



BIROn - Birkbeck Institutional Research Online

Enabling Open Access to Birkbeck's Research Degree output

Simplifying authoring and facilitating component reuse of programming tutors

<https://eprints.bbk.ac.uk/id/eprint/49182/>

Version: Full Version

Citation: Karkalas, Sokratis (2022) Simplifying authoring and facilitating component reuse of programming tutors. [Thesis] (Unpublished)

© 2020 The Author(s)

All material available through BIROn is protected by intellectual property law, including copyright law.

Any use made of the contents should comply with the relevant law.

[Deposit Guide](#)
Contact: [email](#)

SIMPLIFYING AUTHORIZING AND
FACILITATING COMPONENT REUSE
OF PROGRAMMING TUTORS

Sokratis Karkalas

2021

A thesis submitted to the University of London
for the degree of Doctor of Philosophy

Birkbeck, University of London
Department of Computer Science and Information Systems

Declaration

This thesis is the result of my own work, except where explicitly acknowledged in the text.

Abstract

Learning programming is very hard, especially during the early stages. Programming is an exploratory activity and therefore it is more natural to learn it through exploration. Freedom and lack of structure in exploratory learning offer more opportunities for experimentation and discovery of knowledge but at the same time that requires substantial support. For the same reasons provision of support is more challenging and costly in this context. Typical traditional intelligent tutoring systems are highly controllable environments that offer guided learning. Modern environments are more open and exploratory but they lack intelligence and adaptability. There is an emerging need for systems that are both exploratory and intelligent but authoring them is a very challenging task.

The intention of this thesis is not to offer a new exploratory and intelligent learning platform that teaches programming more effectively but to provide the architectural framework, techniques and tools that can be used to develop intelligent tutors for exploratory learning with ease. This thesis is concerned with both task-dependent and task-independent intelligent support. The latter is expected in systems that offer free exploration or in situations where students work with ill-defined problems and define their own tasks dynamically. In these situations there is no explicit knowledge in the system about task-specific objectives.

This thesis presents a process used to identify common student misconceptions for early programming and transform them into task-independent intelligent support. It also presents a novel methodology that can be used to lower the cognitive load and entry threshold for prospective authors of task-dependent support.

Designing and developing support is not enough if the tutors cannot take advantage of the various learning environments available and combine them with intelligent support components. For this reason, this thesis presents a novel approach that simplifies the integration and interoperability of diverse and heterogeneous components so

that authors can synthesise dynamic learning environments with minimal overhead. Having the components and being able to integrate them may be problematic if there is no understanding of the system as a whole. An overview of what is needed to foster intelligent support for programming is given in an architectural framework that shows how the various components are logically interrelated with each other and shows how they should be combined together in an incremental manner.

Finally, a tool to facilitate reusability of existing functionality is presented. This tool can be used to define new and existing languages that can be used in the context of learning platforms either to simplify authoring of support or to enable teaching programming through manipulation of existing learning environments.

The outcomes of this research are materialised in a proof of concept that shows how all the components presented in the text can be combined together to simplify authoring of intelligent support and facilitate reusability of functionality.

Acknowledgements

I would like to express my sincere gratitude to all of those that contributed to this thesis and supported me throughout this process. In particular, I want to thank my supervisors Dr. Sergio Gutiérrez-Santos and Dr. Manolis Mavrikis for their invaluable guidance and encouragement during the project. I would also like to thank Dr. Keith Mannock for the logistical help he provided me during the first stages of my degree.

I am thankful to all the students that participated and contributed to the studies that took place. Also, I am thankful to my fellow teaching assistants that contributed significantly with their work and input in various stages of the project.

I would like to express my thanks to my colleagues at the UCL Knowledge Lab and my fellow engineers at Sopra Steria Ltd for their participation in the studies and their input. Special thanks deserve my colleagues at the Educational Technology Lab of the National and Kapodistrian University of Athens for their invaluable input and feedback throughout the project.

Finally, I would like to thank a dear friend and former lecturer at the Department of Computer Science and Information Systems, Dr. Roger Mitton, for believing in me and helping me in my first steps when I arrived in the UK in 2012.

Contents

Contents	6
List of Figures	12
List of Tables	15
1 Introduction	16
1.1 Motivation	17
1.1.1 The Need for Programming Skills	17
1.1.2 The Need for Automated Support	18
1.1.3 The Need for more accessible AI	21
1.1.4 The Need for Intelligent Exploratory Learning Environments	23
1.1.5 The Need for Flexible Integration and Interoperability	24
1.2 Research Objectives	28
1.2.1 Challenges Translated into Research Objectives	29
1.3 Research Methodology	31
1.3.1 Literature Review	31
1.3.2 Reasoning Behind the Research Project - Design Thinking	34
1.3.3 The Project Step by Step	35
1.4 Thesis Outline	44

2	Related Work	46
2.1	Educational Programming Environments	47
2.1.1	Turtle Graphics in LOGO	47
2.1.2	Karel	48
2.1.3	Toontalk	48
2.1.4	Alice	49
2.1.5	BlueJ	50
2.1.6	Greenfoot	51
2.1.7	SALESPOINT	51
2.1.8	Malt+	52
2.1.9	Scratch	53
2.1.10	Discussion	54
2.2	Exploratory Learning Systems	55
2.2.1	Exploratory Learning	55
2.2.2	Exploratory Learning Environments	56
2.2.3	Discussion	65
2.3	Tutoring Systems for Programming	67
2.3.1	Programming Environments	68
2.3.2	Debugging Aids	74
2.3.3	Intelligent Tutoring Systems	77
2.3.4	Intelligent Programming Environments	80
2.3.5	Discussion	84
2.4	Automated Support Authoring Tools	86
2.4.1	SQL-Tutor	86
2.4.2	ASPIRE	87
2.4.3	Diligent	88
2.4.4	Disciple	89
2.4.5	Demonstr8	90

2.4.6	CTAT	91
2.4.7	Automatic Rule Authoring System for CTAT	92
2.4.8	SimStudent	92
2.4.9	GIFT	93
2.4.10	The FRAME Approach	93
2.4.11	Discussion	94
2.5	Integration and Interoperability	96
2.5.1	Technologies used in Learning Management Systems	101
2.5.2	Epiphytic Integration Systems	102
2.5.3	Discussion	106
2.6	Synthesis of Related Work and Revised Research Objectives	108
3	Exploring Possibilities	123
3.1	Literature Review and Domain Analysis - An Outline	125
3.2	Educational Ethnographic Study	127
3.2.1	The Data Collection Process	129
3.2.2	Managing Bias and Subjectivity	131
3.2.3	Thematic Analysis	133
3.3	Common Student Misconceptions in Elementary Programming	137
3.3.1	Importance of Student Misconceptions	139
3.4	Understanding Challenges by Developing a Prototype	141
3.5	Developing FLIP	143
3.5.1	Knowledge Elicitation	143
3.5.2	Knowledge Representation	144
3.5.3	Knowledge Processing	148
3.5.4	The PoC	150
3.6	An Important Outcome: The Intelligent Tutor Layered Architecture . . .	158
3.7	Usability Testing	166

3.7.1	Participants	166
3.7.2	Method	167
3.7.3	Results	168
3.7.4	Discussion	168
4	Completing the Working Principles	170
4.1	Literature Review and Domain Analysis - An Outline	173
4.2	User Centric Design through a Requirements Elicitation Workshop . . .	175
4.3	Implementation of WIIL	178
4.4	The Web Integration and Interoperability Layer (WIIL)	179
4.4.1	Web Components	180
4.4.2	Design Considerations	181
4.4.3	Browser Security	184
4.4.4	The Technique	188
4.4.5	The Ladders Activity - A PoC	205
4.4.6	GeoGebra Coding - A PoC	206
4.4.7	Results	207
4.5	Learning Environment vs Platform	209
4.6	An architectural aspect of Learning Platforms	210
4.6.1	Tightly Coupled	210
4.6.2	Loosely Coupled	210
4.7	Learning Platforms as Ecosystems of Diverse Components	212
4.8	Implementation of AuthELO	214
4.9	AuthELO	216
4.9.1	Design	217
4.9.2	Architecture	222
4.9.3	Integration at a Technical Level	224
4.9.4	The PoC	226

