution in started exercises.

trends in data series as part of regression analysis. By using regression analysis, one can extend a trendline in a chart beyond the actual data to predict future values. This graph has a linear trendline that is used with simple linear data sets. A linear trendline uses the following equation to calculate the least squares fit for a line: $y = mx + b$ where m is the slope and b is the intercept. The existence of linear trendline shows that something is increasing or decreasing at a steady rate. The chart also includes an R-squared value² that determines the reliability and accuracy of the trend projection. This is intrinsically related with the definition of correlation. Correlation is a statistical technique that shows whether, and how strongly, pairs of variables are related. The measure of a correlation is called the correlation coefficient (or "r"). It ranges from -1.0 to +1.0. The closer r is to +1 or -1, the more closely the two variables are related. If r is close to 0, it means there is no relationship between the variables. If r is positive, it means that as one variable gets larger the other gets larger. If r is negative it means that as one gets larger, the other gets smaller (often called an "inverse" correlation). If the correlation coefficient is squared, then the resulting value (a number from 0 to 1) will represent the proportion of common variation in the two variables (i.e., the "strength" or "magnitude" of the relationship). One can say that the R-squared value reveals how closely the estimated values for the trendline correspond to the actual data. A trendline is most reliable when its R-squared value is at or near 1. Based on this line chart several conclusions can be drawn:

- Students of the experimental group started more exercises over time compared with the control group. A significance test ($p = 0.02$) was made asserting that the differences

²Also known as the coefficient of determination.

are statistically significant. Thus validating with success Hypothesis 1.4;

- The first session had the highest results. One of the possible reasons is the lower complexity associated with the exercises of the first session. In the experimental group the mean of exercises started by students was of 2.88 (96%);
- Both groups decreased the number of exercises initiated over time ($m_{\text{exp}} < 0$ and $m_{\text{ctl}} < 0$). This is probably due to the increase of the exercises complexity;
- In the control group the decrease is slightly higher ($m_{\text{exp}} < m_{\text{ctl}}$). This means that students in the experiment group perform consistently and increasingly better than the control group.
- The correlation coefficient values ($r_{\text{exp}}=-0.84$ and $r_{\text{ctl}}=-0.88$) indicate that exists a dependence between the number of sessions and the number of exercises started in both groups. In this case an inverse correlation. Squaring the correlation coefficient values ($R^2_{\text{exp}}=0.71$ and $R^2_{\text{ctl}}=0.77$) reinforce the overall decrease of exercises initiated indicating that the projections are reliable (values are close to 1).

A significance test was made on the obtained correlations. For this test, the degrees of freedom ($df = 4$) were calculated based on the number of pairs of values less 2. Using a critical value table³ the minimum correlation coefficient r (0.811) was obtained. If the correlations values are greater than 0.811 or less than -0.811 (two-tailed test) one can conclude that the odds are less than 5 out of 100 that this is a chance occurrence. Since the correlations values (-0.84 and -0.88) are less than -0.811 one can conclude that it is not a chance finding and that the correlations are statistically significant. Thus, the null hypothesis (there is no relationship) is rejected and the alternative accepted. Figure 10.3 illustrates the same kind of analysis for the number of completed exercises and the number of effectively solved exercises. Several conclusions can be taken from this analysis:

- Students of the experimental group finished ($p = 0.013$) and solved ($p = 0.0001$) more exercises over time compared with the control group. Thus validating with success Hypothesis 1.5 and 1.6;
- The correlation coefficient values ($r_{\text{exp}}=-0.63$ and $r_{\text{ctl}}=-0.15$) for exercises finished and exercises solved ($r_{\text{exp}}=0.02$ and $r_{\text{ctl}}=-0.27$) indicate dependency between the number of sessions and the number of exercises finished and solve in both groups. In this case an inverse correlation. However, testing the significance of those correlations, probabilities greater than -0.811 (two-tailed test) are obtained. Thus, one can conclude that it is a chance finding and that the correlation is not statistically significant;

³<http://www.gifted.uconn.edu/siegle/research/Correlation/corrchrt.htm>

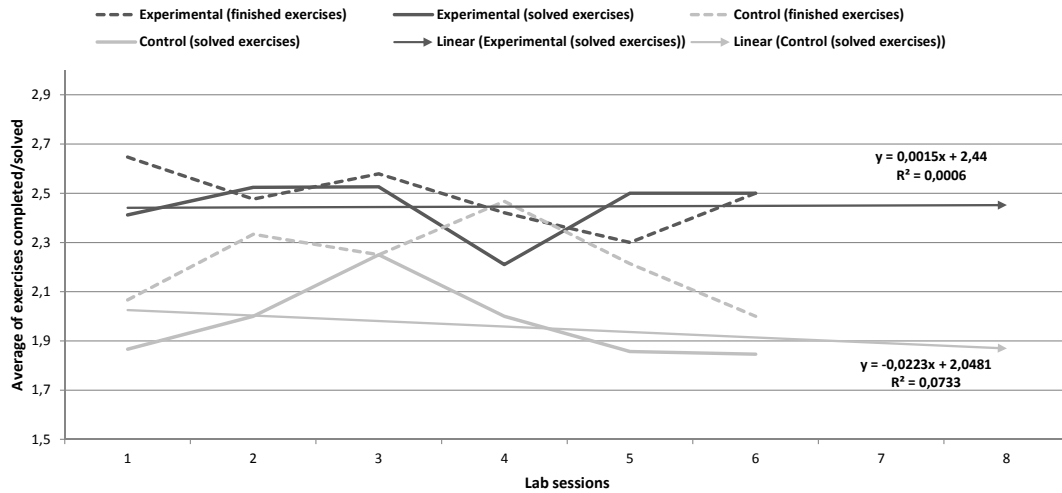


Figure 10.3: Evolution of completed/solved exercises.

- Squaring the correlation coefficient values for exercise solved ($R^2_{\text{exp}}=0.0006$ and $R^2_{\text{ctl}}=0.0733$) indicate that the projections are not reliable (values are far from 1). Although they are not reliable they point to a slight decrease ($m < 0$) in the control group and a slight increase ($m > 0$) in the experimental group;
- The correlation between finished and solved exercises of both groups gives the following values: -0.3953 and 0.0902. Apart from being not statistically significant the experimental values had an higher value then the control group.

10.3.2 Feedback

The second group of hypothesis is related with the feedback and its helpfulness to overcome the students difficulties on solving exercises. Figure 10.4 shows the average and standard deviation of the feedback received and its helpfulness by students participation.

Based on Figure 10.4 one can conclude that the control group had more feedback (and more helpful) than the experimental group since the mean score by participation is higher. In order to verify whether these samples are, or are not, significantly different a t-test was performed as shown in Table 10.4.

Regarding the first question and based on the significance level established ($p = 0.05$) the probability obtained shows that there is no significance differences between the means of both groups. Although there is little difference unfavourable to Ensemble this difference is not statistically significant and may be due to chance. Therefore, one can conclude that the feedback of Ensemble is at least as good as the feedback of the teacher.

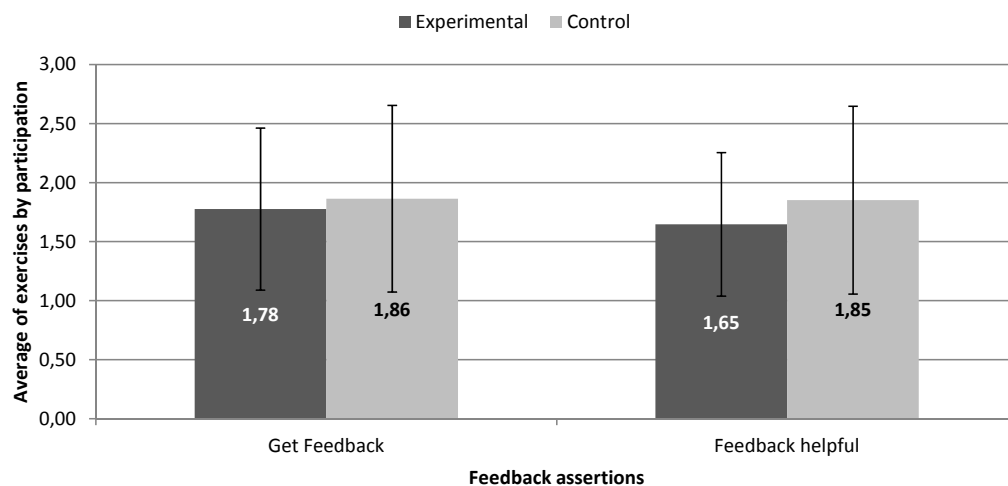


Figure 10.4: Average of feedback received by exercise and its helpfulness.

Table 10.4: Statistics on student participation.

Assertions	Experimental			Control			T-Test	
	n	M	SD	n	M	SD	T	p
Get feedback	116	1.78	0.69	88	1.86	0.79	-0.831	0.407
Feedback helpful	116	1.65	0.61	88	1.85	0.80	-2.019	0.045

In the second question the scenario is different since the probability value is lower than the significance level. This indicates that the difference is not due to chance but to the treatment. Therefore, one can conclude that the teacher's feedback is more helpful than the automatic feedback of Ensemble.

Figure 10.5 depicts the feedback received and its helpfulness to solve exercises over time. Several conclusions can be drawn from the this figure:

- The line chart suggests that students of the control group get more feedback and more helpful over time. However the significance test accept the null hypothesis based on the probability values for both sets of data ($p = 0.499$ and $p = 0.065$) revealing that there is not a statistically significant difference;
- The correlation coefficient values for obtained feedback ($r_{\text{exp}} = -0.29$ and $r_{\text{ctl}} = -0.75$) and feedback helpful ($r_{\text{exp}} = -0.22$ and $r_{\text{ctl}} = -0.68$) indicate that exists a dependence between the number of sessions and the feedback data in both groups. In this case an inverse correlation. However, testing the significance of those correlations, probabilities greater than -0.811 (two-tailed test) are obtained. Thus, one can conclude that the correlation is not statistically significant;

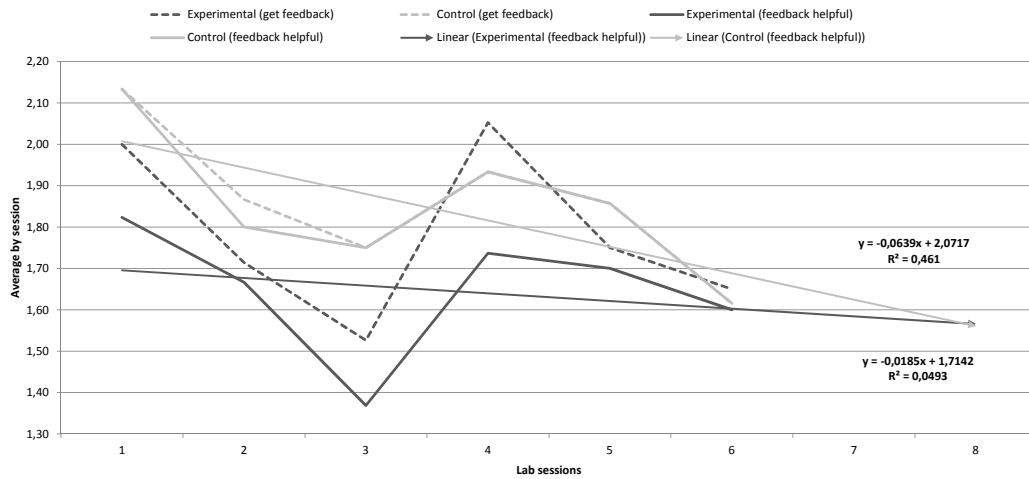


Figure 10.5: Evolution of feedback and its helpfulness.

- Squaring the correlation coefficient values for helpful feedback data ($R^2_{\text{exp}}=0.049$ and $R^2_{\text{ctl}}=0.461$) indicates that the projections are not reliable (values are far from 1). Although they are not reliable, both trendlines predict a decrease over time ($m < 0$). This decrease is more accentuated in the control group ($m=-0.0639$) comparing with the experimental group ($m=-0.0185$);
- The correlation between feedback received and feedback helpful on both groups gives the following values: 0.858 and 0.988. Both are greater than the minimum correlation coefficient r (0.811). Thus, one can conclude that both sets of data are dependent.

Regarding the hypothesis formulated one can say that only hypothesis 2.2 had statistically significant results. These results reject the hypothesis and confirmed that in a practical class students without using Ensemble receive more feedback that effectively overcome their difficulties. That was a predictable result since is very difficult to substitute the human feedback. All the tests for the other hypothesis resulted in the non-rejection of the null-hypotheses, i.e., results were not statistical significant. Arguably, one can say that the automatic feedback provided by Ensemble is at least as good as the human feedback and, for that reason, validate with success the Hypothesis 2.5.

10.3.3 Attendance

Practical class attendance was optional in this course. Hence, class attendance reflects the level of motivation of the students. Figure 10.6 shows the overall attendance of experimental and control groups. The overall attendance is higher on the experimental group. In fact, 92% of participations (116 of 126 possible participations) were registered against 77% of

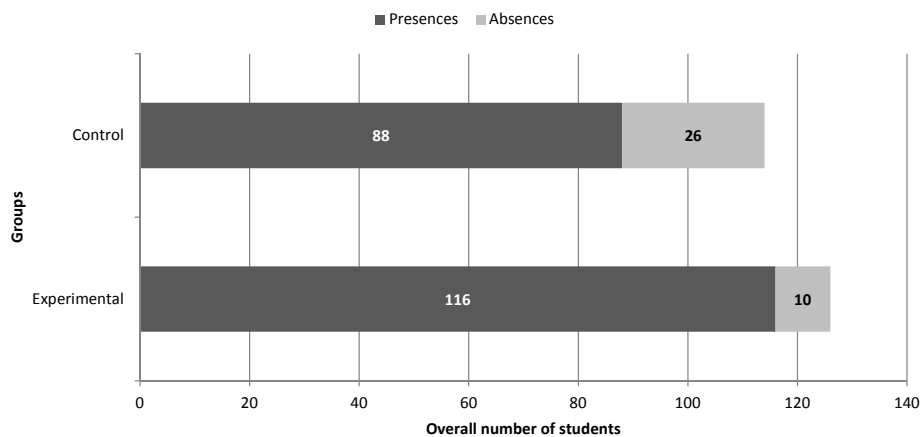


Figure 10.6: Overall attendance of students of both groups.

participations (88 of 114 participations) in the control group. Using a t-test (Table 10.5) the p value is 0.00009 ($p < 0.05$) which rejects the null hypothesis and indicates the difference between both sets. Thus, the first conclusion is that the experimental group was more assiduous than the control group which validates the hypothesis 3.1.

Table 10.5: Statistics on student attendance.

Assertion	Experimental			Control			T-Test	
	n	M	SD	n	M	SD	T	p
Attendance	116	19,33	1,366	88	14.66	1.032	6.674	0.00009

A regressive analysis was also performed on the evolution of attendance. Figure 10.7 shows the evolution attendance percentages of both groups. Several conclusions can be drawn:

- The correlation coefficient values for attendance ($r_{\text{exp}}=0.47$ and $r_{\text{ctl}}=-0.72$) were used to test their significance. The former is lower than 0.811 and the latter is greater than -0.811 (two-tailed test). Thus, one can conclude that the correlation is not statistically significant;
- Squaring the correlation coefficient values ($R^2_{\text{exp}}=0.2204$ and $R^2_{\text{ctl}}=0.525$) indicates that the projections are not reliable (values are far from 1). Although they are not reliable, both trendlines are asymmetric predict a decrease ($m = -0.019$) over time for the control group and an increase ($m = 0.0163$) over time for the experimental group.