5	Evaluation of AuthELO	234
5.1	First Evaluation	235
5.2	Second Evaluation	242
6	Addressing new Requirements	248
6.1	Requirements Elicitation	250
6.2	Making Authoring Simpler	251
6.3	Implementation of LFT	252
6.4	The Lingua Franca Transformer (LFT)	254
6.4.1	Architecture	256
6.4.2	The Language Specification Syntax	259
6.4.3	The Tool	260
6.4.4	Implementation Details	262
7	The Microworld Learning Platform	267
7.1	A Conceptual Overview	268
7.2	The Platform	269
7.3	The Basic Workflow	270
7.4	Creating and Enhancing Instances of Learning Components	276
8	Contributions	281
8.1	Part 1 - Facilitate Reuse	282
8.2	Part 2 - Simplify Authoring	283
8.3	Part 3 - Miscellaneous	284
8.4	How it all fits together	285
9	Future Work	287
9.1	Visual Integration Editor for WIIL	288
9.2	Use AuthELO to handle Common Student Misconceptions	289
9.3	Enhance AuthELO with high-level Authoring Languages	289

9.4 Enhance AuthELO with Machine Learning Techniques	290
Bibliography	291
A Sample Rules	317
A.1 Understanding the Role of Variable Declaration	317
A.2 Understanding the Difference Between Variable Values and Literal Values	318
A.3 Understanding the Necessity of Variables/Constants	320
A.4 Understanding the Necessity of Variables when Referring to Array Length	323
A.5 Understanding off-by-one Errors with Arrays in Loops	325
B Literature Review Strategy Used	330
C Observation Sheet	332
D Usability Test Material	335
D.1 Day 1	335
D.2 Day 2	339

List of Figures

1.1	The project step by step	43
2.1	The CTAT system	95
2.2	The FRAME Approach	95
3.1	The Rule Editor	145
3.2	Facts	152
3.3	Sample Rule	153
3.4	The Consequent	154
3.5	The Reasoner Architecture	154
3.6	Misconceptions identified by FLIP	155
3.7	Initial Support	155
3.8	Third Level of Support	156
3.9	Fourth Level of Support	156
3.10	Fifth Level of Support	157
3.11	Sixth Level of Support	157
4.1	The WIIL stack	196
4.2	The LTI Protocol	202
4.3	The WIIL protocol	203
4.4	The Ladders Interface	206
4.5	The Blockly Interface	207

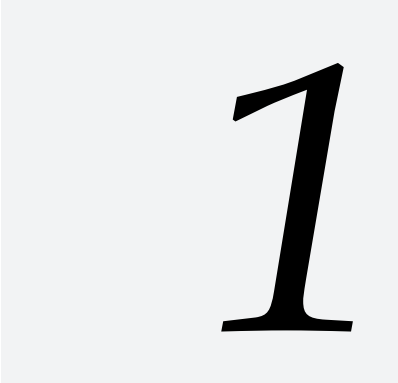
4.6	The JS Interface	208
4.7	The Architecture	224
4.8	The Widget View	228
4.9	The Logging View	229
4.10	The Analysis View	230
4.11	The Feedback View	231
4.12	The Messages View	232
4.13	Feedback Testing	233
6.1	Typical widget manipulation	256
6.2	Widget manipulation with other languages	257
6.3	Widget manipulation with any language	257
6.4	The LFT Architecture	258
6.5	The LFT Editor	261
6.6	The LFT Comparisons Section	262
6.7	The LFT Comparisons Section	263
6.8	The LFT Testbed Section	263
6.9	LFT Real-time Updates	264
6.10	LFT Interactive AST Visualisation	264
7.1	Abstract Factory Pattern	268
7.2	Abstract Factory Pattern in Detail	268
7.3	The MLP Frontpage	271
7.4	The Author's Homepage	272
7.5	Activity Deployment	273
7.6	The Students' Terminal	273
7.7	Connecting to the System	274
7.8	Synchronous Operation between Teacher and Students	274
7.9	Activity in Student Terminal	275

7.10 Real-time Support	275
7.11 Creating a new Activity	277
7.12 Enhancing an Activity with Automated Support	278
7.13 Configuring Logging Rules	278
7.14 Preparing the Facts for the Feedback Rules	279
7.15 Developing the Feedback Rules	279
7.16 Editing Feedback Messages	280
7.17 Checking Rule Validity	280
8.1 Contributions Chart	285
A.1 Understanding the Role of Variable Declaration	319
A.2 Understanding the Difference Between Variable Values and Literal Values	321
A.3 Understanding the Necessity of Variables/Constants	323
A.4 Understanding the Necessity of Variables when Referring to Array Length	326
A.5 Understanding off-by-one Errors with Arrays in Loops	329

List of Tables

- 3.1 Procedural Programming Concepts 137
- 3.2 Other Concepts 138
- 3.3 Misconception Indicators 168

- 5.1 Time Savings with AuthELO 239
- 5.2 Results for Part 1 - AuthELO 245
- 5.3 Results for Part 2 - AuthELO 246
- 5.4 Results for Part 3 - AuthELO 247



Introduction

1.1 Motivation

1.1.1 The Need for Programming Skills

Programming can be seen both as a skill and as a learning activity. The way society sees programming affects the way we approach it as a learning activity. The different degrees of adoption by schools at different periods in the past is indicative of that.

In the 80's schools demonstrated great enthusiasm in supporting programming as a learning activity (Resnick et al. 2009). By the mid-90's there was a serious setback that lasted a decade. As indicated in Noss & Hoyles (1996) one of the reasons was that the educational system was immature and not ready to respond effectively to this demand. There were practical problems that hindered the process i.e. lack of qualified teachers and lack of subject-matter integration. This negative trend became even more intense when educational software with sophisticated graphical user interfaces was introduced. Event-driven environments that enable dynamic manipulation of graphical elements was seen as a more appealing alternative to programming. Programming was seen as a kind of unnecessary noise to doing interesting things with digital media (Kynigos 2015). However, during the last decade, this situation seems to have dramatically reverted (Kafai & Burke 2013). Programming has been recognized as an important learning activity and a plethora of educational tools have been developed in order to promote student engagement with programming design and coding. Tools such as Scratch, Alice, ToonTalk, Imagine Logo and later NetLogo, Kodu and GreenFoot have become very popular and attracted people of all ages to engage with programming.

The society nowadays sees computational media and programming in particular as an enabler that through algorithmic thinking can promote and facilitate the development of powerful ideas in other scientific disciplines (Blikstein 2013) and aspects of human activity in general. But how did that happen? What is the causation of this big comeback of programming? Programming is a craft and as such it is by nature a creative activity. The sense of creativity through making 'tangible' artifacts is inherent

in programming. Therefore, it is not surprising that it has been at the centre of global movements like the 'maker culture' in the mid 00's. This movement was followed by other and more 'focused' ones like "hour of code" and "coding for all" in the early 10's.

These movements sparked and invigorated this new emerging culture around coding that influenced the way we see and use programming in general. This change gradually translated into the release of numerous tools for programming and a significant increase of programming-based learning activities in formal and informal education. People even started designing digital fabrication activities (Blikstein & Krannich 2013) through programming of tangible objects (physical computing). Programming is now seen as the literacy of the future. There is a constantly growing demand for educational systems that promote computational thinking as a means of making peoples' lives better and there is also a huge demand for people with coding skills in the global industry.

1.1.2 The Need for Automated Support

Learning computer programming is particularly hard especially during the early stages (Soloway 1986, Jenkins 2002, Robins et al. 2003). Programming is a craft and requires the development of practical skills that can only be learnt through practical training (Vihavainen et al. 2011). Typically, learning takes place either in a workplace through apprenticeships or in University computer laboratories through training courses. In the latter case students are given problem and/or inquiry-based learning scenarios (Savery 2006) and work individually or in pairs under the supervision of tutors. Students are not expected to follow instructions and repeat actions. They are encouraged to explore their own strategies, designs, patterns and techniques through experimentation. They are expected to discover knowledge in an exploratory manner (Huitt 2003, Vygotskii et al. 1978). Learning this way is painful. It involves investigation, planning, tactics and action. Tutors play a crucial role in this process. They are not just people

that merely give instructions and expect answers. They actively engage in the process as facilitators and they contribute by identifying problems, giving directions and confirming acceptable solutions.

It has been established that considerable effort is required by tutors to ensure effective learning in such open-ended contexts (Kirschner et al. 2006, Kynigos 1992, Mayer 2004). Time and human resources in computer laboratories are limited. The effectiveness of this process highly depends on whether utilisation of these resources is optimal or not. Students and tutors have their own individual characteristics, problems and idiosyncrasies. Students expect individualised support that reflects their particular misconceptions and practices. Tutors are expected to make bias-free and informed decisions about the type and level of support needed in every case and respond accordingly. The latter presupposes that tutors have a deep knowledge of students' profiles and the ability to analyse previous activity on the spot in order to provide suitable and adequate support in every case. Support in this context is a multi-faceted and complex task that requires a lot of preparation, expertise, time and resources. Decisions must be based on a multitude of criteria and a considerable amount of data about students. It is evident that human tutors cannot respond effectively to these challenges without help.

Another important aspect of learning in a computer laboratory is the sequence of actions that take place during a learning cycle (Kolb et al. 1984, Konak et al. 2014). There is a pattern that students follow when they engage with a task. The sequence of actions they execute follows a cyclical process. In every round students attempt to code something that brings them closer to the completion of the task at hand. Sometimes this is interrupted by the inherent inability of the student to move forward. That can be lack of knowledge or a misconception. This is the point where the student hits the inner circle of their particular zone of proximal development (ZPD) (Vygotskiï et al. 1978). The only way to overcome the problem in this case is to receive enough and relevant help in a timely fashion. Typically the tutor intervenes and provides the help

needed so that the student can move on and complete the cycle. The student conceptualises the issue and then confirms the validity of the new knowledge through active experimentation. Computer laboratories are especially busy during the early stages of learning. These interruptions are very frequent and support may not be enough. Tutors have to prioritise and provide help to many people in a very limited time and that apparently can have a negative effect on the quality and quantity of service provided. If these cycles get interrupted, then learning gets interrupted. If support is not adequate and cannot be provided timely, then inevitably the learning process becomes less effective.

Both of the above (limiting) factors influence to a great extent the effectiveness of the learning process. Inadequate support potentially means that students may not be able to engage with the subject in the most constructive way and as a consequence of that they may not be able to exploit their full potential and achieve the best possible learning outcome. If technology can be used to compensate for these limitations then it is expected that the learning process will be much more effective, fair and inclusive. That will also have an effect on the way people approach the learning process of programming even in the early stages. If a significant proportion of teaching can be shifted from human teachers to automated agents, then it might be possible that a significant proportion of learning can take place in different contexts without the pressure of limited time and resources.

It is evident from the above that programming is by nature an exploratory activity where people approach solutions and knowledge through discovery. This is a deeply experiential process that is best conveyed through the constructionist approach (Papert 1993) about learning. Teaching and learning that way requires a certain degree of freedom and is challenging. Learning environments and tools are expected to promote creative expression and exploration, and enhance skills like critical thinking and problem solving but all that comes at a cost. The learners need a significant amount of support to utilise effectively the tools and maximise their potential. It is imperative

that we take advantage of this opportunity and find ways to either utilise existing tools better or create new ones and provide better support to the development of computational thinking.

1.1.3 The Need for more accessible AI

The history of Artificial Intelligence (AI) is full of setbacks and comebacks called winters and summers respectively (Grudin 2009). Despite the difficulties, the community devised numerous systems and technologies since the early 60's. The biggest obstacles have always been the high complexity of AI problems and the consequent cost. There were also technological barriers like the lack of adequate computer power and memory and the ability to gather enormous amounts of training datasets in databases. That inevitably lead to negative publicity and AI was perceived as a theoretical field of computer science with no applicability to real world problems and therefore no real value. In the field of technology enhanced learning (TEL) there have been numerous attempts to devise intelligent systems, yet very little of this outcome has been utilised in real world applications. It is indicative that contemporary learning platforms (in the wider sense), commercial or not, make very little use of AI. Teaching-oriented integrated development environments (like BlueJ), virtual learning environments (like Moodle), Web-based learning platforms (like Khan Academy) or even specialised exploratory learning environments (like Geogebra) lack AI support completely.

In the context of programming the first attempts to utilize technology into supporting programming can be traced back in the 80s. The first tutoring systems of this era (Brown & Burton 1978, Reiser et al. 1985, Johnson & Soloway 1985) were able to provide intelligent and personalized support to students. Another system that appeared a decade later with a somewhat different orientation is the ELM-ART (Brusilovsky et al. 1996). This was a courseware delivery system also equipped with intelligence and adaptivity. Other more recent systems are Mitrovic (2003), Sykes & Franek (2003),

Holland et al. (2009), Peylo et al. (2000). The system in Mitrovic (2003) is a constraint-based approach that does comparisons between student solutions and model solutions defined by tutors. Another system based on constraint-based modelling is J-LATTE (Holland et al. 2009). J-LATTE teaches Java and provides support for both design and implementation issues. Another system that teaches Java is Sykes & Franek (2003). This system is both intelligent and adaptive and the underlying technology used in an expert system supported by decision trees. The system in Peylo et al. (2000) teaches Prolog. In this case domain knowledge is represented as an ontology.

All of these systems present excellent examples of what is possible but from an educational point of view they fail to address an important aspect of the process. They all focus on task-dependent support for well-defined problem-based scenarios. They provide guidance in an intrusive and controllable manner and they fail to support discovery of knowledge through exploration that is essential in motivating students to actively make meaning of their learning process. These systems prove that the technology needed to provide intelligent support exists, yet it is still not mature enough to be used in the context of exploratory learning. Although, today the technological barriers are much lower, still the degree of adoption of intelligent support in learning systems is very low. Existing systems do not fit these educational requirements and building new systems is still perceived as a complex and expensive endeavour. The difference, though, between the past and the present is that AI has moved on from being seen as an over-ambitious and under-achieving field of research. Nowadays, AI is seen as a scientific field with promising ideas and great potential. Recent developments in machine learning algorithms combined with lower technological barriers present new opportunities and people are starting to use it in real applications. Its use in big data and data science is prevalent these days and the exponential growth of IoT will dramatically change the way we live and operate in the near future. These changes present new opportunities for technology enhanced education. It is imperative that we take advantage of this situation and promote the design and development

of techniques and tools that will give us the ability to utilise AI in a cost-effective manner and provide a greater level of high quality support in the educational process.

1.1.4 The Need for Intelligent Exploratory Learning Environments

In the previous sections a wide range of tools and platforms was presented. All of them fall in two broad categories. We have tools that are merely teaching-oriented development environments like BlueJ, Greenfoot, Alice, Karel, ToonTalk, LOGO-based Microworlds and Scratch (Kölling et al. 2003, Kölling 2010, Dann et al. 2000, Bergin et al. 1997a, 2005, Becker 2001, Morgado & Kahn 2008, Jenkins 2012, Maloney et al. 2008). These systems may use guided or exploratory learning but they are not intelligent and/or adaptive. There are also non-academic platforms like Khan Academy, Code school, Coursera and Udemy. These are typically used for guided learning and they are also not intelligent/adaptive. So, this first category comprises systems that are not intelligent/adaptive but they may offer some degree of freedom in terms of method of delivery (free/exploratory). The fact that there is freedom in the way the students can approach learning is of fundamental importance in programming because programming is by nature an exploratory activity. Nevertheless, as research suggests considerable effort is required in terms of teaching and guidance to ensure effective learning in such open-ended contexts (Kirschner et al. 2006, Kynigos 1992, Mayer 2004) and that means that the lack of automated support can be a seriously limiting factor in those systems.

The other category comprises systems like ELM-ART and J-LATTE (Brown & Burton 1978, Reiser et al. 1985, Johnson & Soloway 1985, Brusilovsky et al. 1996, Mitrovic 2003, Sykes & Franek 2003, Holland et al. 2009, Peylo et al. 2000). These systems follow the guided learning paradigm and are intelligent and/or adaptive. The distinguishing characteristic of those systems is that learning takes place in a strictly controllable environment, where support is imposed in an intrusive manner with the intention to

correct the students as early as possible and align them with what is expected to be an acceptable answer. Intelligence and adaptability in these systems proves that it is possible to employ very sophisticated technology to enhance teaching and learning but the fact that this takes place in a very deterministic manner may be too restrictive for systems that teach computer programming which ideally should allow a lot of freedom and experimentation.

This distinction between systems that are exploratory but not intelligent and systems that are intelligent but not exploratory gives a false sense of a dilemma. Does a system have to belong to the first or the second category? In reality, there is no dilemma. A system can be both exploratory and intelligent/adaptive and that can be a very powerful combination. The lack of systems that fulfill both objectives indicates the difficulty of the problem. In an exploratory learning environment there are potentially infinite paths the students might take and you don't know what student trajectories to anticipate. Authoring intelligent support for such a system is not a trivial task. However, these types of systems need significantly more support than others (Kirschner et al. 2006, Kynigos 1992, Mayer 2004). This necessity combined with the emerging free and exploratory learning paradigms signifies the need of contribution in this area.