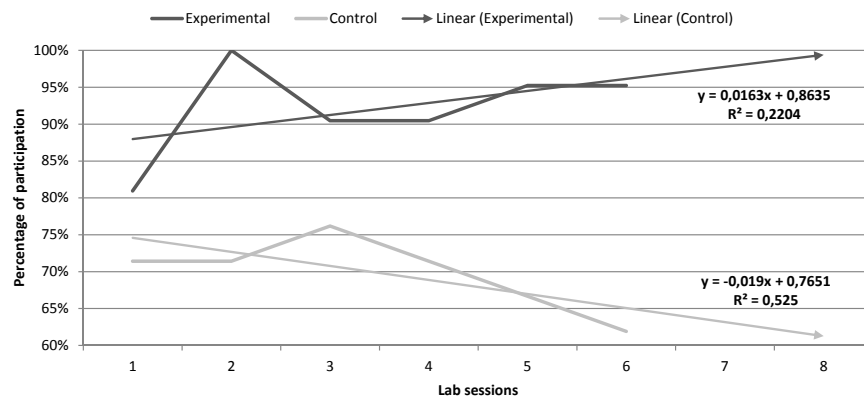


Figure 10.7: Evolution of attendance.

10.3.4 Grades

The last group of hypothesis is related with grades. In order to verify these hypothesis was used a grade-book of the two assessment moments after the experiment: at the end of the programming module and at the end of semester. Figure 10.8 presents the average grades on the two moments for both groups.

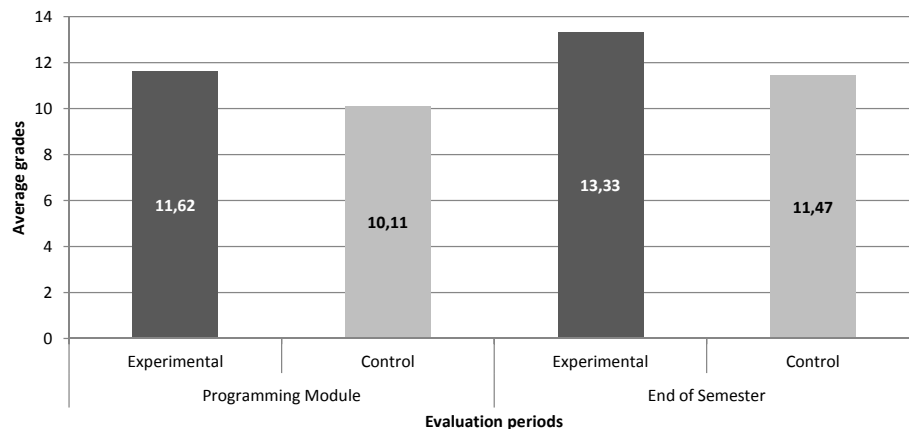


Figure 10.8: Average grades from both groups of students.

Using a t-test (Table 10.6) to infer the statistical significance one can conclude that in the Programming module assessment the means of both groups are statistical different since the probability of a chance finding is lower than the significance level ($p = 0.041$ and $p < 0.05$). This fact rejects the null hypothesis (the two sets are statistically equals) and indicates the difference between both sets. Thus, the first conclusion to infer is that the experimental group has better grades than the control group on the Programming module.

In the second assessment moment the probability of a chance finding is also lower than the

Table 10.6: Statistics on student participation.

Assertions	Experimental			Control			T-Test	
	n	M	SD	n	M	SD	T	p
Programming module	21	11.62	2.75	19	10.10	1.66	2.130	0.041
End of Semester	21	13,33	2,35	19	11.47	2.61	2.357	0.024

significance level ($p = 0.024$ and $p < 0.05$). This fact rejects the null hypothesis and one can conclude that the experimental group has better grades than the control group in the end of semester as result of using Ensemble. Also a paired t-test was computed from the grades of the experimental group. Dependent samples (or "paired") t-tests typically consist of a sample of matched pairs of similar units. In this case the goal is to infer if the difference of means in both periods of the experimental group are caused by the use of Ensemble. With the paired t-test a probability value of 0.0011 was obtained proving the influence of treatment in the difference of the means. Thus, proving the influence of Ensemble on the increasing of the mean grades of the students in the experimental group. These figures validate with success the hypothesis 4.1 and 4.2.

10.4 Summary

This chapter validates most of the hypothesis formulated at the beginning of this dissertation with an Ensemble network for the computer programming domain. The assertions are organized in four groups of hypothesis: exercises solving, feedback, attendance and grades. Then, each group of hypothesis is validated using statistical methods. For the exercise solving group all the hypothesis were validated with success proving that with Ensemble the students start, complete and effectively solve more exercises. For the feedback group the results confirmed that in a practical class students without using Ensemble receive more feedback that effectively overcome their difficulties. All the other hypothesis had results that were not statistical significant. Arguably, one can say that the automatic feedback provided by Ensemble is at least as good as the human feedback. The attendance results prove that students from the experimental group are more assiduous than the control group. However no statistical conclusion can be drawn on the evolution of attendance in the practical classes. The results for the grades group prove the influence of Ensemble on increasing of the mean grades of the students in the experimental group.

With these figures one can conclude that the helpful feedback is the only hypothesis that was not validated with success. This was more or less expected since it is very difficult to replace a human TA. Improving automated feedback is a challenge to be tackled as part of the future work.

Chapter 11

Conclusions

"A conclusion is the place where you got tired of thinking."

Martin H. Fischer

Learning complex skills is hard. Introductory programming courses are generally regarded as difficult and often have high failure and dropout rates [AM05, OG06, RRR03]. Many educators claim that "learning through practice" is by far the best way to learn computer programming and to engage novice students [GP05, Eck09]. Practice in this area boils down to solving programming exercises. Nevertheless, solving exercises is only effective if students receive an assessment on their work. Assessing the work of students and providing individualised feedback to all students is time-consuming for teachers and frequently involves a time delay. The existent tools and specifications prove to be insufficient in complex evaluation domains where there is a greater need to practice [RKK04].

This thesis proposes an e-learning framework - called Ensemble - that acts as a conceptual tool in the definition and deployment of e-learning networks using complex evaluation. The framework relies on interoperability standards and specifications, thus several studies and surveys were conducted to select the most relevant for the framework. Based on this framework a network of systems and services was created and deployed for a specific domain - the computer programming domain. Content issues are tackled with a standard format to describe programming exercises as learning objects. Communication issues are addressed with the development or adaptation of systems and services for managing the life-cycle of exercises, namely their authoring, storage, conversion and assessment.

The framework instance was deployed for use in practical classes of undergraduate programming courses. The experience gained using Petcha (the pivot component) in this context and the experiments designed to assess the impact of this tool were also presented in this

dissertation. These experiments showed an increase on exercises solving, attendance and grades when Petcha replaced a human Teaching Assistant (TA). However, these results show also that the automatic feedback provided by Petcha is less effective than that of a human TA. There is clearly room for improving automatic feedback in Petcha, although it can be argued that automated feedback is still a remedy for situations where a human TA is not available.

11.1 Contributions

The main contribution of this work is a conceptual model - the Ensemble framework - for the definition and deployment of e-learning networks using complex evaluation. The architectural model of this framework relies on central components (called axial systems) replicated for each teacher and student machines. One of these central components assumes a pivot role orchestrating all the communications within a single deployment of an Ensemble instance. Since it is distributed over each network user, this approach prevents any single-point-of-failure issues that might occur. This pivot component communicates locally with other axial systems and remotely with core and secondary services representing a distinctive feature regarding other e-learning frameworks. Part of this main contribution is the specialization of the framework for a specific domain - the computer programming domain. This framework instance comprises several systems and services and their integration poses interoperability issues at two levels: content and communication. The content interoperability relies on the definition of an interoperability language for programming exercises called PExIL. The communication among systems and services was supported by the extension of existing specifications (e.g. IMS DRI, IMS LTI) and the creation of new ones (e.g. Evaluate service).

Other important contribution is the systematic study on the state of the art regarding e-learning systems and standards. This comprehensive study focuses on several surveys presented in the state of the art that were instrumental to chose the better e-learning systems and standards for the Ensemble framework. This contribution may prove helpful to other researchers studying the interoperability of e-learning systems.

The remaining contributions are related with the design and implementation of components that comprises the Ensemble network for the computer programming domain. These components are the crimsonHex repository (including a plug-in for accessing crimsonHex based repositories from Moodle), the Petcha teaching assistant and the BabeLO exercises converter. All these components are open source and can be downloaded from the following URL: <http://ensemble.dcc.fc.up.pt>.

Most of the research presented in this dissertation was already published in international journals and conferences with peer reviewing. A total of 35 articles was published:

- Five articles in international journals;
- Five chapters in international books;
- Eleven papers in international conference proceedings with main publisher¹;
- Thirteen papers in national and international conference proceedings with peer reviewing without a main publisher;
- One technical report.

Nevertheless, part of this research is still unpublished and a few remaining publications are expected to be submitted by the end of 2012. In particular, the thesis validation results were not yet published.

11.2 Opportunities for future work

The motivation for this research was drawn from the computer programming domain. However, it was always kept in mind that the proposed concepts and tools could be used in other domains. The main opportunity for future research comes from extending this framework to other domains and requirements. Nevertheless, the evaluation of Ensemble and the validation of the thesis highlighted a number of issues that must be resolved in the computer programming instance of Ensemble.

11.2.1 Framework validation

The main challenge resulting from this research is to apply the framework to other domains. Although the research hypothesis were in general validated, the framework as such is not yet validated since it was only applied to a single domain. One interesting domain is serious games applied to management courses where students develop their skills using simulation. Business simulation games improve the strategic thinking and decision making skills of students in several areas (e.g. finances, logistics, and production). Through these simulations students compete among them as they would in a real world companies. A business simulation service fulfils a role similar to that of the assessment systems in programming exercises and it also requires a repository containing specialized LO describing simulations. Thus, this specific domain poses challenges not only in the development of the network TA , but also in the refinement of the framework specifications and services (e.g. repository, assessment system) to meet the new evaluation domains requirements.

¹ACM, IEEE or Springer.

11.2.2 Framework extension

The current version of the framework focuses mainly on exercise authoring, exchange and evaluation. Other kinds of data and services should be added to improve the practice-based environments supported by Ensemble, both in the computer programming domain and in new domains.

Plagiarism checker - this component can be added to the framework to avoid plagiarism and ensure good scholarly practices. This tool is transversal to several areas and is therefore a good candidate to integrate the Ensemble framework;

Sequencing component - Sequencing of exercises is another topic that can be explored in the future and it is closely related with pedagogical issues during the construction of a learning scenario. Several standards appeared in recent years trying to cope this topic but fail due its complexity for e-Learning systems to implement. One research path is to deliver exercises to students dynamically according with their profiles, knowledge evolution and course goals. An intended addition is a sequencing and adaptation tool to guide the student through a collection of expository and evaluation resources. The network pivot component will report the exercise assessment to this new tool that will use it to propose the appropriate content or exercise to the student;

E-Portfolio - this is a special type of a repository where a collection of electronic evidence is assembled and managed by a user. They are distinct from LMSs since they are user-centric rather than course-centric. The integration of such tool in the Ensemble framework can be achieved at content or communication level through data (e.g. LeapA specification) or tools integration (e.g. IMS LTI specification);

Standards and specifications - support for other LO package specifications (e.g. SCORM objects and for MathJax for displaying math expressions in the description of exercises.

11.2.3 Ensemble instance improvements

The computer programming instance of Ensemble is currently being used in the practical classes of undergraduate programming courses at ESEIG and will continue to be used in the next academic year. Several improvements are planned for immediate implementation based on the suggestions of teachers and students after the experiment. These improvements focus on Petcha - the visible system of the network, and include:

User interface - make the GUI more intuitive and flexible;

Evaluation reports - improve the visualization of the evaluation reports using new formats (e.g. PDF);

Statistics - improve statistical data on student activity (e.g. time to solve, rankings);

Help - extend the documentation to guide users;

In general, the improvements presented previously are minor issues that should be easily fixed for the next version of the Ensemble instance. There is also a collection of new features that would improve automatic assessment but that will require a major redesign of the AS.

Feedback - improve the feedback mechanism based on, for instance, the use of static analysis over the students' code. Existing work in this area [NNH99] can be used to improve the feedback given to students after submission.

New evaluation models - a programming problem definition must have an unambiguous evaluation model. Typically a program from a student is assessed by the evaluator as a single program. Another approach is the student code be included within a set of programs from different learners for competitive evaluation. A third approach is where several programs from different learners are evaluated simultaneously interacting with a central component (an "oracle") also in a competitive fashion. For instance, in the Tic Tac Toe game the student's program plays the game against the oracle;

Other types of languages - the current evaluator can be configured for any programming language with a command line interface and processing standard input/output. There are computer languages that are not strictly programming but are regularly used in computer science courses such as query languages (e.g. SQL), modelling languages (e.g. UML) and user interfaces (e.g. HTML). In most cases these languages can be evaluated statically by comparing the submitted source code with the solution.

Appendix A

Nielsen's heuristics

1. **Visibility of system status** - The system should always keep users informed about what is going on, through appropriate feedback within reasonable time;
2. **Match between system and the real world** - The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order;
3. **User control and freedom** - Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo;
4. **Consistency and standards** - Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions;
5. **Error prevention** - Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action;
6. **Recognition rather than recall** - Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate;
7. **Flexibility and efficiency of use** - Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions;

8. **Aesthetic and minimalist design** - Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility;
9. **Help users recognize, diagnose, and recover from errors** - Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution;
10. **Help and documentation** - Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

Appendix B

Session survey

Questionário de satisfação do Petcha

Este questionário tem por objectivo recolher informação do aluno sobre a utilização do Petcha.

***Required**

1. Visibilidade

Visibilidade do estado do sistema (feedback apropriado em tempo razoável)

Quando solicito ajuda ao sistema a resposta é clara *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Quando executo uma tarefa o sistema informa sobre o que está a acontecer *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Os botões usados para realizar as tarefas mais importantes estão claramente identificados *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

O estado dos botões (seleccionado/não seleccionado) é indicado com clareza *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre

☐ Não Aplicável

Comentários

2. Compatibilidade

Compatibilidade entre o sistema e o mundo real (linguagem familiar ao utilizador)

Quando tento executar uma tarefa encontro rapidamente o botão pretendido *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

A ordem dos botões está numa sequencia familiar *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Quando pressiono um botão o resultado é o esperado *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Os comandos agrupados num menu pertencem todos à categoria indicada pelo texto do botão desse menu *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre

- ☐ Sempre
- ☐ Não Aplicável

Comentários

3. Liberdade

Liberdade e controlo do utilizador (deve ser possível ao utilizador desfazer ou refazer operações)

Quando cometo um erro o sistema permite voltar atrás *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Posso interromper uma acção e retomá-la em qualquer instante *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Posso cancelar uma operação que está a decorrer *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Posso eliminar qualquer alteração que está a ser feita e voltar ao estado anterior *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre

- ☐ Sempre
- ☐ Não Aplicável

Comentários

4. Consistência

Consistência e padrões (o sistema deve ser compreendido indubitavelmente e seguir as convenções conhecidas)

A localização de botões e janelas é mantida quando mudo de ecran *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Os botões possuem sempre o mesmo significado quando mudo de ecran *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

O significado dos códigos de cores é consistente *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

É possível o uso do scroll em todas as janelas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular

- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários

5. Prevenção

Prevenção de erro (evitá-los)

O sistema avisa quando ocorrem problemas de entrada de dados, antes de eu executar a validação *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Quando faço uma validação, o sistema apresenta uma mensagem de erro se o formato de dados não é o esperado *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Sou avisado pelo sistema se estou prestes a cometer um erro grave *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Existe uma separação clara entre os botões que possibilitam o acontecimento de erros graves e os restantes botões *

- ☐ Nunca

- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários

6. Ênfase

Ênfase no reconhecimento (minimizar o esforço de memória do utilizador tornando os objectos, as acções e as opções sempre presentes)

As cores usadas nos textos estão conforme as convenções aceites para o seu significado *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

O texto contido em cada botão transmite a ideia do que é esperado *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

A informação contida no ecrã encontra-se disponível no local onde eu espero *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Os itens encontram-se agrupados por género em zonas lógicas distintas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários

7. Flexibilidade

Flexibilidade e eficiência no uso (deve ser permitido ao utilizador personalizar ou programar acções frequentes - ex: criar atalhos)

Consigo configurar o ecran *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Existem teclas de atalho para executar as funções mais utilizadas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Consigo desactivar temporariamente alguma(s) função(ões) *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários**8. Estética**

Estética e desenho minimalista (só informação indispensável)

A informação contida no ecrã é somente a que preciso **Consigo desactivar temporariamente alguma(s) função(ões) ***

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

A informação contida no ecrã destaca-se do fundo *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Esteticamente o sistema é agradável nos factores: cores, brilho, etc. *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários**9. Ajuda Utilizadores**

Ajuda os utilizadores a reconhecer, diagnosticar e recuperar de erros (as mensagens de erros

devem ser claras e sugerir soluções)

As mensagens de erro/ajuda são claras e adequadas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

As mensagens de erro indicam o problema com precisão *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

As mensagens são curtas e objectivas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários

10. AjudaDocumentação

Ajuda e documentação (documentação sempre disponível)

Pesquisa facilmente a informação *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre

☐ Não Aplicável

A função de ajuda é bem visível *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

A informação é precisa, completa e perceptível *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários

11. Facilidade

Facilidade na Aprendizagem

O sistema é intuitivo (percebo-o facilmente) *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Tenho facilidade em aprender a trabalhar com o sistema *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre

☐ Não Aplicável

Não necessito de ajuda para trabalhar com o sistema *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários

12. Velocidade

Velocidade de resposta do sistema

O tempo de resposta para as operações realizadas é suficientemente curto *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

O tempo de resposta na mudança de tarefas é suficientemente curto (ex: está a ver “pendentes” e altera para “impressões”) *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários

13. Fiabilidade

Fiabilidade das suas funções

Com que frequência encontro dificuldades quando...

me registo no sistema *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

vizualizo o enunciado dos problemas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

submeto programas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

tento compreender as mensagens de erro resultantes da submissão de programas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

acedo à lista de soluções dos problemas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular

- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

esclareço questões com o júri *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

visualizo questões colocadas ao júri pelos restantes concorrentes *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

visualizo a minha classificação em concurso *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

visualizo a avaliação/classificação dos problemas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

imprimo programas *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular

- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

pretendo obter informações relativas aos restantes concorrentes *

- ☐ Nunca
- ☐ Quase nunca
- ☐ Regular
- ☐ Quase sempre
- ☐ Sempre
- ☐ Não Aplicável

Comentários

14. Classificação

Atendendo a todos os parâmetros que analisou como classificaria o Petcha *

- ☐ Muito Bom
- ☐ Bom
- ☐ Suficiente
- ☐ Insuficiente
- ☐ Mau

Submit

Powered by [Google Docs](#)

[Report Abuse](#) - [Terms of Service](#) - [Additional Terms](#)

Appendix C

Petcha's user manual

Petcha is an automated Teaching Assistant (TA) with two main tasks: to assist teachers in authoring exercises and to help students in solving them. The installation instructions can be found in the following URL: <http://ensemble.dcc.fc.up.pt/Petcha>. This chapter presents an user manual both for teachers and students.

C.1 Teacher's user manual

Petcha allows teachers to create programming exercises and publish them in public repositories using standard package formats. This manual explains step by step this process.

C.1.1 Launching Petcha

Petcha (currently in its version 0.9) runs as a JAVA application. After installing, you can start using Petcha by accessing this URL: <http://your.machine/Petcha>. This manual shows how to create a programming exercise to calculate the transposition of a matrix. The first task to perform is to create a project.

C.1.2 Creating a project

In order to create a new project go to the main menu by choosing *Project* → *New*. The following fields are mandatory (Figure C.1):

- Project Name - name of the project to be automatically created in the IDE;

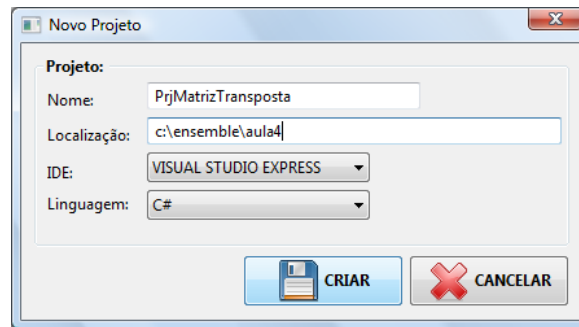


Figure C.1: New project.

- Location - location of the project (the workspace for Eclipse IDE and the solution for Visual Studio Express IDE);
- IDE - preferred code editor for coding and testing the program solution;
- Language - programming language used to solve the exercise.

After filling in all the fields one must press the *Create* button. A confirmation screen appears (Figure C.2). In case of success a project is created in the teacher's machine.

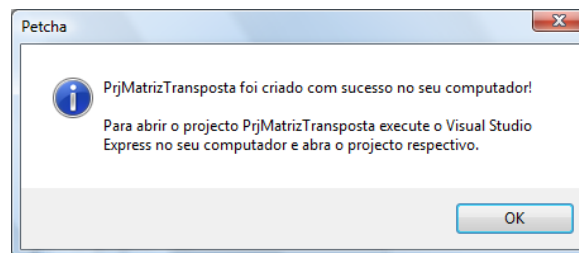


Figure C.2: Confirmation of the creation of the project.

C.1.3 Writing a program solution

After the creation of the project the teacher should open the generated project file of the selected IDE (in this case the Visual Studio Express - VSE). With the IDE open the teacher should click on the predefined class file (in CSE is the Program.cs) and start to code the program solution as depicted in Figure C.3.

While codifying the program solution the teacher should be aware that writing to the standard output should only occur at the end of the code at the time of presenting the results and should not include supplementary text.

Figure C.4 shows the code that calculates the transposition of a matrix.

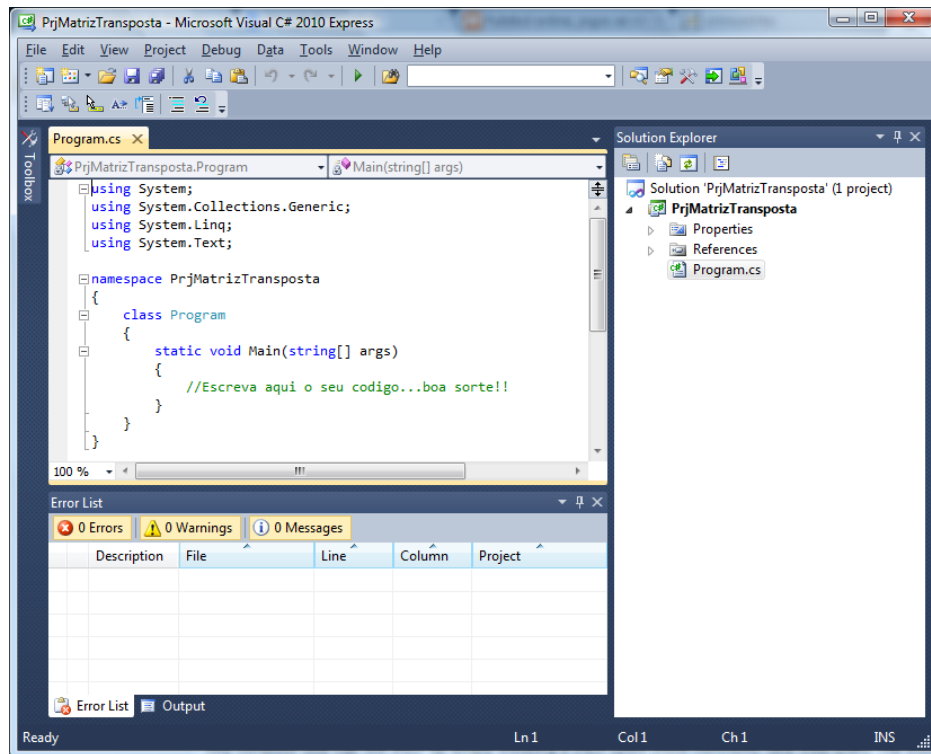


Figure C.3: Coding the program solution in VSE.

After coding the program the teacher should return to Petcha to continue the description of the exercise. Petcha is organized in four areas (Figure C.5):

1. Description - data related to the exercise and that will appear in its statement;
2. Tests - test cases to be used by the Assessment System (AS) to evaluate the student attempts;
3. Feedback - definition of the feedback levels to be include in the evaluation report delivered to the student after submission;
4. Publication - packaging and deploying of the programming exercise in a repository.

C.1.4 Defining the exercise statement

In the *Description* tab (Figure C.6), the teacher must include all the information necessary to describe the exercise. This information will be used by Petcha to generate automatically the exercise statement in several formats (e.g. PDF, HTML) to be showed to the student. The mandatory fields are:

- Title - title of the exercise to be included at the top of the statement;

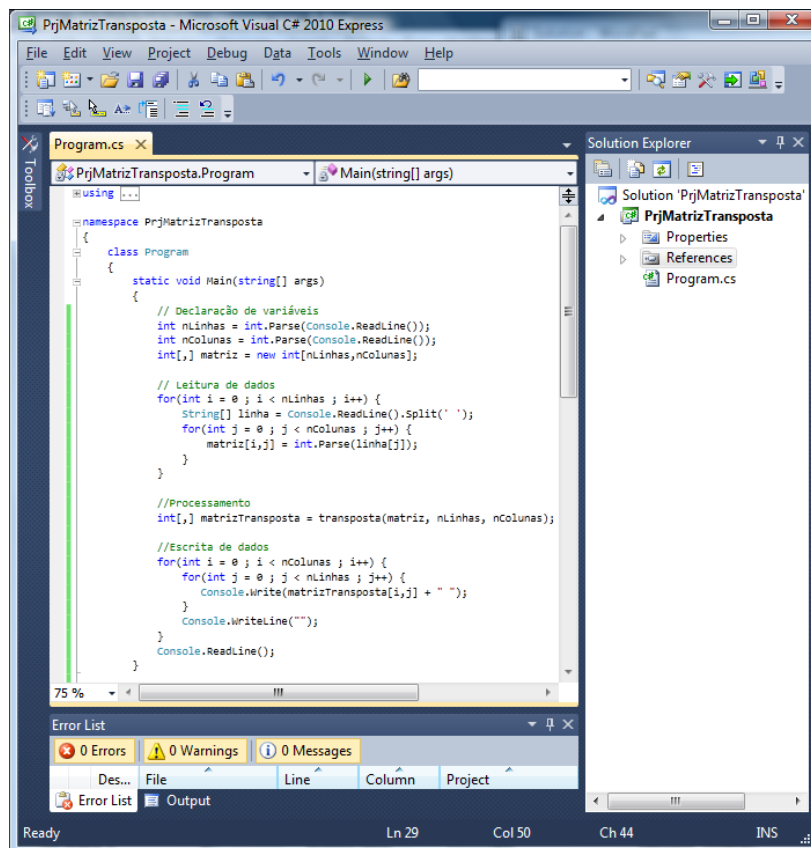


Figure C.4: Matrix transposition code.

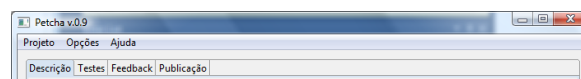


Figure C.5: User interface areas of Petcha.

- Authors - creators of the exercise. In the case of multiple authors their names should be separated by commas;
- Date - date of creation of the exercise;
- Institution - where the exercises will be used;
- Event - in what context the exercises will be used;
- Context - brief overview of some important concepts for the resolution of the exercise;
- Challenge - description of the problem to be solved;
- Keywords - terms that summarize the exercise. For multiple terms use spaces as delimitation.

Petcha v.0.9

Projeto Opções Ajuda

Descrição Testes Feedback Publicação

Metadados:

Título: Matriz transposta

Autor(es): Ricardo Queirós (múltiplos autores separados por vírgulas)

Data: 24/10/2011 Instituição: ESEIG Evento: AP

Contexto: A transposta de uma matriz A do tipo $m \times n$ designa-se por A^t e obtém-se trocando ordenadamente as linhas pelas colunas de A . A operação de obtenção de uma matriz transposta de A é denominada transposição da matriz. Observe o exemplo: `<html:br/>` `<html:img alt='imgTransposta.jpg' src='imgTransposta.jpg'/>`

Desafio: Escreva um programa que dada uma matriz retorne uma matriz equivalente à sua transposição.

Palavras-Chave: matrizes arrays (separadas por espaços)

HTML BR HTML IMG VISUALIZAR

Figure C.6: The Description tab.

The fields *Context* and *Challenge* allows the inclusion of certain HTML tags such as the `<html:br>` and `<html:img>`. The first makes a line break. The second allows the teacher to include a picture. By clicking in the respective buttons the respective HTML code is generated. For the image selection a pop-up window appears. The image formats supported are PNG, GIF or JPG.

At any time of the creation of the exercise the teacher can view the statement by clicking in the *Preview* button. Figure C.7 shows the PDF generated based on data filled until that moment.

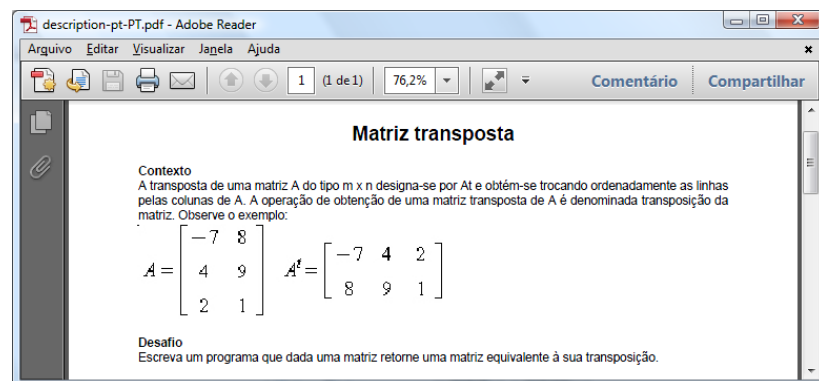


Figure C.7: Exercise statement.

C.1.5 Defining and generating the test cases

The *Tests* tab comprises two groups: *Program Solution* and *Test Cases*. The former shows information on the program solution (Figure C.8), namely:

- Path - location of the program solution (according to location and project name);
- Language - programming language used to code the program solution;
- Version - version of the language;
- Compiler - associated language compiler (csc for C#, javac for JAVA, etc.);
- Executor - command used to run the compiled file (for C# does not exist because the program is executed by invoking its own name);
- Source file - the name of the file containing the program solution (prior to compilation);
- Executable file - the name of the file containing the program solution (after compilation).

Programa Solução:	
Caminho:	c:\ensemble\aula4\PrjMatrizTransposta\PrjMatrizTransposta\Program.cs
Linguagem:	C#
Versão:	
Compilador:	csc
Executor:	
Ficheiro fonte:	Program.cs
Ficheiro executável:	Program

Figure C.8: Data about the program solution.

All previous data is automatically generated in the group *Program Solution*. Thus, the teacher should only confirm that everything is correct.

The *Test Cases* group is used to define test cases. A test case consists of a maximum of three files with the input data of the program, the correspondent output data and the feedback message that appears if the student fails the test. The test cases will be used by the AS to evaluate the attempt of the students. The evaluation process is straightforward:

1. The teacher provides the solution of the program and several valid test cases;
2. The student solves the exercise and submit its solution to the AS;
3. The AS uses the student's program and the test cases supplied by the teacher. For each test case the AS runs the input data in the solution of the student, and verifies if the data returned (output) are equal to those supplied by the teacher. If all tests run successfully the attempt of the student is considered correct. If any test fails the attempt of the student is found to be incorrect;

4. the AS sends an assessment report to the student notifying them of the success/failure of its attempt.

The test cases can be created manually by the teacher or generated automatically by Petcha. The management of the test cases is made in the *Test Cases* group as depicted in Figure C.9.

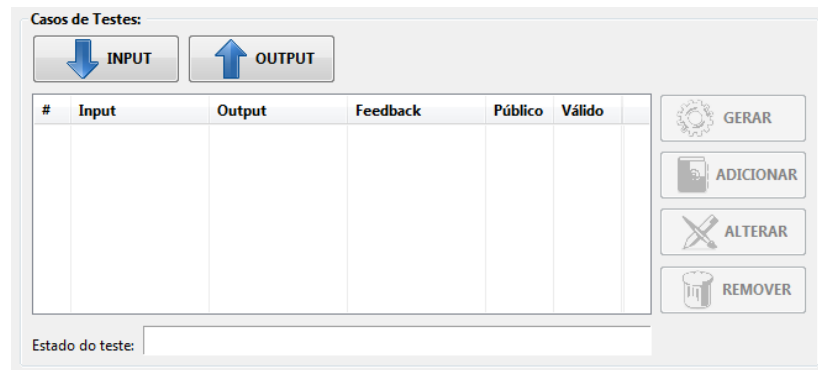


Figure C.9: Test cases management.

There are six buttons:

- Input - defines an input specification;
- Output - defines an output specification;
- Generate - generates automatically a set of test cases according to the definitions of input/output;
- Add - adds a test case manually;
- Change - changes an existing test case;
- Remove - removes an existing test case.

For Petcha to be able to automatically generate a test case is necessary to describe what is the input/output (for instance, how many input/output parameters and what type of parameters the program read/write). This description can be configured by clicking in the Input/Output buttons included in Figure C.9.

In the new window the teacher should define a brief (but strict) description on the input/output that the program must receive/return. This text will appear in the exercise statement. The teacher should also define a set of elements that describe the structure and content of the input/output data. The main elements are:

- Line - defines a row in the input/output file;

Figure C.10: Description of the input data.

- Data - defines a variable included in a row. There may be several variables. A variable can be characterized by several attributes (identified below);
- Repeat - defines a repetition of rows or variables. This repetition is controlled by the attribute Cont (counter);
- When - defines a feedback message resulting from the evaluation of a particular condition.

The Data element is the central element of the test cases specification. The following attributes can be associated to this element:

- Id - defines a name for the variable. in some cases, to access the variable (e.g. count) the teacher should precede its reference with a \$;
- Type - define the data type of the variable (e.g. integer, float, string, enum). In the case of choice of enum should add/remove values of the enumeration using the +/- buttons;
- Value - represents the value of the variable to be included in the input/output file. If filled in it acts as a constant. Otherwise the value is automatically generated based on a set of constraints (e.g. attribute types, min/max, spec);
- Min/Max - represents the value limits. The semantic of this attribute depends solely on the data type. If the data type is integer or float then these attributes represent the range of valid values for the variable. If the data type is a string then these attributes

represent the range of number of characters that the string can contain. If the data type is enum then these attributes represent the number of values that can be selected from the list;

- Spec - defines a regular expression for generation/matching a string.

In order to define the structure and content of the input files for this exercise it is necessary to define two elements Line including elements Data to accommodate the number of rows and columns of the matrix. Then one must use two elements Repeat (one inside the other) whose counters should reflect the number of rows/columns and include an element representing the Data variable will assume different values along the fill of the matrix.

To add an element (Figure C.11) one should select the item in the list box and then press the *Add* button. Note that to add an element within another one must first select the parent element. It is possible always to change a particular element selecting the element on the table of elements and then change the specific attribute and finish by clicking in the *Save* button. The *Remove* button removes a selected element. Note that this action also removes all the descendants of the element to remove. In order to add top elements one should always remove the focus of the element selected in the table of elements by clicking in the *Deselect* button.

Editor de especificações para testes

Especificação:
 Descrição: O programa deve receber três parâmetros. Na primeira linha um inteiro (entre 1 e 10) indicando o número de linhas da matriz. Na segunda linha um inteiro (entre 1 e 10) indicando o número de colunas da matriz. O terceiro parâmetro é a própria matriz onde as

Terminador de linha: Separador entre dados: Condição para testes públicos:

Elemento	ID	Contador	Tipo	Min	Max	Valor	Spec	Visível	Condição	Feedback	Texto
LINE								true			
DATA	nLinhas		integer	1	10			true			
LINE								true			
DATA	nColu...		integer	1	10			true			
REPEAT		\$nLinhas									
LINE								true			
REPEAT		\$nColunas									
DATA	num		integer	1	128			true			

Novo elemento:
 Elemento: ☒ Visível

ID: Cont: Enumeração:

Valor: Tipo: Condição:

Min: Spec: Feedback:

Max:

ADICIONAR GRAVAR REMOVER DESSELECIONAR

GRAVAR CANCELAR

Figure C.11: Definition of a test case (input part).

The *Record* button saves the definition. Below is an excerpt that shows how the specification is serialized in the file system:

You should now repeat this process for the output by pressing the *Output* button. In this case the definition of the output (Figure C.12) should only represent the matrix elements

Listing C.1: Specification format.

```

1 <LINE visivel='true' >
2   <DATA id='nLinhas' min='1' max='10'
3     type='integer' visible='true' />
4 </LINE>
5 <LINE visivel='true' >
6   <DATA id='nColunas' min='1' max='10'
7     type='integer' visible='true' />
8 </LINE>
9 <REPEAT count='\$nLinhas' >
10  <LINE visivel='true' >
11    <REPEAT count='\$nColunas' >
12      <DATA id='num' max='1' min='128'
13        type='integer' visible='true' />
14    </REPEAT>
15  </LINE>
16 </REPEAT>

```

and not capture the number of rows/columns.

Elemento	ID	Contador	Tipo	Min	Max	Valor	Spec	Visível	Condição	Feedback	Texto
REPEAT		\$nColunas									
LINE								true			
REPEAT		\$nLinhas									
DATA	num		integer	1	128			true			

Figure C.12: Definition of a test case (input part).

After defining the structure and content of files input/output the teacher can press the *Generate* button to automatically generate a set of test cases complying with the specification defined above. The table with the test cases (Figure C.13) comprises several columns:

- # - Number of the test case;
- Input - the first line of the input file;
- Output - the first line of the output file;
- Feedback - feedback message associated with the test case (defined by the element When);
- Public - visibility of the test. The default is not public. A test case can be used as public and included in the feedback in the evaluation report delivered to the student;

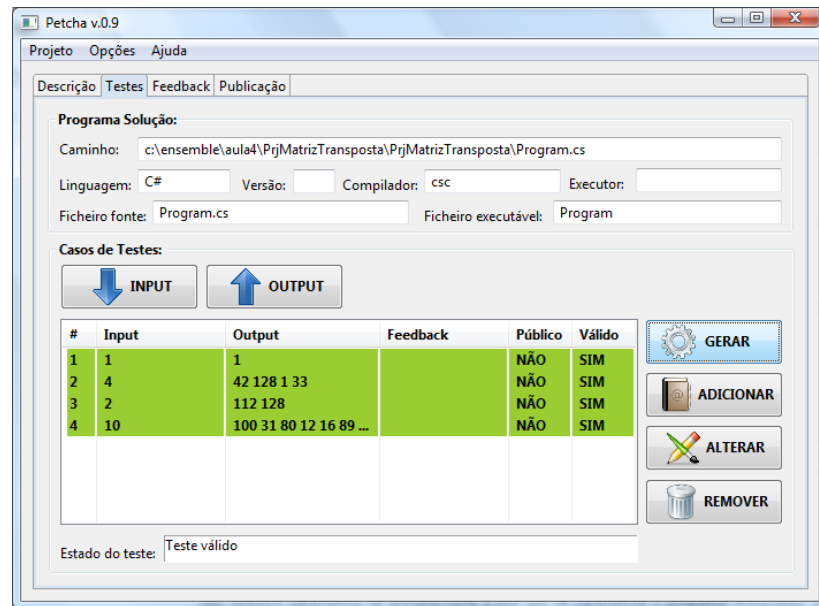


Figure C.13: Automatic generation of test cases.

- Valid - validity of the test. A test case generated from the specification is valid according to this specification. This column is extremely useful in situations in which test cases are created manually and it is necessary to verify whether complies with the specification.

It is possible to re-generate new test cases pressing the *Generate* button or add them manually by pressing the *Add* button. At any time the teacher can also change the existing test cases pressing the *Change* button or remove them by pressing the *Remove* button. For instance, selecting the test case 2 and then clicking *Change* the following window appears (Figure C.14).

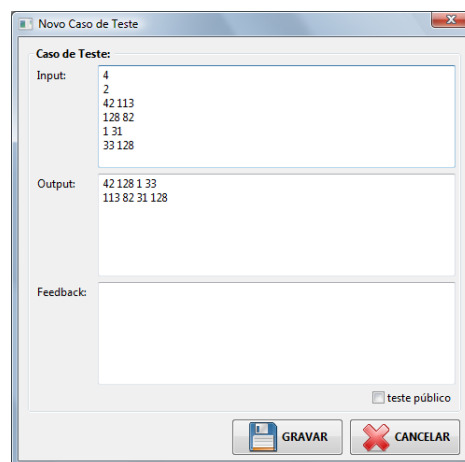


Figure C.14: Test case update.

The teacher can now change the test or simply add feedback or make the public test. There are two major advantages of defining a specification for testing: 1) automatically generate test cases from the specification and 2) validate test cases. Suppose that the teacher manually creates a test case and instead of putting an integer put a letter. In this case, Petcha notifies the teacher with an error message as shown in Figure C.15.

#	Input	Output	Feedback	Público	Válido
1	1	1		NÃO	SIM
2	4	42 128 1 33		NÃO	SIM
3	2	112 128		NÃO	SIM
4	10	100 31 80 12 16 89 ...		NÃO	SIM
5	2	1 X		NÃO	NÃO
Estado do teste: Formatação dos dados errada. Esperado um inteiro.					

Figure C.15: Invalid test case.

Figure C.16 shows the first test case with feedback and public.

#	Input	Output	Feedback	Público	Válido
1	1	1	A transposta de uma matriz 1x1 é igual à ...	SIM	SIM
2	4	42 12...		NÃO	SIM
3	2	112 1...		NÃO	SIM
4	10	100 3...		NÃO	SIM

Figure C.16: Public test case with feedback.

C.1.6 Defining the feedback types

The feedback is an essential part of the automatic evaluation of exercises. Based on the feedback the student is aware if his resolution attempt was successful or not. In Petcha the teacher can set the level of feedback to show to the student. There are five levels of feedback:

1. Worst classification of tests - it presents the student how many tests failed (example: two tests with Wrong Answer);
2. All tests marks - it presents the student the status of all tests that were applied to the solution of the student (example: a test accepted and three tests with Wrong Answer);

3. Tip of the Teacher - it presents a teacher's clue to the student related to a test that produced a wrong answer (example: The factorial of 0 is 1. The empty product, ie the product of any number is always 1);
4. Wrong tests (Data Input) - it presents to the students the input data of a test which produced an incorrect answer (example: input produces 78,117 Wrong Answer);
5. Wrong tests (Data Input / Output) - it presents to the student the input and output data of a test which produced an incorrect answer (example: the input should return 63 110 173).

The teacher can choose (Figure C.17) which levels to show to the students and in which order.

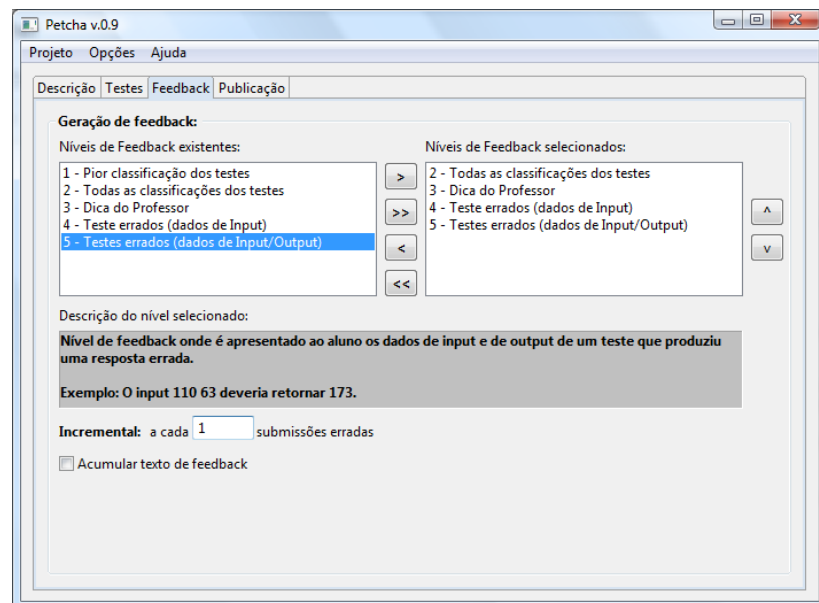


Figure C.17: Feedback levels definition.

To accomplish this task one must select them in the *Existing Feedback Levels* of list box. The > button copies the feedback levels to the listbox *Feedback levels selected*. The ^ and v buttons manage the order of the levels. The >> button allows a complete copy of the levels of feedback and the << button made the opposite.

The teacher can also set whether the feedback is incrementally adding an offset value in the *Incremental* text box. The example shows four levels of feedback with an increase of 1. This means that if the student fails the first time a message will appear associated with the feedback level "All classifications of tests." If the student miss again a message will appear associated with the feedback level "Tip of the Professor" (this feedback level is associated with a given test case and will only appear to the student if the test case in question has

produced a wrong answer) and so until reach the maximum number of levels. After that the last message persists as the feedback for the student.

The teacher can also set what messages are stored in history and are shown in all the evaluation reports. This option is set by activating the *Accumulate Text Feedback* check box.

C.1.7 Packaging and deploying the exercise

As the exercise will be used in a teaching environment and manipulated by several systems including the LMS makes sense to use a standard description of learning objects. Currently, Petcha supports the IMS CC specification which can be defined as a standard packaging of learning objects. A IMS CC package is a ZIP file comprising a manifest file with metadata about the exercise and references to all resources used by exercise (statement, test cases, feedback, program solution and so on).

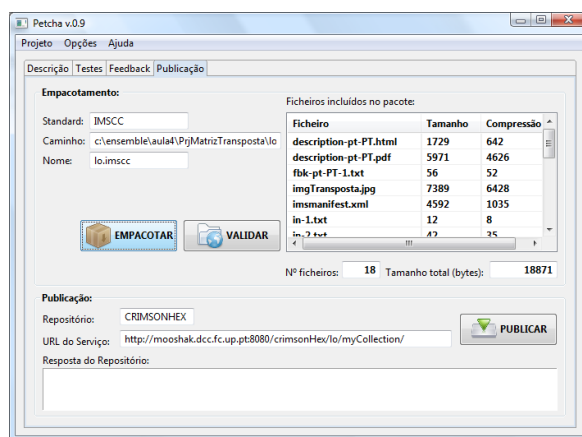


Figure C.18: Exercise packaging.

The *Packaging* group (Figure C.18) includes three fields: the export standard to use, the path for the package and its name. After that by pressing the *Packaging* button the table on the right is populated with all the generated files. It is possible to get information about the number of files in the package and the overall size (in bytes). The *Validate* button starts the validation process that verifies if the package is conform with the IMS CC specification. This action invokes an on-line validation service sponsored by IMS.

After packaging is time to publish the package in a public repository. By default, the repository crimsonHex appears. All the options that appear by default can be configured through the main menu option *Options* → *Settings*.

The exercise is published in the crimsonHex repository. The collection will be *myCollection/aula4*. To do this one must define the group Published URL field of service for

<http://mooshak.dcc.fc.up.pt:8080/crimsonHex/lomyCollection/aula4>.

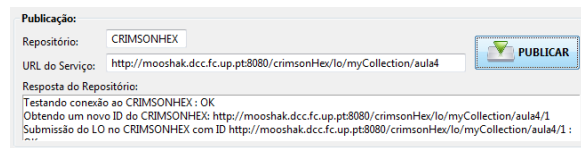


Figure C.19: Exercise deploy in crimsonHex repository.