1.1.5 The Need for Flexible Integration and Interoperability

Educators have always been trying to take advantage of technology affordances of their time and introduce innovative approaches in their teaching. One major component of teaching and learning is the development and delivery of courseware material. Since TCP/IP and the advent of WWW a multitude of systems emerged as precursors of modern Learning Management Systems (LMS). Systems like TrainingPartner ¹ (by GeoMetrix), Teachers Toolbox and Interactive Learning Network ² (by CourseInfo) and

¹<http://www.trainingpartner.com/>

²<http://en.wikipedia.org/wiki/CourseInfo>

ASAP³ (by ePath Learning) attempted to leverage the potential of new technologies and offer efficient organisation, management and dissemination of teaching resources. The appearance of modern LMSs like Moodle⁴, Blackboard⁵, Sakai⁶ and ANGEL Learning⁷ and the development of standards like SCORM⁸ changed radically the educational landscape (Bohl et al. 2002).

In the 00s the level of acceptance and adoption started to dramatically increase and the LMS established itself as the dominant technology for more than a decade. High degree of adoption is seemingly a good development. Global acceptance and wider adoption entail higher investment, faster implementation and better support and maintainability. Nowadays LMSs are mature and current implementations are stable, robust and reliable but that is just one side of the coin. LMSs ended up being treated like any other large-scale enterprise-wide application (Severance et al. 2010). Universities and other institutions have heavily invested on them and they consequently heavily depend on them. The implication of that is more stringent and conservative system management. The primary concern gradually shifted from education-related issues to considerations like system stability and reliability. As a consequence of that the process of integrating new functionality and instructional content became more difficult (Severance et al. 2010). Educationalists with the ability and willingness to experiment with new software and instructional content are suppressed for the sake of keeping systems robust for daily production. The new challenge now is the ability to balance innovation with stability.

The solution for stability was a shift to an architectural approach that offers the ability to decouple functionality into independent and self-sufficient components that interoperate via standardised communication protocols potentially over a network

³<http://www.eathlearning.com/services/lms/>

⁴<https://moodle.org/>

⁵<http://uki.blackboard.com/sites/international/globalmaster/>

⁶<https://sakaiproject.org/>

⁷http://www.angellearning.com/community/higher_ed.html

⁸<http://www.adlnet.gov/scorm.html>

(González et al. 2009). Componentisation offers better testability and resilience to system changes. A Service Oriented Architecture (SOA) implies that components are self-contained and have no dependency on specific products, technologies and architectures. Components can be hosted in different machines and functionality can be made available in the form of a service. In such a system, functionality can be easily added, removed or changed without affecting its operation and robustness (stability).

Innovation is empowered by the ability and the freedom to combine potentially heterogeneous learning components into formations that offer new and unique learning experiences. The solution employed for the stability problem led to the development of a new market for learning components. These components are typically fully-fledged web-based applications equipped with their own infrastructure in terms of security and operations and able to provide their services as stand-alone applications. The need for these applications to integrate with LMSs without sacrificing stability led to the development of standards like the IMS Learning Tools Interoperability (LTI) specification and OpenAjax. That gave the opportunity to educators, to develop new instructional material by combining possibly disparate and heterogeneous learning components together and integrate them with LMSs with transparency.

This is definitely a step forward but educators are still finding development in LMSs too restrictive for their purposes (Mott 2010). LMSs are designed to be very controllable and well-structured. That makes them very efficient in supporting administrative functions but relatively inflexible in supporting student-centered learning scenarios. Nowadays, educators see learning platforms as highly customisable mashup applications that don't necessarily impose prefabricated and static teaching-centered material to students. These platforms are like constantly evolving networks of components that may or may not have been designed for learning purposes. Educators want the freedom to easily develop formations of components that are available on the web and make them part of their educational practice with little or no configuration overhead in a way that resembles systems like Gurram et al. (2008). Modern learning

platforms are expected to support dynamic and authorable material that can be easily combined and processed both by teachers and students. Students should be able to configure their own learning environments by selecting tools that reflect their particular needs and circumstances. Web 2.0 or any other component (learning or not) should be able to integrate with such a system with minimal or no additional development. Typical examples of such components are wikis, blogs, services provided by social networking platforms and any other web tool that can be used to enhance productivity and satisfy basic communication and collaboration needs. Learning components like Geogebra, Cinderella, Scratch and eXpresser can also be added to the mix.

This new trend leads to systems that deviate from the basic LMS norm. These systems are called Personal Learning Environments (PLE) (Severance et al. 2008) or Personal Learning Networks (PLN) and are expected to be used in conjunction with LMSs. Another, more radical approach is the Open Learning Network (OLN) (Mott 2010) that unifies both worlds in a single platform. The logic behind these systems and its consequent architecture is radically different and promises greater flexibility, portability, adaptability and openness but the stringent and expensive to implement processes of LTI still remain and have to be used even when practically they have nothing to offer.

Nowadays, there is a multitude of components that exist solely in the browser. These components may not have dependencies on back-end services and typically there is no requirement for user authentication and other sensitive operations. Usually, these components do not have the ability to extend themselves and natively comply with interoperability standards. Integrating them with a learning platform using LTI and OpenAjax requires a significant overhead that is not necessary. Facilitating reusability in this case that minimises overheads will offer opportunities for the development of new learning environments and compelling learning experiences.

1.2 Research Objectives

The analysis presented so far reveals three interesting aspects of the current situation:

- **What is needed:** Computer programming is the literacy of the present and the future and learning programming deserves to be treated accordingly. Learning programming is very hard, especially during the early stages. Programming is by nature an exploratory activity and therefore it is more natural to learn it through exploratory learning. The level of support required in exploratory learning is much higher than it is in guided learning (Kirschner et al. 2006, Kynigos 1992, Mayer 2004). Exploratory learning takes place in more open environments where instruction may be less explicit. Learners discover knowledge through exploration of fuzzy, ill-defined and ill-structured problems. Personal inquiry, divergent thinking and self-directed learning is encouraged (Hannafin et al. 1999). In this context there is no predefined path for the course of actions that take place and thus the possible student trajectories are potentially infinite. For these reasons, provision of support in this context is a very challenging task. Automated support may be used to augment human support and optimise the learning process. Designing and implementing automated support for exploratory learning is a very challenging and demanding endeavour. According to Murray (1999) ITS design, development and evaluation is a challenging and costly process. Evidence show that it takes roughly 200 to 1000 hours of development time to create an hour of instruction for a guided learning system (Woolf & Cunningham 1987). Insertion of knowledge into a system requires highly skilled people with expertise in knowledge engineering and specialised software development as well as domain experts. If that is challenging and expensive then it is logical to assume that development of support for open environments is at least equally challenging as that, if not more. According to Gutierrez-Santos, Cocea & Magoulas (2010) the nature of student engagement in those environments combined with the con-

structivist intentions behind their design, make the development of support even more difficult and costly.

- **What is missing:** Old systems fail to support free, exploratory learning and constructivism. Current systems may offer exploratory learning but are not intelligent and/or adaptive. Technologies and tools exist but they are not being used to enhance these systems with adaptivity and intelligence so that they can support the learning process more efficiently. Automated support is literally nonexistent in current systems.
- **What is emerging:** Modern learning platforms that support dynamic and authorable material that can be easily combined and processed both by teachers and students. Nowadays, educators see learning platforms as highly customisable mashup applications that don't necessarily impose prefabricated and static teaching-centered material to students. These platforms are like constantly evolving networks of components. Educators want the freedom to easily develop formations of components that satisfy their particular needs and make them part of their educational practice with little or no configuration overhead.

1.2.1 Challenges Translated into Research Objectives

As illustrated in the previous section the greatest challenge identified is to enable the development of intelligent support and adaptability for programming education in an exploratory context. Under these settings the level of complexity expected is quite substantial as it is of course the respective cost of development. This challenge may have many facets. Reducing the cost of development may be related to reducing the complexity of the authoring process. It may also be the case that a process applicable to a wide range of use cases (possibly dissimilar in nature) can further reduce the cost due to reusability. Reusability on the other hand implies that there is an architectural framework in place as well as certain integration and interoperability techniques

to allow the actual application of the process on different (possibly diverse) learning components in a logically organised and technically feasible manner. The technical challenge to enable reuse in this case should not outperform the benefit of having it. All of this constitutes a complex mix of issues that can be logically categorised under the following two major components of this work:

- Facilitate reuse of diverse web components
- Simplify authoring of intelligent tutors for programming education in exploratory learning environments

The aim of this research is thus to devise techniques, methods, frameworks and tools that can facilitate reusability of existing functionality with minimal overhead and simplify the development of intelligent support for educational programming environments that operate in an exploratory context.