The *Publish* button (Figure C.19) allows the submission of the package in the configured repository. Figure C.20 depicts an image of the crimsonHex repository with the new LO.

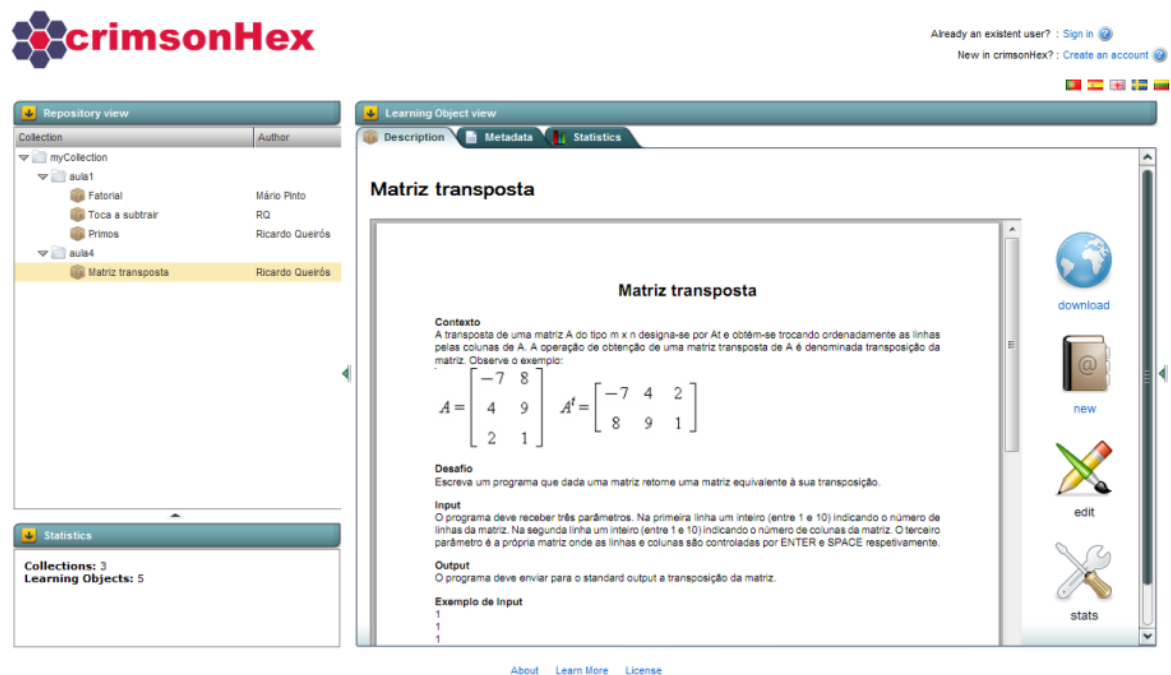


Figure C.20: crimsonHex repository.

C.2 Student manual (distributed to the students - PT)

Este tutorial explica passo a passo como é que o Petcha pode ser usado para auxiliar um Aluno na resolução de exercícios de programação.

C.2.1 Execução do Petcha

O Aluno para aceder aos exercícios deve abrir um LMS (ex: Moodle) e executar um link fornecido pelo Professor. O Petcha (atualmente na sua versão 0.9) usa o protocolo JNLP (Java Network Launching Protocol) que permite a execução de aplicações JAVA no lado do

cliente. Dependendo do browser usado deve executar o ficheiro JNLP após descarregamento. A Figura C.21 mostra o ecrã de arranque do Petcha (modo Aluno).

Petcha - Gestor de Exercícios

Atividade:

Curso: AEISP

Título: Aula nº1

Descrição: primeira aula de C#

Exercícios:

Título	Projeto	Data	Resolvido
Fatorial		28/10/2011	NÃO
Peso Ideal		28/10/2011	NÃO
Massa Corporal		28/10/2011	NÃO
Ano Bissexto		28/10/2011	NÃO
Classificar triangulos		28/10/2011	NÃO

Sequenciação: NÃO

Projeto:

IDE: VISUAL STUDIO EXPRESS Linguagem: C#

Localização: C:\Users\Ricardo

Projeto: Projeto1

CRIAR SAIR

Figure C.21: Ecrã inicial do Petcha (modo aluno).

O link fornecido no LMS pelo Professor aponta para uma atividade. Uma atividade (ou aula) é composta por um conjunto de exercícios que o Professor espera que o Aluno resolva. Para além de uma lista de exercícios o ecrã inicial inclui a seguinte informação:

- Nome do Curso/Disciplina: nome do Curso/Disciplina onde foi colocado o link pelo Professor;
- Título da Atividade: nome da atividade (conjunto de exercícios);
- Descrição da Atividade: informação adicional sobre a atividade. Pode conter quais as temáticas incluídas nos exercícios e objetivos que o Professor espera que os Alunos atinjam ao resolver os exercícios;
- Nome do Aluno: nome do aluno que clicou no link.

C.2.2 Criação de um projeto

Para começar a resolver os exercícios o Aluno deverá seleccionar um exercício (Figura C.22).

The screenshot shows a software window titled 'Exercícios:'. It contains a table with four columns: 'Título', 'Projeto', 'Data', and 'Resolvido'. The first row is highlighted in blue and contains the text 'Fatorial', an empty 'Projeto' field, '28/10/2011', and 'NÃO'. Below the table, it says 'Sequenciação: NÃO'. Underneath is a section titled 'Projeto:' with four input fields: 'IDE:' (a dropdown menu showing 'VISUAL STUDIO EXPRESS'), 'Linguagem:' (a dropdown menu showing 'C#'), 'Localização:' (a text box showing 'C:\Users\Ricardo'), and 'Projeto:' (a text box showing 'Projeto1'). At the bottom right are two buttons: 'CRIAR' with a folder icon and 'SAIR' with a power button icon.

Título	Projeto	Data	Resolvido
Fatorial		28/10/2011	NÃO
Peso Ideal		28/10/2011	NÃO
Massa Corporal		28/10/2011	NÃO
Ano Bissexto		28/10/2011	NÃO
Classificar triangulos		28/10/2011	NÃO

Sequenciação: NÃO

Projeto:

IDE: Linguagem:

Localização:

Projeto:

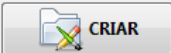

 

Figure C.22: Criação de projeto.

O grupo Exercícios contém uma tabela com os exercícios propostos pelo Professor. Um exercício é incluí a seguinte informação:

- Título: título do exercício (dado pelo Professor na altura da criação do exercício);
- Projeto: nome do projeto associado ao exercício. Se o Aluno ainda não tiver criado um projeto para o exercício este campo aparece em branco;
- Data: data de criação do exercício pelo Professor;
- Resolvido: estado do exercício (SIM - já foi resolvido pelo Aluno e NÃO - caso contrário).

O Grupo inclui também uma referência à sequenciação de exercícios. Esta sequenciação é definida pelo Professor na altura da definição da atividade e indica se os exercícios devem ser resolvidos sequencialmente (ordem evidenciada na tabela). Após a selecção de um exercício o Grupo Projecto é activado com as informações necessárias à criação de um Projecto:

- IDE: nome do ambiente de desenvolvimento desejado para resolver o exercício;
- Linguagem: nome da linguagem de programação desejada para resolver o exercício;
- Localização: localização onde o projecto vai ser criado (por omissão o local é o home do utilizador que fez login na máquina);
- Projeto: nome do projeto associado ao exercício.

Renomeie o projecto para PrjFatorial e pressione o botão *Criar*. Se o projeto já existir o mesmo botão surge com o texto *Abrir*.

C.2.3 Resolvendo um exercício

Após a ação anterior é aberta a janela de Gestão de Resolução de Exercícios (Figura C.23) que vai acompanhar o Aluno durante a resolução dos exercícios.

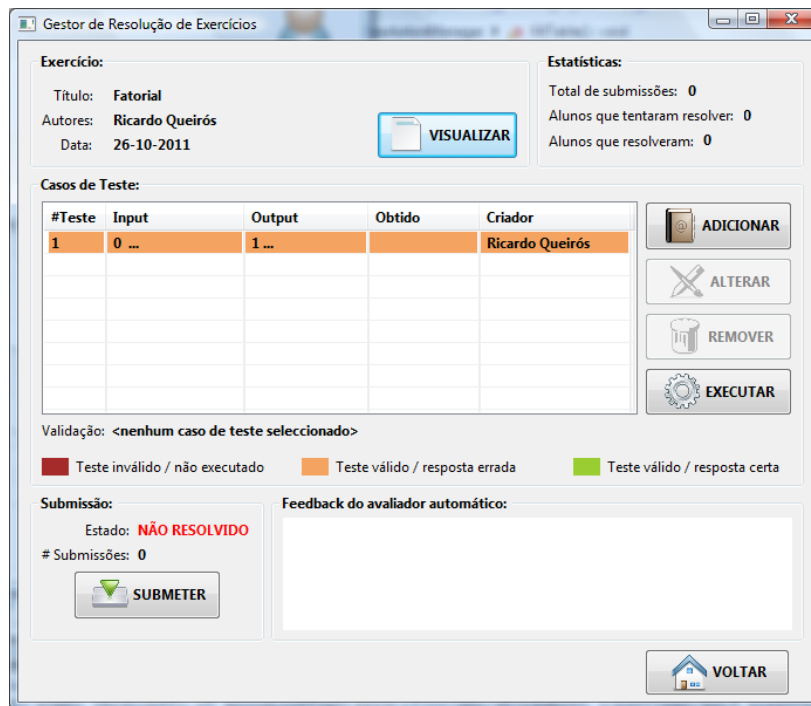


Figure C.23: Gestão da resolução de exercícios.

Para começar o Aluno deverá ler o enunciado pressionando o botão *Visualizar*.

Fatorial

Contexto

Na matemática, o fatorial de um número natural n , representado por $n!$, é o produto de todos os inteiros positivos menores ou iguais a n .

Desafio

Calcular o fatorial de um número.

Input

O programa deve receber um valor inteiro entre 0 e 10.

Output

O programa deve retornar o fatorial do número como um inteiro.

Exemplo de Input

0

Exemplo de Output

1

Figure C.24: Enunciado do exercício.

Após a leitura do enunciado (Figura C.24) o Aluno deve aceder ao projeto no sistema de ficheiros. Após duplo-clique sobre o ficheiro PrjFatorial o Aluno pode começar a resolver o exercício (Figura C.25).

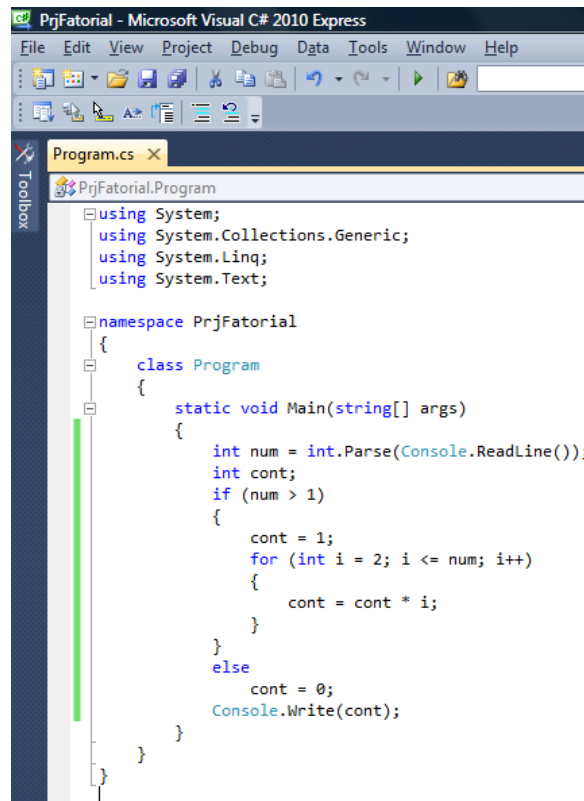


Figure C.25: Codificação da solução por parte do Aluno.

C.2.4 Testando

Após a codificação da solução o Aluno poderá usar o Petcha para testar a sua solução. O grupo Casos de Teste permite ao Aluno executar dois tipos de testes: testes públicos criados pelo Professor (aparecem automaticamente na tabela) e testes adicionados pelo próprio Aluno (clicando no botão *Adicionar*).

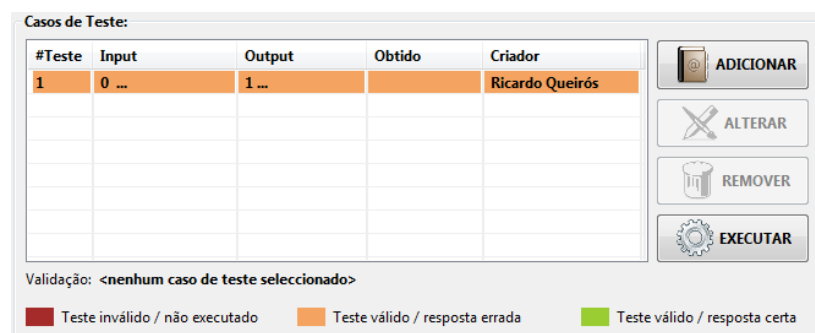


Figure C.26: Gestão de testes.

A tabela da Figura C.26 inclui os seguintes campos:

- # Teste - número do teste;
- Input - dados de input (testes do Professor apenas se pode ver parte dos dados);
- Output - dados de output (testes do Professor apenas se pode ver parte dos dados);
- Obtido - output obtido pelo Aluno usando a sua solução e os dados de input do teste;
- Criador - nome de quem criou o teste.

Para definir um novo teste o Aluno deve clicar no botão *Adicionar*.

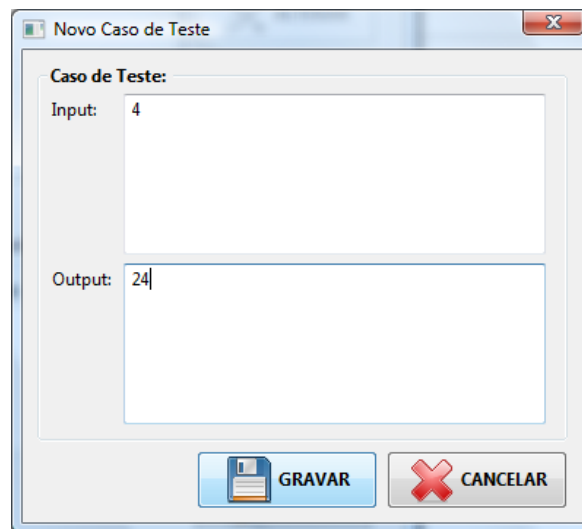


Figure C.27: Novo teste.

A definição de um novo teste (Figura C.27) passa por definir os dados de input e output do teste. Adicione os seguintes testes:

1. Input = 4 e Output= 24
2. Input = 5 e Output= 120
3. Input = x e Output= 2

Após definição de todos os testes o Aluno deve pressionar o botão *Executar* para executar os testes localmente.

Os testes podem ter 3 estados (realçados com cores distintas):

- Teste inválido/não executado - os dados de input/output não obedecem à especificação de testes criados pelo Professor. Neste caso o Aluno deverá verificar o texto no campo Validação para mais informação sobre como corrigir o problema. Estes testes não são executados;

Casos de Teste:

#Teste	Input	Output	Obtido	Criador
1	0 ...	1 ...	0	Ricardo Queirós
2	4	24	24	
3	5	120	120	
4	x	2		

Validação: <nenhum caso de teste seleccionado>

■ Teste inválido / não executado
 ■ Teste válido / resposta errada
 ■ Teste válido / resposta certa

ADICIONAR
 ALTERAR
 REMOVER
 EXECUTAR

Figure C.28: Execução local de testes.

- Teste válido/resposta errada - os dados de input/output são válidos, mas o output esperado é diferente do output obtido. Isso significa que o Aluno deve recodificar a sua solução de forma a resolver este problema;
- Teste válido/resposta certa - os dados de input/output são válidos e o output esperado é igual ao output obtido. Isso significa que a solução do Aluno passou neste teste com sucesso.

A Figura C.28 mostra que o primeiro teste é válido mas é esperado o output de 1 e o output da execução da solução do aluno dá 0 o que evidencia que o código do Aluno tem um erro de lógica. Os dois testes seguintes que foram criados pelo Aluno são válidos e os outputs coincidem. No último teste criado pelo Aluno a sua validação falhou pelo que o Aluno deverá ler o campo validação e verificar a origem do problema (ex: esperado um inteiro).


C.2.5 Submetendo uma solução

A qualquer altura o Aluno poderá submeter a sua resolução ao Avaliador e verificar se o feedback enviado pelo Avaliador é esclarecedor. Para enviar a sua tentativa de resolução do exercício o aluno deve pressionar o botão *Submeter*.

Submissão:

Estado: **NÃO RESOLVIDO**

Submissões: 1

 SUBMETER

Feedback do avaliador automático:

Dica:

O fatorial de 0 é 1 porque o produto vazio, isto é,

Figure C.29: Feedback automático providenciado pelo Avaliador.

O Grupo Submissão indica o estado da solução do Aluno após submissão (RESOLVIDO ou NÃO RESOLVIDO) e o número de submissões feitas. O Grupo Feedback apresenta a mensagem de feedback providenciada pela Avaliador. No exemplo da Figura C.29, a primeira mensagem de feedback apresentada é a dica fornecida pelo Professor na altura de criação dos

testes. O tipo de feedback está associado à sequenciação definida pelo Professor na altura da definição do feedback incremental. Após nova submissão é apresentada a mensagem de feedback associada ao nível de feedback atual (Figura C.30).

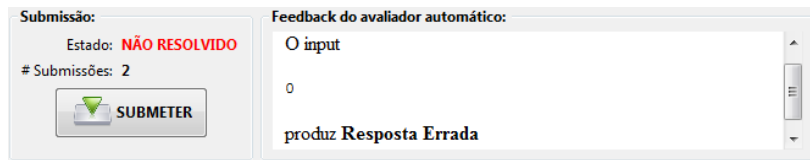


Figure C.30: Feedback automático providenciado pelo Avaliador.

E assim sucessivamente até que o Aluno consiga resolver com sucesso o exercício (Figuras C.31 e C.32).

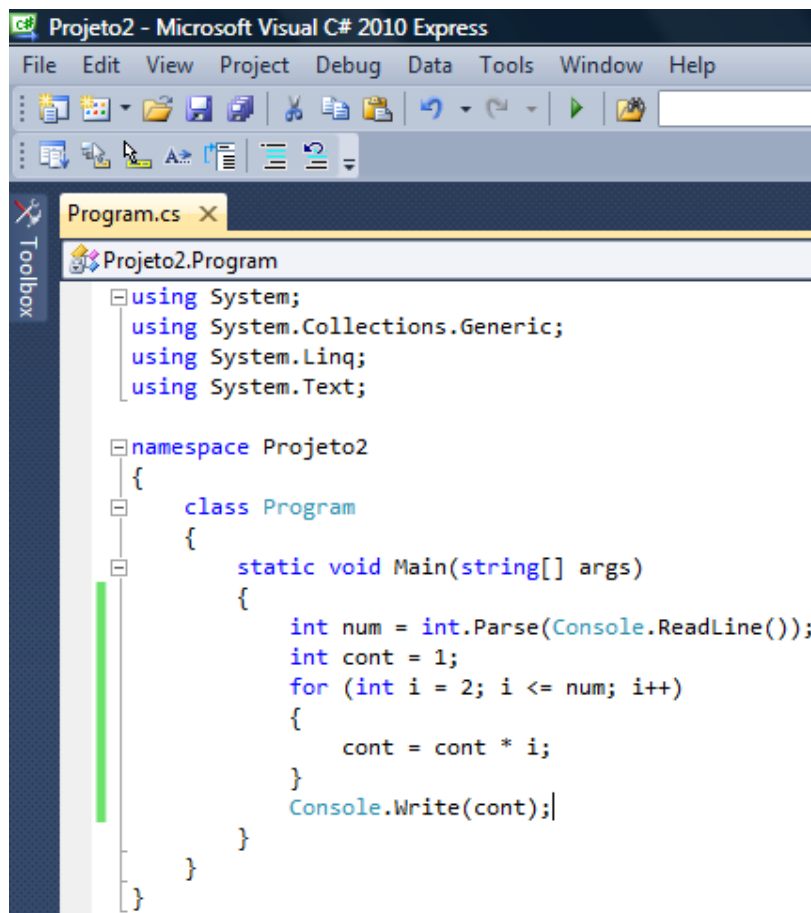


Figure C.31: Solução correta.

ATENÇÃO: Mesmo tendo todos os testes correctos isso não significa que a resposta esteja correta. Os testes usados no Petcha são apenas aqueles que foram definidos pelo Professor como sendo públicos mais os testes acrescentados pelo Aluno. No entanto, existem outros testes que foram definidos na altura da criação do exercício que não foram usados. Para usar

Submissão:
Estado: **RESOLVIDO**
Submissões: 3
SUBMETTER

Feedback do avaliador automático:
RESOLVIDO

Figure C.32: Feedback automático providenciado pelo Avaliador.

esses testes o Aluno terá que submeter a sua resolução para um Avaliador público. Esse Avaliador usa todos os testes e, após avaliação, envia um relatório de avaliação para o Aluno contendo feedback. A qualquer altura o Aluno pode ver as estatísticas associadas ao exercício no Grupo Estatísticas (Figura C.33).

Estatísticas:
Total de submissões: 0
Alunos que tentaram resolver: 0
Alunos que resolveram: 0

Figure C.33: Estatísticas do exercício.

Os dados estatísticos são: total de submissões - nº de submissões total feitas pelos alunos; Alunos que tentaram resolver - nº de alunos que tentaram resolver o exercício; Alunos que resolveram - nº de alunos que efetivamente resolveram o exercício. Após resolução o Aluno deve pressionar o botão *Voltar* e resolver os restantes exercícios (Figura C.34).

Petcha - Gestor de Exercícios

Atividade:
Curso: AEISP
Título: Aula nº1
Descrição: primeira aula de C#

Exercícios:

Título	Projeto	Data	Resolvido
Peso Ideal	C:\Users\Ricardo\Projeto2	30/10/2011	NÃO
Fatorial		30/10/2011	SIM
Massa Corporal		30/10/2011	NÃO
Ano Bissexto		30/10/2011	NÃO
Classificar triangulos		30/10/2011	NÃO

Sequenciação: NÃO

Projeto:
IDÉ: VISUAL STUDIO EXPRESS Linguagem: C#
Localização: C:\Users\Ricardo\
Projeto: Projeto2

ABRIR SAIR

Figure C.34: Resolução dos restantes exercícios.

Appendix D

crimsonHex Core Functions

This section describes the request and response messages of each crimsonHex core function (Table D.1) adopting the REST interface syntax.

Table D.1: Core functions of the repository.

Function	SOAP	REST
Register	URL getNextId()	GET /?nextId > URL
Submit	submit(URL loid, LO lo)	PUT URL < LO
Request	LO request(URL loid)	GET URL > LO
RequestAsset	LO requestAsset(URL loid, String asset)	GET URL/asset > ASSET
Search	XML search(XQuery query)	POST / < XQUERY > XML
Report	Report(URL loid, LOReport rep)	PUT URL < LOREPORT
Alert	RSS getUpdates([Integer minutes])	GET /?alert+minutes > RSS
Create	XML Create(URL collection)	PUT URL
Remove	XML Remove(URL collection)	DELETE URL
Status	XML getStatus()	GET /?status > XML

D.1 Register function

The Register function (Listing D.1) enables client systems to reserve a unique ID.

Listing D.1: The Reserve function.

```
1 GET http://repository/lo?nextId
2 // The response is included in the Location HTTP header.
3 Location: http://repository/lo/123
```

D.2 Submit function

The Submit function uploads an LO to a repository and makes it available for future access. Listing D.2 includes an example.

Listing D.2: The Submit function.

```

1 // The HTTP post of a LO compliant with the IMS CC specification.
2 POST http://repository/lo/123
3 ...
4 [BINARY DATA]
```

D.3 Request and RequestAsset functions

The Request and RequestAsset functions get a LO or part of it from the repository. A request/response example of the latter is presented in Listing D.3.

Listing D.3: The RequestAsset function.

```

1 // The HTTP request for an existing asset.
2 GET http://repository/123/SOLUTION
3
4 // The server response returns the asset (if multiple files
5 the server must package them before sent to the client).
6 HTTP/1.x 200 OK
7 Content-Type: application/zip
8 Transfer-Encoding: chunked
```

D.4 Search function

The Search function enables client systems to query the repository based on the XQuery specification.

Listing D.4: The Search function.

```

1 declare namespace imsm="http://www.msglobal.org/xsd/imsm_v1p2";
2   for $p in //imsm:lom
3     where contains
4       ($p/imsm:educational/imsm:difficulty/imsm:value/imsm:langstring,"easy")
5       return \ $p/imsm:general/imsm:title/imsm:langstring/text()
```

Listing D.4 shows an excerpt of a XQuery file for the selection of all programming exercise titles with an easy difficulty level. Listing D.5 shows the respective POST request.

Listing D.5: The Search function - POST request.

```

1 // The HTTP POST of a XQuery file.
2 POST http://repository/lo
3 ...
4 [XQUERY FILE]

```

Alternatively, it is possible to use a GET request with the searched fields and respective values as part of the URL query string. Listing D.6 shows an example of a simple search for the selection of all exercises whose author is Manzoor.

Listing D.6: The Search function - GET request.

```

1 // The HTTP GET request for the Search function.
2 http://repository/lo?author=Manzoor

```

In both approaches the result is a valid XML document as shown in Listing D.7.

Listing D.7: The Search function - HTTP response.

```

1 // The HTTP response for the Search function.
2 <result base-url="http://repository/lo">
3   <request source="http://repository/lo"
4     message="Querying repository" />
5   <response message="3 LOs found...">
6     <resources>
7       <resource idCol="" idLo="5">
8         Hashmat the Brave Warrior
9       </resource>
10      <resource idCol="" idLo="123">
11        Summation of Four Primes
12      </resource>
13      <resource idCol="graphs/" idLo="2">
14        InCircle
15      </resource>
16    </resources>
17  </response>
18 </result>

```

D.5 Report function

The Report function associates a usage report to an existing LO. Listing D.8 shows a report of the use of a specific LO. Listing D.9 shows the respective POST request.

Listing D.8: The Report function - XML report.

```

1 // Report file
2 <?xml version="1.0" ?>
3 <report loid="http://repository/lo/123" >
4   <item item-type="attempt"
5     item-name = "data" item-value = "11-01-2008 15:21:23" />
6   <item item-type="attempt"
7     item-name = "time" item-value = "3332" />
8   <item item-type="attempt"
9     item-name = "attempts" item-value = "2" />
10  <item item-type="attempt"
11    item-name = "success" item-value = "false" />
12  <item item-type="learner"
13    item-name = "gender" item-value = "female" />
14  <item item-type="learner"
15    item-name = "age" item-value = "14" />
16  <item item-type="learner"
17    item-name = "country" item-value = "pt-PT" />
18 </report>

```

Listing D.9: The Report function - POST request.

```

1 // The HTTP POST of a report file.
2 POST http://repository/lo
3 ...
4 [REPORT FILE]

```

D.6 Alert function

The Alert function notifies users of changes in the state of the repository using an RSS feed. A request/response example of this function is presented in Listing D.10.

Listing D.10: The Alert function.

```

1 GET http://repository/lo?alert+minutes
2 // The response message is a RSS compliant file with the repository updates.

```

D.7 Create function

The Create function adds new collections to the repository. Listing D.11 includes both request and response messages.

Listing D.11: The Create function.

```
1 // The HTTP request for the creation of a collection.
2 PUT http://repository/newCollection
3
4 // The response message.
5 <result base-url="http://repository/" ...>
6   <request
7     source="http://repository/newCollection"
8     message="Creating new collection" />
9   <response message="Collection created">
10     <resource idCol="newCollection" idLo="" />
11   </response>
12 </result>
```

D.8 Remove function

The Remove function removes an existent collection or learning object. Listing D.12 shows an example.

Listing D.12: The Remove function.

```
1 // The HTTP request for the remotion of a LO.
2 DELETE http://repository/123
3
4 // The response message.
5 <result base-url="http://repository/" ...>
6   <request
7     source="http://repository/123"
8     message="Deleting a LO" />
9   <response message="LO deleted">
10     <resource idCol="" idLo="123" />
11   </response>
12 </result>
```

D.9 Status function

The Status function returns a general status of the repository, including versions of the components, their capabilities and statistics.

Index

- .LRN, 20
- abstract frameworks, 49
- AJAX, 141
- AMS, 22, 24
- Apache FOP, 166
- API integration, 78
- APML, 64
- ARIADNE, 29
- AS, 31
- Atom, 84
- AutoGrader, 37
- axial systems, 90, 91
- BabeLO, 130, 147
- Bepress, 30
- BerkleeShares, 30
- Blackboard, 20, 21
- BOSS2, 37
- CATS, 73
- CMS, 17
- Component based systems, 12
- concrete frameworks, 52
- Connexions, 30
- CONTENTdm, 28
- core services, 92
- CourseMaker, 37
- CQL, 84
- crimsonHex, 129, 133
- CTPracticals, 37
- data integration, 78
- data pull, 84
- data push, 83
- DCMI, 59
- Desire2Learn, 20
- digital libraries, 27
- DigiTool, 28
- Dokeos, 20
- DOMJudge, 33
- DSpace, 28
- Dublin Core, 58
- E-Framework, 54, 96
- e-learning frameworks, 48
- e-learning services, 13
- e-portfolio, 78
- Eclipse, 130
- EduComponents, 37
- EPrints, 28
- Equella, 28
- eXist, 136
- experiment, 175
- Fedora, 28
- Flori, 29
- FPS, 73, 152
- GAME, 37
- GEM, 30
- Greenstone, 28
- GWT, 141
- HarvestRoad Hive, 29
- Hot Potatoes, 23
- HR-XML, 64
- HUSTOJ, 37
- IEEE LOM, 58, 170

- IEEE LTSA, 50
- ILOX, 70, 84
- IMS Abstract Framework, 51
- IMS CC, 67, 93, 116, 118, 168
- IMS CP, 65
- IMS DRI, 84, 94, 98, 116
- IMS LD, 66
- IMS LIS, 100, 122
- IMS LODE, 70, 84
- IMS LTI, 95, 99, 120, 162
- IMS QTI, 3, 69, 71, 96, 104
- IMS SS, 66
- IntraLibrary, 29
- ISO MLR, 63

- JAX-RS, 151
- JAXB, 164
- Jersey, 151
- JNLP, 162
- JUnit, 142

- LAO, 68, 94, 168
- LCMS, 17
- LeMill, 30
- LMS, 17, 77
- LO, 57, 93, 104, 133
- LO.NET, 30
- LOR, 27, 29
- LRE (European Schoolnet), 30

- Mahara, 78
- Maricopa, 30
- MathJax, 202
- Merlot, 24, 30
- METS, 64
- MIX, 64
- MLE, 17
- Mobile learning, 19
- MODS, 64
- Moe, 37
- Moodle, 20, 21, 72, 78, 129, 143

- Mooshak, 32, 37, 74, 129, 152

- Nielsen's model, 175, 180

- OAI-PMH, 84
- OAuth, 100
- OKI, 51, 83
- OUSS, 53

- Peach, 74
- PENS, 83
- Petcha, 130, 157, 175
- PExIL, 135, 149
- PLQL, 85
- PREMIS, 64