1.3 Research Methodology

Overall the research conducted in this project lies under the principles of the design thinking methodology (Melles et al. 2015). In this section we present a brief literature review of the related methodologies and a detailed description of the research journey followed as well as the theoretical underpinning and methodological reasoning behind it.

1.3.1 Literature Review

This thesis follows a design-oriented approach that seeks to create artifacts in order to solve real world problems. This recognises that a practice-oriented approach in social sciences is needed to address problems that require solutions with immediate and transferable value in order to support practice. The text that follows presents the underlying theory that underpins these concepts.

Research methodologies in the social sciences evolved over the past three decades. The traditional approach was concerned with theory-oriented research aimed at knowledge with no transferable practical orientation. At some point a new, more practice-oriented approach emerged that mainly focused on improvements of existing reality. This approach gradually changed into research aimed at creating new artifacts in order to solve construction or inventive problems and this is called design-oriented research (Verschuren & Hartog 2005, Fällman 2004).

Educational research followed the same path. Educational research and educational practice should ideally go hand in hand but in reality that was not always the case. This disconnection or lack of relevance between the two has been the reason that led to the development of alternative approaches to research. The orientation of these approaches leans more towards addressing and solving real problems in educational practice rather than answering research questions. An example that clearly highlights the difference between the two perspectives is given in McKenney & Reeves (2018).

The argument in this text is that instead of doing more studies comparing whether one or the other method is better in a certain context, it would bring more value to actually undertake research aimed at developing an optimal solution for the problem in that context. In the area of learning sciences, the belief that context is important changed the perspective under which research paradigms are being used. Research that simply examines learning processes as isolated variables within laboratory settings is doomed to fail as it will inevitably lead to an incomplete understanding of their relevance in real life.

Design Research A notable example of an approach with a problem-solving orientation is the design research approach. Design research has a clear orientation to support practice. The intention is typically to support improvement of systems. It embraces methods that help us understand reality and develop support for that improvement (Chakrabarti 2010). It embraces all facets of design and combines it with social and technical sciences to create tools and methods to support the process.

Educational design research follows the exact same principles to resolve educational problems. It blends scientific investigation with practical and systematic development of solutions to educational problems. This approach encompasses the systematic study of designing, developing and evaluating educational interventions that may take the form of programs, learning processes, learning environments, teaching-learning materials and learning systems in general. Empirical studies must lead to usable and effective solutions and in order to achieve that these studies take place in real learning conditions and not under 'sterilised', controllable laboratory settings (McKenney & Reeves 2018). According to Plomp (2013) the systematic analysis, design and evaluation of educational interventions has a dual purpose. The obvious objective is to build design solutions for complex problems in educational practice. The not so obvious objective is to advance our knowledge about the particularities of these interventions and the surrounding processes of designing and developing them.

Developing solutions to practical and complex educational problems is an iterative process and the motive is to improve practice (McKenney & Reeves 2014). Scientific inquiry in this context may target multiple goals simultaneously. Prioritisations of the work relates to how the research work and the relative importance of goals evolve. The aim is always a tight and rigorous connection between the theoretical principles and practical educational innovation (Gravemeijer & Cobb 2006). Rigorous theoretical analyses of educational problems generate ideas for possible interventions. These ideas are then used by designers to implement educational systems and evaluate them. The journey from theoretical principles to educational interventions is not straightforward. Designers must do design research in order to ensure that the system rigorously implements the research principles derived from theory. The purpose of design research is not to evaluate theories but to discover ways to create new systems based on those theories. Then, it is not the theory that gets evaluated but the effectiveness of the system in practice.

Design Thinking Dorst (2011), Lor (2017) is a methodology that follows the same approach but it is more loose and flexible. In this approach nothing is fixed as everything may be challenged during a design cycle. Even the definition of the problem itself may be revisited and reviewed. Both problem and solution may be constructed and reconstructed multiple times throughout this journey. The whole process is characterised by uncertainty and the element of subjectivity is always present as the approach pursued reflects the researcher's perspective. In this context evaluation and assessment of the final product and its usefulness might be tricky as the target might be continuously changing and adapting to the complex nature of educational problems. The design outcomes and their credibility rely on the design narrative, its cohesion and how it is supported by the research undertaken. Design is not applied science.

Design thinking is aspirational in the sense that it seeks to change things for the better. The proposition that a solution carries some value remains an aspiration un-

tils that solution gets evaluated. The process is highly dynamic as it aims at finding novel and innovative ways of framing problems that evolve during the development of a solution. This unique idiosyncrasy makes this approach especially suitable for ill defined, complex problems. In practice, research projects following this path take a constructivist approach that leads to solving problems through making and that naturally implies collaborations between people with different specialties like designers, learning technologists, teachers, students and others.

1.3.2 Reasoning Behind the Research Project - Design Thinking

The traditional approach to research is to address a given question and try to set up the best possible study that can provide an answer. In some cases this is quite narrow because there are research projects, typically design projects, that face many questions and of varying degrees of uncertainty. The single study pattern may not be adequate in this case (Gravemeijer & Cobb 2006). A sensible alternative would be to identify the most important questions related to a design problem and plan the studies to address them. The idea is to start with brief, inexpensive studies that reveal the most promising approaches. As new questions arise or existing questions are given different weight it is sensible to gradually invest in more rigorous and expensive studies on the ones that seem more crucial for the discovery of the truth. As Dorst (2011) states, there are basic reasoning patterns used in problem solving that, in general, try to complete the unknown factors in the equation that follows:

WHAT + HOW leads to RESULT

When it comes to design research, the equation transforms into the following:

WHAT + HOW leads to VALUE

The core difference between the formal logic and the design logic is that instead of predicting results the researcher wants to create valuable things. In this thesis the starting

point is the belief that programming is the literacy of the future and the present. The intention is to see if some aspect of teaching programming can be improved so that we can help society cultivate more literate people. At that stage this value is an aspiration based on personal views and experiences but it will be confirmed as the research progresses. Regarding the reasoning pattern we only know the end value we want to achieve, that is to "make teaching introductory programming easier". So the equation takes the form:

$$\text{WHAT} + \text{HOW} \rightarrow \text{"make teaching introductory programming easier"}$$

1.3.3 The Project Step by Step

In the text that follows we describe the process through which we gradually give substance to the two unknown components of the equation.

Exploring Possibilities

The first step is to start with the leftmost component. We start with a literature review in the areas of teaching and learning programming in order to get more familiar with the area and identify elements that can help us formulate an initial idea about the WHAT. The dominant finding in this research is that learning programming is not natural for everyone and requires support. Consequently teaching and supporting learning in this subject is expensive and thus limited due to limited resources. That strengthens our initial belief about the value. Another important finding is that teaching and learning programming in exploratory settings seems more natural and offers more opportunities for learning but also carries a heavier cost in terms of support. Thus, the problem becomes even more intense if learning takes place in an exploratory setting. A possible viable solution to this may be to complement human support with automated support through a learning system. Current learning systems that offer opportunities for exploratory learning lack automated support and that is an in-

dication that the WHAT in the reasoning pattern might be a system that facilitates exploratory learning in programming and offers automated support and adaptability.

At that stage we have research findings from the literature and some idea about WHAT might give a viable solution to the issues we identified. We need a first hand experience of the reality of those problems so that we can see for ourselves the issues and understand better the area of interest. We need to understand processes, flaws and weaknesses in the currently used teaching approaches, understand how learners approach learning and solve their problems and identify opportunities for improvement and innovation. The intention is to verify what we already know and put it in a real world educational context so that we can have a more concrete understanding of the situation. The approach we follow for that part is to do an ethnographic research (Mills & Morton 2013) by becoming part of the process ourselves. This part of the research was conducted in computer laboratories where students were undertaking programming assignments and practical training. We were involved in the process as tutors and in this capacity we provided help, we did observations and non-structured interviews.

This part gave us insight about the actual problems the students experience when they first start learning programming. We realised that laboratories become quite busy during these early stages of learning and students require a significant amount of support. Quite often the level of support may not be adequate or prioritised properly and students feel intimidated and excluded as they are not able to progress. We identified a set of common initial misconceptions that most students experience and that helped us realise the distinction between support that is independent of any task and support that is task specific. We verified both the need for exploratory and free learning and the need to provide as much automated support as possible. The fact that, according to the literature, there is no system that provides Task Independent (TI) support specifically, lead us to the next step which is the development of a system that offers free exploration and provides TI support on demand. The objective for this component is to

learn from the process about the technical feasibility, to understand the actual requirements for the development of such a system and identify opportunities for innovation. A final objective is to verify that there is indeed a need for this type of support and that learners are willing to use it. The approach for this last objective is to test the prototype with real students (Henson & Knezek 1991, Wong 1993).