- quasi-experiment, 177

- referatories, 26
- Reload, 23
- REST, 95, 117, 151
- RPC, 95, 117
- RSS, 119

- SaaS, 19, 80
- Sakai, 20
- Saxon, 166
- Schematron, 136, 138
- SCORM, 67, 70
- Scriptorium, 30
- secondary services, 92
- SIF, 53
- SOA, 14, 19
- SOAP, 95
- SPI, 83
- SPOJ, 33
- SQI, 84
- SRU/SRW, 84
- Submit, 37
- survey, 179
- SWORD, 83
- SWT, 163

TA, 91, 157
Talent Management, 19
tool integration, 79

UVA Online Judge, 33

Verkkoke, 37
Visual Studio, 129
VLE, 17

Web 2.0, 19
Web-CAT, 37
Wisc-Online, 30

XAMPP, 178
XML Schema, 125, 136
XPath, 137, 167
XQuery, 118, 136
XSL-FO, 166
XSLT, 136, 137, 166

Z39.50, 84
Zentity, 28

References

- [ADjH⁺06] Lora Aroyo, Peter Dolog, Geert jan Houben, Milos Kravcik, Ambjörn Naeve, Mikael Nilsson, and Fridolin Wild. Interoperability in personalized adaptive learning. *Journal of Educational Technology & Society*, 9(2):4–18, 2006. http://www.ifets.info/journals/9_2/2.pdf.
- [AK03] Theodore K. Apostolopoulos and Anna S. Kefala. An e-learning service management architecture. In *ICALT*, pages 140–144, 2003.
- [AKD06] Hend S. Al-Khalifa and Hugh C. Davis. The evolution of metadata from standards to semantics in e-learning applications. In *Hypertext*, pages 69–72, 2006. <http://doi.acm.org/10.1145/1149941.1149956>.
- [AM05] K. Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Journal of Computer Science Education*, 15(2):83–102, 2005. <http://www.tandfonline.com/doi/pdf/10.1080/08993400500150747>.
- [AS10] C. Al-Smadi, M.; Gutl. Soa-based architecture for a generic and flexible e-assessment system. In *EDUCON*, 2010.
- [ASQ⁺06] Sandra Aguirre, Joaquín Salvachúa, Juan Quemada, Antonio Fumero, and Antonio Tapiador. Joint degrees in e-learning systems: A web services approach. In *Proceedings of the 2nd IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006)*, November 2006.
- [BBF⁺11] Steve Benford, Edmund Burke, Eric Foxley, Neil Gutteridge, and Abdullah Zin. Early experiences of computer-aided assessment and administration when teaching computer programming. *Research in Learning Technology*, 1(2), 2011.
- [BC10] Phil Barker and Lorna M. Campbell. Metadata for learning materials: an overview of existing standards and current developments.

- Technology, Instruction, Cognition and Learning*, 7(3-4):225–243, 2010. http://www.icbl.hw.ac.uk/publicationFiles/2010/TICLMetadata/TICLpaper.MetadataForEducation_postref.pdf.
- [Ber09] Howard C. O’Leonard K. & Mallon D. Bersin, J. Learning management systems 2009: Executive summary. Bersin & Associates, 2009.
- [BG07] Walter R. Borg and Meredith Damien Gall. *Educational research; an introduction, by Walter R. Borg and Meredith D. Gall*. McKay New York,, 8th ed. edition, 2007.
- [BGNM04] Michael Blumenstein, Steven Green, Ann Nguyen, and Vallipuram Muthukumarasamy. An experimental analysis of game: a generic automated marking environment. *SIGCSE Bull.*, 36:67–71, June 2004.
- [Bri98] Liber O. Britain, S. A Framework for Pedagogical Evaluation of Virtual Learning Environments. Technical report, 1998. <http://www.leeds.ac.uk/educol/documents/00001237.htm>.
- [Bur10] Juan C. Burguillo. Using game theory and competition-based learning to stimulate student motivation and performance. *Comput. Educ.*, 55(2):566–575, September 2010.
- [CCF⁺07] Giovanni Casella, Gennaro Costagliola, Filomena Ferrucci, Giuseppe Polese, and Giuseppe Scanniello. A scorm thin client architecture for e-learning systems based on web services. *IJDET*, 5(1):19–36, 2007.
- [CCMN04] Girish B. Chaffle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Decentralized orchestration of composite web services. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. ’04, pages 134–143, New York, NY, USA, 2004. ACM.
- [CDK⁺02] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *Internet Computing, IEEE*, 6(2):86–93, March 2002. <http://dx.doi.org/10.1109/4236.991449>.
- [CF07] Jason Cole and Helen Foster. *Using Moodle: Teaching with the Popular Open Source Course Management System*. O’Reilly Media, 2 edition, November 2007.
- [CKLO03] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. On automated grading of programming assignments in an academic institution. *Comput. Educ.*, 41:121–131, September 2003.

- [CZS10] George Chloros, Panayiotis Zervas, and Demetrios G. Sampson. Ask-lom-ap: A web-based tool for development and management of iee lom application profiles. In *ICALT*, pages 138–142, 2010. <http://dx.doi.org/10.1109/ICALT.2010.46>.
- [Dal99] Charlie Daly. Roboprof and an introductory computer programming course. *SIGCSE Bull.*, 31(3):155–158, June 1999.
- [Dig07] Digital Library Federation. Metadata encoding and transmission standard: Primer and reference manual. <http://www.loc.gov/standards/mets/mets-schemadocs.html>, September 2007.
- [DLO05] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, 5, September 2005.
- [DOL⁺07] Declan Dagger, Alexander O’Connor, Seamus Lawless, Eddie Walsh, and Vincent P. Wade. Service-Oriented E-Learning Platforms: From Monolithic Systems to Flexible Services. *Internet Computing, IEEE*, 11(3):28–35, 2007. http://ieeexplore.ieee.org/xpls/abs/_all.jsp?arnumber=4196172.
- [Don02] J. Donello. Theory & practice: Learning content management systems. *ELearningMag*, 2002.
- [DW09] Carmean C. Davis, B. and E.D. Wagner. The Evolution of the LMS: From Management to Learning - Deep Analysis of Trends Shaping the Future of eLearning. Technical report, Sage Road Solutions, LLC, 2009.
- [EBC⁺08] Stephen H. Edwards, Jürgen Börstler, Lillian N. Cassel, Mark S. Hall, and Joseph Hollingsworth. Developing a common format for sharing programming assignments. *SIGCSE Bull.*, 40(4):167–182, 2008.
- [Eck95] Wayne W. Eckerson. Three tier client/server architecture: Achieving scalability, performance, and efficiency in client server applications. *Open Information Systems*, 10(1), 1995.
- [Eck09] Anna Eckerdal. *Novice Programming Students’ Learning of Concepts and Practise*. PhD thesis, Uppsala UniversityUppsala University, Division of Scientific Computing, Numerical Analysis, 2009.
- [EFMM10] Micaela Esteves, Benjamim Fonseca, Leonel Morgado, and Paulo Martins. Improving teaching and learning of computer programming through the use of the Second Life virtual world. *British Journal of Educational Technology*, March 2010.

- [EHR04] Ty Mey Eap, Marek Hatala, and Griff Richards. Digital repository interoperability: design, implementation and deployment of the ecl protocol and connecting middleware. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. '04, pages 376–377, New York, NY, USA, 2004. ACM.
- [ELC07] Steve Engels, Vivek Lakshmanan, and Michelle Craig. Plagiarism detection using feature-based neural networks. *SIGCSE Bull.*, 39:34–38, March 2007.
- [Ell09] Ryann K. Ellis. Field guide to learning management systems. ASTD Learning Circuits, 2009.
- [EP06] Stephen H. Edwards and William Pugh. Toward a common automated grading platform. In *Birds-of-a-Feather session at the 37th SIGCSE Technical Symposium on Computer Science Education*, March 2006.
- [Erl05] T. Erl. *Service-oriented architecture - Concepts, Technology and Design*. Prentice Hall, 2005.
- [Fay10] Ed Fay. Repository software comparison: Building digital library infrastructure at lse. *Ariadne*, 64, 2010. <http://www.ariadne.ac.uk/issue64/fay/>.
- [FCN⁺11] J L Fernandez, J M Carrillo, J Nicolas, A Toval, and M I Carrion. Trends in e-learning standards. *International Journal of Computer Applications*, 353(1):49–54, 2011.
- [Fri04a] Norm Friesen. chapter Semantic and Syntactic Interoperability for Learning Object Metadata. ALA Editions, 2004.
- [Fri04b] Norm Friesen. Editorial - a gentle introduction to technical e-learning standards. *Canadian Journal of Learning and Technology*, 30(3), 2004. <http://www.cjlt.ca/index.php/cjlt/article/view/136>.
- [Fri05] N. Friesen. Interoperability and learning objects: An overview of e-learning standardization. *Interdisciplinary Journal of Knowledge and Learning Objects*, 2005.
- [FT99] Frank Farance and Joshua Tonkel. Ltsa specification - learning technology systems architecture, draft 5. Technical report, IEEE, 1999.
- [FT00] R.T. Fielding and R.N. Taylor. Principled design of the modern web architecture. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 407–416, 2000.

- [GG08] P. Guerreiro and Katerina Georgouli. Enhancing elementary programming courses using e-learning with a competitive attitude. *International Journal of Internet Education*, 10, 01 2008.
- [Gil10] T. Gilbert. Leveraging sakai and ims lti to standardize integrations. In *10th Sakai Conference*, 2010.
- [GM07] Anabela Gomes and António José Mendes. Learning to program - difficulties and solutions. *Proceedings of the International Conference on Engineering Education*, 2007.
- [GP05] Paul Gross and Kris Powers. Evaluating assessments of novice programming environments. In *Proceedings of the first international workshop on Computing education research*, ICER '05, pages 99–110, New York, NY, USA, 2005. ACM.
- [GRACG⁺09] Israel Gutiérrez Rojas, Álvaro Agea, Raquel M Crespo García, Abelardo Pardo, and Carlos Delgado Kloos. *Assessment Interoperability using QTI*. 2009.
- [Har06] Linda Harasim. A History of E-learning: Shift Happened. pages 59–94. 2006.
- [HGST05] Colin A. Higgins, Geoffrey Gray, Pavlos Symeonidis, and Athanasios Tsintzifas. Automated assessment and experiences of teaching programming. *J. Educ. Resour. Comput.*, 5, September 2005.
- [HK06] Keith Harman and Alex Koochang. *Learning Objects: Standards, Metadata, Repositories, and LCMS*. Informing Science Press, Santa Rosa, CA, USA, 2006.
- [HM11] Tore Hoel and Jon Mason. Expanding the scope of metadata and the issue of quality. In *19th International Conference on Computers in Education*, 2011. http://hoel.nu/publications/ICCE_workshop_paper_Hoel_Mason2011\\-final.pdf.
- [IAKS10] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 86–93, New York, NY, USA, 2010. ACM.
- [JBK05] Borka Jerman-Blazic and Tomaz Klobucar. Privacy provision in e-learning standardized systems: status and improvements. *Computer Standards & Interfaces*, 27, 2005. <http://www.qou.edu/arabic/researchProgram/eLearningResearchs/privacyp.pdf>.

- [Jen02] Tony Jenkins. On the Difficulty of Learning to Program. In *3rd annual Conference of LTSN-ICS*, Loughborough, 2002.
- [Jen08] Somyajit Jena. Authoring and sharing of programming exercises. Master’s thesis, San Jose State University, 2008. http://scholarworks.sjsu.edu/etd_projects/19.
- [JU97] D. Jackson and M. Usher. Grading student programming using assyst. In *In Technical Symposium on Computer Science Education, Proceedings of the 28th SIGCSE*, pages 335–339, 1997.
- [Jue03] D.W. Juedes. Experiences in web-based grading. In *In 33rd ASEE/IEEE Frontiers in Education Conference*, pages 5–8, 2003.
- [KC] Jan M. Pawlowski Kati Clements, Àgueda Gras-Velázquez. Educational resources packaging standards scorm and ims common cartridge – the users point of view. In *Search and Exchange of e-learning Materials 2010 Proceedings*.
- [Kle11] Alexander Klenin. Common problem description format: Requirements. ACM-ICPC World Final CLIS (Competitive Learning Institute Symposium), 2011.
- [KSSM10] P. Kumar, S.G. Samaddar, A.B. Samaddar, and A.K. Misra. Extending ieeeltsa e-learning framework in secured soa environment. In *2nd International Conference on Education Technology and Computer (ICETC)*, 2010.
- [Kur12] E. Kurilovas. *European Learning Resource Exchange: A Platform for Collaboration of Researchers, Policy Makers, Practitioners, and Publishers to Share Digital Learning Resources and New e-Learning Practices*. IGI-Global, 2012.
- [LAMJ05] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. *SIGCSE Bull.*, 37(3):14–18, June 2005.
- [Les06] Scott Leslie. Challenges to Implementing DSpace as a LOR, 2006.
- [LJ99] Michael Luck and Mike Joy. A secure on-line submission system. In *SOFTWARE - PRACTICE AND EXPERIENCE*, pages 721–740, 1999.
- [LL10] L. Levensaler and M. Laurano. Talent management systems 2010: Market realities, implementation experiences and solution provider profiles. Bersin & Associates, 2010.
- [LH00] Fong lok Lee and Rex Heyworth. Problem complexity: A measure of problem difficulty in algebra by using computer, 2000.

- [LLXW09] Yingli Liang, Quanbo Liu, Jun Xu, and Dongqing Wang. The recent development of automated programming assessment. In *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, pages 1–5, dec. 2009. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5365307>.
- [LQ09a] José Paulo Leal and Ricardo Queirós. Crimsonhex: a service oriented repository of specialised learning objects. In *ICEIS 09 - 11th International Conference on Enterprise Information Systems, Milan, Italy*, volume 24 of *Lecture Notes in Business Information Processing*, pages 102–113. Springer-Verlag, LNBIP, Springer-Verlag, LNBIP, May 2009.
- [LQ09b] José Paulo Leal and Ricardo Queirós. crimsonhex: um repositório de objectos de aprendizagem. In *Jornadas Luso Brasileiras de Ensino e Tecnologia em Engenharia (JLBE09), Porto, Portugal*, pages 3–10, Fevereiro 2009.
- [LQ09c] José Paulo Leal and Ricardo Queirós. Defining programming problems as learning objects. In *ICCEIT - World Academy of Science, Engineering and Technology*, volume 58, pages 1033–1040, 2009.
- [LQ09d] José Paulo Leal and Ricardo Queirós. Designing a user interface for repositories of learning objects. In *International Conference on e-Learning (IADIS 2009), Algarve, Portugal*, June 2009.
- [LQ09e] José Paulo Leal and Ricardo Queirós. Extending the learning object definition to represent programming problems. In *Actas do Inforum - Simpósio de Informática*, pages 421–432, Lisbon, September 2009.
- [LQ10a] José Paulo Leal and Ricardo Queirós. elearning frameworks: a survey. In *International Technology, Education and Development Conference, Valencia, Spain*, March 2010.
- [LQ10b] José Paulo Leal and Ricardo Queirós. *From eLearning Systems to specialised eLearning Services*. Sello Editorial, Madrid, 2010.
- [LQ10c] José Paulo Leal and Ricardo Queirós. Integration of repositories in elearning systems. In *ICEIS 10 - 12th International Conference on Enterprise Information Systems*, Funchal, Madeira, Portugal, June 2010.
- [LQ10d] José Paulo Leal and Ricardo Queirós. Integration of repositories in moodle. In Alberto Simões, Daniela da Cruz, and José Carlos Ramalho, editors, *Conferência Nacional em XML, Aplicações e Tecnologias Aplicadas*, page 57–68, Vila do Conde, Maio 2010.

- [LQ10e] José Paulo Leal and Ricardo Queirós. Interfacing repositories of learning objects with support for programming problems. In *ACM-ICPC 3rd Competitive Learning Symposium, Harbin, China*, February 2010.
- [LQ10f] José Paulo Leal and Ricardo Queirós. Modelling non-trivial evaluation processes. In *CENTERIS 2010 - Conference on Enterprise Information Systems*, Viana do Castelo, Portugal, October 2010.
- [LQ10g] José Paulo Leal and Ricardo Queirós. Modelling text file evaluation processes. In *2010 International Workshop on Cognitive-based Interactive Computing and Web Wisdom (CICW'10)*, LNCS, Shanghai, China, December 2010. Springer-Verlag, Springer-Verlag.
- [LQ10h] José Paulo Leal and Ricardo Queirós. Specifying a programming exercises evaluation service on the e-framework. In Xiangfeng Luo, Marc Spaniol, Wolfgang Nejdl, and Wu Zhang, editors, *ICWL - 9th International Conference on Web-based Learning*, volume 6483/2010, pages 141–150, Shanghai, China, December 2010. Springer Lecture Notes in Computer Science (LNCS), Springer Lecture Notes in Computer Science (LNCS).
- [LQ11a] José Paulo Leal and Ricardo Queirós. *A comparative study on LMS interoperability*, pages 142–161. IGI-Global, 2011.
- [LQ11b] José Paulo Leal and Ricardo Queirós. Integrating the lms in service oriented elearning systems. *International Journal of Knowledge Society Research (IJKSR)*, IGI-Global, 2(2):1–12, 2011.
- [LQ11c] José Paulo Leal and Ricardo Queirós. Modelling a network of heterogeneous elearning systems. In *Fifth International Workshop on Enterprise Systems and Technology (I-WEST)*. Published by INSTICC, 2011.
- [LQ11d] José Paulo Leal and Ricardo Queirós. A programming exercise evaluation service for mooshak. ACM-ICPC World Final CLIS (Competitive Learning Institute Symposium), ACM-ICPC World Final CLIS (Competitive Learning Institute Symposium), 2011.
- [LQ11e] José Paulo Leal and Ricardo Queirós. Using the learning tools interoperability framework for lms integration in service oriented architectures. In *"Technology Enhanced Learning", TECH-EDUCATION'11*. Springer Verlag, May 2011 2011.
- [LQF10] José Paulo Leal, Ricardo Queirós, and Duarte Ferreira. A contribution to the e-framework – a specification of a programming exercise evaluation service. Technical Report 03, DCC / FCUP, Porto, Portugal, June 2010.

- [LS03] José Paulo Leal and Fernando M. A. Silva. Mooshak: a web-based multi-site programming contest system. *Softw., Pract. Exper.*, 33(6):567–581, 2003.
- [Mas10] D. et al Massart. Taming the metadata beast: Ilox. *D-Lib Magazine*, 16(11/12), 2010. <http://www.dlib.org/dlib/november10/massart/11massart.html>.
- [McC06] S. H. McCallum. A look at new information retrieval protocols: Sru, opensearch/a9, cql, and xquery. In *In World Library and Information Congress: 72nd IFLA General Conference and Council - IFLA*, 2006. <http://archive.ifla.org/IV/ifla72/papers/102-McCallum-en.pdf>.
- [McG08] R. McGreal. A typology of learning object repositories. In Heimo H. Adelsberger, Kinshuk, Jan M. Pawlowski, and Demetrios G. Sampson, editors, *Handbook on Information Technologies for Education and Training*, International Handbooks on Information Systems, pages 5–28. Springer Berlin Heidelberg, 2008.
- [MdL01] Marcus Eduardo Markiewicz and Carlos J. P. de Lucena. Object oriented framework development. *Crossroads*, 7:3–9, July 2001. <http://doi.acm.org/10.1145/372765.372771>.
- [Mei02] Wolfgang Meier. exist: An open source native xml database. In *Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops*, pages 169–183. Springer, 2002.
- [MGH98] Fatima Z. Mansouri, Cleveland A. Gibbon, and Colin A. Higgins. Pram: prolog automatic marker. In *ITiCSE*, pages 166–170, 1998.
- [MKKN05] Lauri Malmi, Ville Karavirta, Ari Korhonen, and Jussi Nikander. Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *J. Educ. Resour. Comput.*, 5, September 2005.
- [MMR06] Amit Kumar Mandal, Chittaranjan Mandal, and Christopher M P Reade. Architecture of an automatic program evaluation system. *CSIE*, 2006. <http://sit.iitkgp.ernet.in/~chitta/pubs/CSIEAIT06-p152.pdf>.
- [Mor07] Edna H. Mory. *Feedback Research Revisited*. Association for Educational Communications and Technology, 2007.
- [MR03] Robin Mason and D Rehak. Keeping the learning in learning objects. In A Littlejohn, editor, *Reusing online resources: a sustainable approach to e-learning*, pages 20–34. Kogan Page, London, 2003. <http://oro.open.ac.uk/800/>.

- [Nic01] M. Nichani. Lcms = lms + cms [rlos] – how does this affect the learner? the instructional designer? ELearningPost, 2001.
- [Nie94] Jakob Nielsen. *Usability engineering*. Morgan Kaufmann Publishers, San Francisco, Calif., 1994.
- [NNH99] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.
- [OD09] X Ochoa and E Duval. Quantitative analysis of learning object repositories, 2009.
- [OG06] Jackie O’Kelly and J. Paul Gibson. Robocode & problem-based learning: a non-prescriptive approach to teaching programming. *SIGCSE Bull.*, 38(3):217–221, June 2006.
- [OKVD11] Xavier Ochoa, Joris Klerkx, Bram Vandeputte, and Erik Duval. On the use of learning object metadata: The globe experience. In *EC-TEL*, pages 271–284, 2011. DBLP, <http://dblp.uni-trier.de>.
- [Onc11] Cakir H. Oncu, S. Research in online learning environments: Priorities and methodologies. *Computers and Education*, 57(1):1098–1108, 2011. cited By (since 1996) 3.
- [PRS⁺03] Yusuf Pisan, Debbie Richards, Anthony Sloane, Helena Koncek, and Simon Mitchell. Submit! a web-based system for automatic program critiquing. In *Proceedings of the fifth Australasian conference on Computing education - Volume 20*, ACE ’03, pages 59–68, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.
- [QL09] Ricardo Queirós and José Paulo Leal. Interoperability in pedagogical elearning services. In *Doctoral Consortium, ICEIS 2009: 11th International Conference on Enterprise Information Systems, Milan, Italy*, May 2009.
- [QL11a] Ricardo Queirós and José Paulo Leal. Modelling an elearning environment for learning programming languages. *Learning Technology Newsletter of IEEE Computer Society’s Technical Committee on Learning Technology (TCLT)*., 2011.
- [QL11b] Ricardo Queirós and José Paulo Leal. Programming exercises interoperability language. ACM-ICPC World Final CLIS (Competitive Learning Institute Symposium), ACM-ICPC World Final CLIS (Competitive Learning Institute Symposium), 2011.

- [QL11c] Ricardo Queirós and José Paulo Leal. A survey on elearning content standardization. In *"World Summit on the Knowledge Society", WSKS'11*. Springer Verlag, September 2011 2011.
- [QL11d] Ricardo Queirós and José Paulo Leal. Using the common cartridge profile to enhance learning content interoperability. In *ECEL 2011: The 7th European Conference on e-Learning, Brighton, UK*, November 2011.
- [QL11e] Ricardo Queirós and José Paulo Leal. Pexil: Programming exercises interoperability language. Conferência - XML: Aplicações e Tecnologias Associadas (XATA), 2011.
- [QL12a] Ricardo Queirós and José Paulo Leal. crimsonhex: a learning objects repository for programming exercises. *Softw., Pract. Exper.*, 2012. (to appear).
- [QL12b] Ricardo Queirós and José Paulo Leal. Petcha - a programming exercises teaching assistant. In *ACM SIGCSE 17th Annual Conference on Innovation and Technology in Computer Science Education*, Haifa, Israel, July 2012 2012. ACM.
- [QL12c] Ricardo Queirós and José Paulo Leal. Programming exercises evaluation systems: an interoperability survey. In *4th International Conference on Computer Supported Education*, Porto, Portugal, April 2012.
- [Ree89] Kenneth A. Reek. The try system -or- how to avoid testing student programs. *SIGCSE Bull.*, 21:112–116, February 1989.
- [Reh03] Mason R. Rehak, D. R. Keeping the learning in learning objects. In *Littlejohn, A. (Ed.) Reusing online resources: a sustainable approach to e-Learning*, pages 22–30, 2003.
- [RKK04] Timo Rongas, Arto Kaarna, and Heikki Kälviäinen. Classification of computerized learning tools for introductory programming courses: Learning approach. In Kinshuk, Chee-Kit Looi, Erkki Sutinen, Demetrios G. Sampson, Ignacio Aedo, Lorna Uden, and Esko Kähkönen, editors, *ICALT*. IEEE Computer Society, 2004.
- [Rob02] Eddie Robertsson. Combining schematron with other xml schema languages. Technical report, -, 2002.
- [Rog03] Sally A Rogers. Developing an institutional knowledge bank at ohio state university: From concept to action plan. *portal Libraries and the Academy*, 3(1):125–136, 2003.

- [RRR03] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13:137–172, 2003.
- [RSA06] E Rodríguez, M A Sicilia, and Sinuhe Arroyo. *Bridging the semantic gap in standards-based learning object repositories*, pages 478–483. 2006.
- [RSS10] Repository software survey. Repositories Support Project, 2010.
- [RSZ10] R. Romli, S. Sulaiman, and K.Z. Zamli. Automatic programming assessment and test data generation a review on its approaches. In *Information Technology (ITSim), 2010 International Symposium in*, volume 3, pages 1186–1192, june 2010.
- [RWS06] William Reilly, Robert Wolfe, and MacKenzie Smith. Mit’s cwspace project: packaging metadata for archiving educational content in dspace. *Int. J. Digit. Libr.*, 6(2):139–147, April 2006.
- [SB06] Carsten Schulte and Jens Bennedsen. What do teachers teach in introductory programming? In *Proceedings of the second international workshop on Computing education research, ICER ’06*, pages 17–28, New York, NY, USA, 2006. ACM.
- [SG10] Michael Striewe and Michael Goedicke. Visualizing data structures in an e-learning system. In *CSEDU*, pages 172–179, 2010.
- [SHP⁺06] Jaime Spacco, David Hovemeyer, William Pugh, Fawzi Emad, Jeffrey K. Hollingsworth, and Nelson Padua-Perez. Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. *SIGCSE Bull.*, 38(3):13–17, June 2006.
- [SKA08] Atiq Siddiqui, Mehmood Khan, and Sohail Akhtar. Supply chain simulator: A scenario-based educational tool to enhance student learning. *Comput. Educ.*, 51(1):252–261, August 2008.
- [SMK01] Riku Saikkonen, Lauri Malmi, and Ari Korhonen. Fully automatic assessment of programming exercises. *SIGCSE Bull.*, 33:133–136, June 2001.
- [SMvA⁺05] Bernd Simon, David Massart, Frans van Assche, Stefaan Ternier, Erik Duval, Stefan Brantner, Daniel Olmedilla, and Zoltán Miklós. A simple query interface for interoperable learning repositories. In *PROCEEDINGS OF THE 1ST WORKSHOP ON INTEROPERABILITY OF WEB-BASED EDUCATIONAL SYSTEMS*, pages 11–18, 2005.

- [SZC12] Demetrios G. Sampson, Panagiotis Zervas, and George Chloros. Supporting the process of developing and managing lom application profiles: The ask-lom-ap tool. *IEEE TRANSACTIONS ON LEARNING TECHNOLOGIES*, 2012.
- [Tan09a] Yu Y. T. & Poon C. K. Tang, C. M. An approach towards automatic testing of student programs using token patterns. In *In Proceedings of the 17th International Conference on Computers in Education (ICCE 2009)*, pages 188–190, 2009.
- [Tan09b] Yu Y. T. & Poon C. K. Tang, C. M. Automated systems for testing student programs: Practical issues and requirements. In *In Proceedings of the International Workshop on Strategies for Practical Integration of Emerging and Contemporary Technologies in Assessment and Learning*, pages 132–136, 2009.
- [TC10] Poon C.K. Tang C.M., Yu Y. T. A review of the strategies for output correctness determination in automated assessment of student programs. In *In Proceedings of Global Chinese Conference on Computers in Education*, 2010.
- [Tea06] JORUM Team. E-learning repository systems research watch. Technical report, JISC, 2006.
- [Ter08] Stefaan Ternier. *Standards Based Interoperability for Searching in and Publishing to Learning Object Repositories (Interoperabiliteit voor het publiceren en ontsluiten van leerobjecten in repositories met gebruik van standaarden)*. PhD thesis, K.U.Leuven, March 2008.
- [TGPS08] G. Tremblay, F. Guérin, A. Pons, and A. Salah. Oto, a generic and extensible tool for marking programming assignments. *Softw. Pract. Exper.*, 38(3):307–333, March 2008.
- [TMT⁺10] S. Ternier, D. Massart, M. Totschnig, J. Klerkx, and E. Duval. The simple publishing interface (spi). *D-Lib Magazine*, 16(9/10), 2010. <http://www.dlib.org/dlib/september10/ternier/09ternier.html>.
- [Tru07] Nghi Khue Dinh Truong. *A web-based programming environment for novice programmers*. PhD thesis, Queensland University of Technology, 2007.
- [TS05] White A. Tastle, J. and P. Shackleton. E-learning in higher education: the challenge, effort, and return of investment. *International Journal on ELearning*, 2005.
- [Tsu10] Takashi Tsunakawa. *Pivotal Approach for Lexical Translation*. PhD thesis, University of Tokyo, 2010.

- [Tzi09] Manouselis N. & Vuorikari R. Tzikopoulos, A. An overview of learning object repositories. In *In T. Halpin (Ed.), Selected Readings on Database Technologies and Applications*. IGI Global, 2009.
- [VA06] Iraklis Varlamis and Ioannis Apostolakis. The present and future of standards for e-learning technologies. *Interdisciplinary Journal of Knowledge and Learning Objects*, 2, 2006. <http://www.ijello.org/Volume2/v2p059-076Varlamis.pdf>.
- [VD03] Maarten Vansteenkiste and Edward L. Deci. Competitively contingent rewards and intrinsic motivation: Can losers remain motivated? *Motivation and Emotion*, 27:273–299, 2003. 10.1023/A:1026259005264.
- [Ver08] Tom Verhoeff. Programming task packages: Peach exchange format. *International Journal Olympiads In Informatics*, 2:192–207, 2008.
- [VRV⁺11] Elena Verdú, Luisa M. Regueras, María J. Verdú, José Paulo Leal, Juan P. de Castro, and Ricardo Queirós. A distributed system for learning programming on-line. *Computers & Education*, 2011.
- [War04] Jewel Ward. Unqualified dublin core usage in oai-pmh data providers. *OCLC Systems & Services*, 20(1):40–47, 2004. <http://dx.doi.org/10.1108/10650750410527322>.
- [WBR04] Scott Wilson, Kerry Blinco, and Daniel Rehak. Service-Oriented Frameworks: Modelling the infrastructure for the next generation of e-Learning Systems. Technical report, JISC Report, July 2004. http://www.jisc.ac.uk/uploaded_documents/AltilabServiceOrientedFrameworks.pdf.
- [Wil05] Goldberg M. Williams, J. The evolution of e-learning. Universitas 21 Global, 2005.
- [WLK04] Susan Wiedenbeck, Deborah Labelle, and Vennila N. R. Kain. Factors affecting course outcomes in introductory programming. In *In 16th Annual Workshop of the Psychology of Programming Interest Group*, pages 97–109, 2004.
- [WW08] Fu Lee Wang and Tak-Lam Wong. Designing programming exercises with computer assisted instruction. In *Proceedings of the 1st international conference on Hybrid Learning and Education, ICHL '08*, pages 283–293, Berlin, Heidelberg, 2008. Springer-Verlag.
- [XC11] J. Xavier and A. Coelho. Computer-based assessment system for e-learning applied to programming education. In *ICERI2011 Proceedings*, 4th Interna-

tional Conference of Education, Research and Innovations, pages 3738–3747.
IATED, 14-16 November, 2011 2011.