The outcome of this process is a system called FLIP Learning that fulfills all the design objectives set. Throughout this process we developed a more concrete understanding of what is needed to develop such a system and we realised the technical challenges for componentization, reusability and interoperability. A significant development was the design of a generic architectural framework (refer to layered architecture in 3.6) that is suitable for web-based automated tutors in programming. Another useful outcome was the opportunities that arose for making this process less expensive through reusability of existing components. The next step is to conduct a usability testing (Barnum 2020) to verify the need for TI support with real users.

FLIP was used effectively by University students in two sessions with very encouraging results. Students were able to tackle the tasks given to them with minimal human support despite the fact that they had no previous experience in the language used. Student activities were recorded and analysed and the results indicate that TI support was used and learners were comfortable and confident to consult an automated tutor for their problems.

The experiences and findings acquired throughout this phase strengthen our belief about what is needed to make teaching of programming easier. The WHAT is clearly a system that allows learning programming through exploration and provides TI and TD automated support. The equation takes now the following form:

“ELE with automated support” + HOW → “make teaching programming easier”

Completing the Working Principles

At that stage we have established the WHAT component and we also have findings that give some substance to the HOW component. These are the layered architecture, the need for web-based design, componentization, reusability, interoperability and ultimately the need to find ways to make tutors in an easy and cost-effective way. The next step is to discover the rest of the HOW component.

In this phase we focus on the TD side of automated support. We need to know what other systems exist that offer automated support and what teaching and learning approaches they use. We need to know what systems support authoring of automated tutors, in what context and how usable they are. Finally, we need to identify limitations, deficiencies and weaknesses of current approaches so that we can recognise opportunities for improvement and innovation. For this part we did a literature review on all these areas of interest.

The outcome of this review was that there are a lot of learning systems that offer opportunities for exploratory learning yet these systems lack automated support and adaptability. On the other hand there are Intelligent Tutoring Systems (ITSs) that utilise very sophisticated technologies to provide support and adaptability but they offer limited opportunities for exploration and the learning process typically takes place in a quite controllable manner.

On the authoring side, tools for the generation of automated support are typically oriented towards guided learning. These systems are platform-specific technologies and offer domain-specific solutions. In general, the entry threshold for end users is quite high and development of automated support is quite expensive. What gives us motivation to move on at that stage is that there is very little work on authoring automated support for exploratory learning and there is no authoring tool that can develop domain-independent support for any learning environment and diverse technologies. Furthermore, there are technologies and approaches that can be give us a good starting

point for what we need to develop.

At that point we need a fresh look at the design considerations for building an authoring tool. The next step is to conduct a requirements elicitation workshop (Millard et al. 1998) and use participatory design methods to elicit the challenges in developing support for exploratory learning environments. The information we collected from this workshop helped us form an idea of the development requirements both at a technical and a functionality level.

Based on the low level technical requirements derived from this step we decided to give some time for a technical spike to address a very important obstacle in this process which is integration and interoperability with diverse technologies. The outcome of this spike was the development of a technique that can be used to overcome web component heterogeneity and achieve seamless integration and interoperability between any web component and its host environment. A prototype (Henson & Knezek 1991, Wong 1993) was implemented and named Web Integration and Interoperability layer (WILL). The technique enables seamless integration and unrestricted two-way communication between web components and their environment with minimal technical overhead. This output gives an answer to two problems: reusability of any web component that carries some educational value and reusability of web-based authoring tools with those environments. This gives us a more concrete picture of the HOW component.

Following the findings derived from the requirements elicitation workshop in combination with suitable techniques, technologies and architectures found in the literature, a prototype, named AuthELO was designed and developed.

The tool addresses successfully integration and interoperability issues and is able to perform dynamic and continuous testing of automated feedback in a sandbox with minimal administrative overhead. The next step is to evaluate it (Nieveen & Folmer 2013) and see whether it makes development of automated tutors for exploratory learning easier and less expensive.

At this stage we make the desired value more specific to focus more on the contribution of this project and the reasoning pattern takes the following form:

“ELE with automated support” + “HOW” → “make authoring of automated support for teaching programming easier”

Evaluation of Authelo

Evaluation of authELO took place in two steps. The first evaluation was a workshop involving the three learning designers that contributed to the initial design of authELO. The evaluation revealed aspects of the process that helped us revisit and refine the design of the tool. The evaluation also revealed that authELO can reduce substantially the time and development effort required regardless of learning environment and task. This was the first confirmation that authELO complements successfully the HOW component. This evaluation led to a re-design iteration which was subsequently evaluated with a quasi experiment - single group study (Privitera & Delzell 2019) that took place in the context of a real project. The aim of the experiment was again to evaluate the tool in terms of speed and ease of use. This time two learning designers were involved in the process and they did an integration and development exercise with and without authELO. The designers shared estimates for both iterations and the results clearly confirm the findings obtained from the first evaluation. The fact that we have similar results and positive user satisfaction strengthens our belief that there is value in this work.

The second evaluation was conducted with a focus group (O’leary 2017) of five top software engineers senior to lead level working in the software industry. The aim of this session was to obtain time estimations from industry experts about the development of the basic infrastructure for a system like authELO and the key phases in the authoring process identified in the first evaluation. The overall estimations confirmed the time gains achieved in the previous steps and the qualitative evaluation gave top

ratings to all the distinctive features of authELO. At that point we have a solid confirmation that the HOW component is complete.

Addressing new Requirements

The qualitative evaluation revealed a new requirement which is the need to have a more high-level language specialised in authoring feedback. That led to another iteration to revisit the design (Nieveen & Folmer 2013). In order to address this requirement we organised a requirements elicitation workshop with two categories of participants: ICT teachers with some programming skills and EdTech students with no programming skills. The ICT teachers were asked to develop support for a number of activities using authELO and the group of EdTech students were asked to do the same at a much higher level. The outputs from the first cohort were analysed and a common pattern for the data acquisition part was identified. This gave the requirements for the development of the high-level language. The outputs from the students confirmed that there is indeed a need for a block-based language for less skilled people.

The next step is to implement a software library to provide high level language constructs for the data acquisition and analysis part. Based on the requirements obtained in the workshop the library was implemented and integrated with authELO. This was an easy and cost-effective improvement to lower the entry threshold and give more value to the HOW component.

The final step of this research is an attempt to address the requirement of developing a high-level language specialised in authoring feedback. Instead of doing that, we designed and developed a tool that can be used for the specification of new languages executable in web browsers. The tool is developed and called *Lingua Franca Transformer* (LFT). It can be used to facilitate the authoring process of a new or an existing language and generates transpilers from that language to JavaScript. The transpilers allow real-time execution of any language in the browser with no server-side

dependencies.

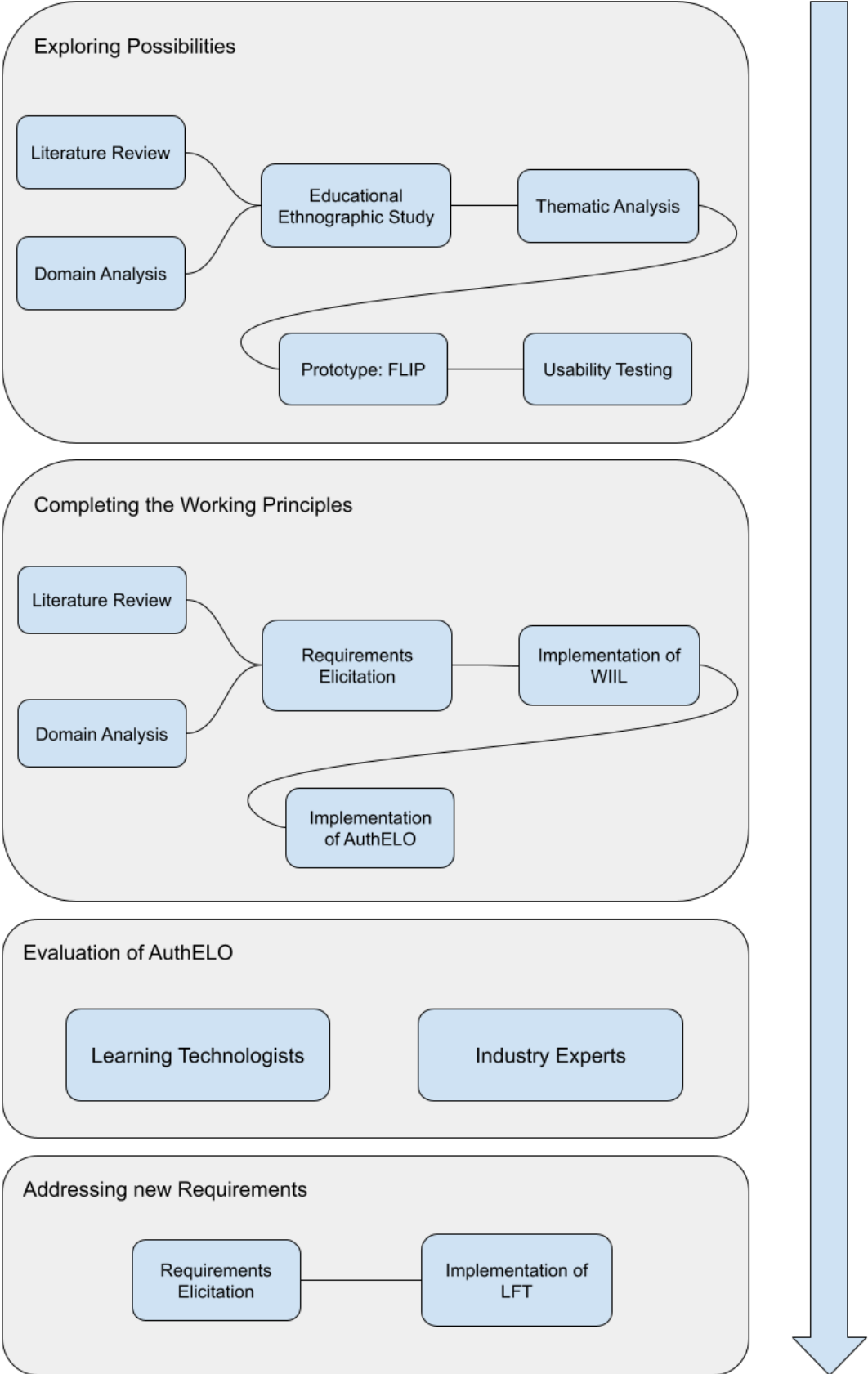
The benefit of this last development is two-fold: the ability to speak any language in the browser context increases reusability of web components. People can use the language of preference to manipulate widgets. In particular, learning designers can teach any programming language with any learning environment. In addition we give the ability to skilled designers to develop a high-level specialised language of their preference for authoring feedback and thus increase the reusability of authELO.

The final form of the reasoning patterns is the following:

"ELE with automated support" + "authELO" → "make authoring of automated support for teaching programming easier"

AuthELO is the final outcome for the HOW component as it gives us the means to achieve the aspired value. Alongside the final product, a series of methods, processes, tools, techniques that embrace the working principle are given as artifacts that provide added value. These artifacts can either be re-used as individual values to complement other projects or help other researchers and practitioners as a roadmap that shows proven and effective ways to develop similar projects.

Figure 1.1: The project step by step



1.4 Thesis Outline

The remainder of this thesis is organised as follows: Chapter 2 presents research projects related to the subjects this thesis is concerned with and gave inspiration to the ideas proposed. It also discusses issues identified with current approaches and opportunities for contributions.

Chapter 3, 4, 5 and 6 discuss the phases of this project as we gradually progress along the components of the reasoning pattern presented in 1.3.3. In chapter 3 there is a detailed presentation of the process through which we identify what needs to be accomplished so that we can achieve the desired value. In this part we show how we build up a better understanding of the areas of interest and how that supports our decisions in the process. We also show how we build up practical knowledge and we identify ways of how this objective can be accomplished.

In chapter 4 we focus more on how we can develop the entity that will give us the desired value. In this part we complement what we already know from previous steps with new knowledge and more technical artifacts.

In chapter 5 we focus on the evaluation of the developments presented in chapter 4. The new requirements that emerge from this evaluation are addressed in chapter 6. Chapter 7 presents a fully fledged learning platform that shows how all the components developed in previous steps can be combined together to achieve the desired value. Chapter 8 presents and discusses the contributions of this work having categorised them in components that facilitate reuse, components that simplify authoring of automated support and other miscellaneous components that provide additional value.

Chapter 9 discusses what we envisage as possible strands this work can follow after this thesis is submitted. Appendix A gives a diagrammatic presentation of a few sample rules designed for task-independent support in programming. Appendix B discusses the literature review strategy used throughout this project. Appendix C

shows the observation sheet used in the educational ethnographic study presented in 3.2 and finally appendix D shows the material used for the usability test presented in 3.7.

2

Related Work

2.1 Educational Programming Environments

There is a category of systems that have been designed to be used specifically for teaching introductory programming to high school or early University students. These systems, typically offer highly interactive environments with visual interfaces and allow the development of small scale projects, usually graphical applications. Students work in a design window, the equivalent to an editor, with objects that can be dragged, dropped and linked together to form interactions. The systems that fall under this category follow:

2.1.1 Turtle Graphics in LOGO

Turtle Graphics in LOGO (Solomon & Papert 1976, Papert 1980) is a system designed to teach basic algorithmic thinking to primary school children. The actor in this learning environment is a turtle that lives on a display screen. Its initial state is to stand in the middle of the screen with its nose pointing upwards (north). The user can communicate with the turtle using the LOGO language. LOGO is a very primitive and simple to learn and use language. The turtle can be instructed to move around the screen and as she moves draws shapes. Of course the system is much more than a drawing device since the turtle can also exhibit behaviour and respond to stimuli. The idea is to let the student explore strategies on how to manipulate the turtle and once they are successful to teach the system on how to repeat the task. Previous tasks can be combined with new ones to draw more complex shapes and thus teach notions of componentization and code reusability. This system has been very successful since there have been multiple replicas since its initial inception using different interfaces and exploiting new technologies as they became available. This system is probably one of the most heavily used and therefore thoroughly evaluated with students.

2.1.2 Karel

Karel (Pattis 1981) is a learning environment designed to teach early programming through robots that live in a virtual world (microworld). The virtual world is a graphical area that looks like a 2D matrix. Columns and rows are used to designate streets and walls. Robots inhabit this world and can move in all four directions as long as there is no obstacle (wall) in front of them. Robots can also pick up, carry, and drop beepers. Beepers are placed in intersections of streets and multiple robots may exist in the world. The initial idea behind this design was to teach basic programming without the need to spend time learning the syntax. Students can control robots using a very small set of simple commands and familiarise themselves with the logic of programming without spending significant time to fiddle with the particularities of a language. Since its inception the system evolved and took many forms. The first system used a Pascal-like language. Another version named Karel++ was based on a language similar to C++ (Bergin et al. 1997b). Later implementations were designed around the object-oriented paradigm and the underlying language used was Java (Becker 2001). Karel proved to be a very good fit for the OO paradigm as it made teaching of the fundamental OOP principles possible, very early in the process. That was the main problem with other approaches. OO principles had to be introduced very late in the learning process and there was little time left for the students to master the topics. No adaptability or automated assistance is supported in the system. Karel has been evaluated and found helpful in learning programming.

2.1.3 Toontalk

ToonTalk (Kahn 1996) is a general purpose programming environment that resembles a video game which is 'materialised' as a virtual world (microworld). The user can generate code using a programming by example (PBE) approach and code is animated. In this world, computational abstractions are mapped into concrete metaphors.

For example, a city is a computation whereas a robot is something that can be trained - by the user - to execute a task. The user can develop, run, debug and modify code by controlling a programmer persona in the virtual world. The design goal of ToonTalk is to eliminate the need to learn a text-based programming language to express things and provide a self-taught programming environment, thus, significantly reduce the cognitive load required to start programming and lower the entry threshold. This is based on the premise that people, mostly children, can learn on their own how to build complex Lego constructions or master video games that require exploration and problem solving abilities in complex fictional worlds. The design of ToonTalk is heavily influenced by analysis of video games and Lego systems (Malone 1980, Provenzo Jr 1991). Despite the fact that ToonTalk is easy to learn it is very generic and powerful in terms of expressiveness as it allows the user to build a wide range of applications like games, physical robots with motors and sensors as well as conventional programming examples like sorting algorithms. ToonTalk has been evaluated and found very helpful in learning programming especially with pre-school and school children.

2.1.4 Alice

Alice (Cooper et al. 2000) is a visual development environment for interactive 3D applications designed to assist students learn algorithmic thinking and introductory programming. Students can easily develop highly interactive 3D environments (virtual worlds) and use simple scripts to control objects' appearance and behavior. There is a clear OO flavor in the design of Alice and that makes it particularly suitable for teaching OO principles and programming. The design is based on the premise that some students find it difficult to visualise what actually takes place when the computer executes code. Therefore, students may be able to progress much easier if there is an alternative method to conceptualise that part much easier. The basic idea is to overcome this problem by using animations to show program execution instead of us-

ing code tracing or other complex visualisations. The objective is to provide a system that is as inclusive as possible and can accommodate the needs of students at every part of the spectrum. Alice was initially built on top of Python but latest incarnations used Java as the underlying technology. Alice has been evaluated and found useful in learning programming in the sense that students were very comfortable debugging and correcting programs.

2.1.5 BlueJ

BlueJ (Kölling & Rosenberg 1996, Van Haaster & Hagan 2004) is an integrated development environment specifically developed for teaching object-oriented programming. It is designed to overcome known problems that hinder the learning process in OO programming courses. The main argument is that the environment the students work with should itself reflect the paradigm of the language. If students have to work with different abstractions then their work is distracted and they spend time doing things not relevant to what they need to learn. BlueJ offers a graphical environment that simplifies interaction with the language significantly. The alternative is to either use the compiler directly or to use a sophisticated IDE designed for professional development. In both cases there is a significant overhead that increases unnecessarily the cognitive load required to deal with learning tasks. Students work in a main design window that shows a Unified Modelling Language (UML) class diagram that visualises the application structure. The underlying language used is Java. Students can interact directly with classes and objects using the respective icons in the graph. They can get access to the underlying source code and modify the text and they can compile individual components. They can also execute code on components like create instances of objects and run methods on them. All of that is immediately reflected in the visual environment they work in. That means that students get immediate feedback about what is going on in their program. This forces students face problems immediately as they can visually

inspect the results of their actions and all that is happening in a purely OO manner. BlueJ does not provide any adaptability or any automated support to students. It does not maintain learner profiles and knows nothing about the subject being taught. It has been evaluated and the results showed that students found it easy to use and quite helpful in learning OO programming.

2.1.6 Greenfoot

Greenfoot (Henriksen & Kölling 2004, Kölling & Henriksen 2005) is a visual programming environment designed to teach OO programming to novices. The interface consists of a main window that shows the scene which is the virtual world to be built. On the right there is another window that shows the class diagram that reflects the state of the virtual world. The learner can work with visual components directly to drag and drop, create the world and add interactivity. Gradually, once the user is comfortable with the concepts they can start interacting with the source code through the class diagram (like in BlueJ). To an extent the system looks like a combination of BlueJ and Alice. It comes with all the features and advantages of BlueJ but reduces significantly the entry level and the cognitive load by introducing a 3D microworld that can be manipulated without code. Basic OO concepts can be articulated and presented much easier and logic in programs can be followed without unnecessary details and cognitive overload. The underlying language is Java. Greenfoot has been evaluated and found very useful in learning programming.

2.1.7 SALESPOINT

SALESPOINT (Zschaler et al. 2014) is a java-based framework designed to support students learn software development in a holistic way. The basic premise in this project is that it is not enough for students to develop small scale isolated academic examples of algorithmic solutions to problems. In typical real-life projects software engineers

have to deal with a multitude of problems that go far beyond code writing. They have to understand and apply fundamental approaches to systematic software design and development as well as collaborate efficiently with co-workers and communicate their work with stakeholders. For this to work in academic settings, especially when there is little time and resources combined with large student cohorts, there needs to be a framework that streamlines the process and provides the foundation for the students to work on. In SALESPOINT students are able to plan the work, organise their teamwork, establish communication channels, do systems analysis, design, prototype, implement, test, maintain and document components. The core components cover application control, data management and GUI functionality. They are generic and allow learners to extend them and develop functionality specific to the business scenario dictated by their project design. The framework has been used for many years in University settings and despite the fact that it involved a significant overhead (about 30% of the total effort for the project) students found it very useful. Results showed that the system met its design objectives and students gained valuable experiences by participating in simulations of real-world development contexts.

2.1.8 Malt+

Malt+ (Kynigos & Latsi 2007) is an exploratory learning environment designed to teach introductory programming to secondary school students. The concept and the interface is the same as in turtle graphics with LOGO. The difference is that the student can manipulate the actor in a 3D space and create 3D dynamic geometry models. The scene can be switched between 2D and 3D and there is also an automation that displays a variation tool with sliders if there is variable identified in the code. There is also a camera available that can be used to change the perspective. The tool has been evaluated and found quite helpful in learning programming.

2.1.9 Scratch

Scratch (Maloney et al. 2008) is a visual programming environment designed to teach introductory programming to school students (primary to high school). The environment is supported by a block-based programming language designed to facilitate media manipulation. The design is essentially based on the ideas of LOGO and manipulating graphical objects in a scene. The difference is that the LOGO language is replaced by a drag and drop block-based language and therefore the user does not even have to learn LOGO instructions. This reduces further the entry threshold as it dramatically simplifies initial engagement. Emphasis is given on media manipulation and support of programming activities that young learners find interesting in order to stimulate engagement. Animated stories, games and interactive presentations are typical examples of Scratch projects. The visual setup is similar to Turtle Graphics and Malt+. There is a main window that shows the stage. The stage is inhabited by movable sprites that can be used to draw shapes. Each sprite is associated with their own set of images, sounds, variables and scripts. Programming constructs are presented as blocks in a palette that is always visible. The user can assemble stacks of blocks by dragging them from the palette to the scripting pane. Blocks that match in terms of syntax snap together when they are close enough. Stacks and blocks can be executed with a simple mouse click. Feedback is immediate in the stage. Code tracing is not easy thought. Scratch is probably the predominant learning environment for programming in schools today. It is being supported by a huge community of teachers, students, learning designers, researchers and programmers. There is unlimited material developed that can readily be used in the classroom. It is probably the most thoroughly and intensively evaluated tool under school conditions.

2.1.10 Discussion

The above systems are representative examples of the attempts that have been made to ease the burden of learning programming and make it an enjoyable and rewarding experience. Programming can indeed become more approachable and accessible by people using approaches like ways to streamline the development / learning process, provide suitable abstractions, metaphors, visualisations to assist in understanding the concepts and to minimise or even eliminate the need to use a proper language. All of these systems offer a lot of opportunities for exploratory learning and they cover a wide range of learning aspects like the event-driven and OO paradigms but they lack completely intelligent support and the ability to adapt to individual learning needs.

2.2 Exploratory Learning Systems

2.2.1 Exploratory Learning

Exploratory learning can be thought of as the opposite of guided learning. In exploratory learning, the learner discovers knowledge through experimentation, investigation and exploration in pursuit of a real or artificial task, whereas in guided learning the learner is given precisely sequenced training materials that have to be followed in a systematic way. Exploratory learning and its potential value were first introduced in the early 80's (Carroll 1982). The origin of exploratory learning was not in the area of education but it first began to reveal itself in the industry when interactive computing started to enter the business workplace (Rieman 1996). In many cases people were given computers and were asked to operate them with minimal training and therefore they had to improvise and devise their own learning strategies to discover knowledge in an exploratory manner. Carroll et al. (1985) analysed the learnability of office applications in the workplace in situations where users had no prior training or coaching during the process and the research revealed that most users were willing and able to learn through exploration. The process was not flawless as there were cases where users made major and unrecoverable errors, even with the aid of some basic instructions from manuals and tutorials (Carroll & Mazur 1986), but in general the process proved to be effective. Research also showed that there are individual differences in exploratory aggressiveness (Carroll 1990, 1987, Neal 1986).

An interesting research development that followed in this area was the effect of minimalism in the nature and the length of instructions in exploratory learning. The basic premise behind minimalism was that people that want to learn something are curious, impatient and mentally active as they typically want to challenge themselves and exercise their problem-solving abilities right away. They don't want to go through very structured and lengthy instructions that impede their urge to discover things. Therefore, instructions should be as brief as possible, relevant, provide support for

real problems, and where possible permit non-sequential reading. Carroll (1990) implemented this approach with great success as users managed to learn how to use fairly complex environments quite effectively and they also expressed their preference for it over other methods.

Another development that sparked researchers' interest in this area was the success of computer games (Malone 1982, Carroll 1982, Shneiderman 1993). There are a number of distinguishing characteristics that make computer games quite different from other software applications (Rieman 1996). Typically, there is no training required or expected in order to play a computer game. Playing the game should be intrinsically motivating and the user should be able to start using the software with no previous training or even knowledge of what it is or how it looks like. Typically, a computer game starts with a very easy to use environment that offers only the basic functionality, and as the game progresses new features are revealed incrementally. That helps users start and continue using the software as they become more knowledgeable, skilled and confident with it. Users learn how to play the game by playing it and normally they are quite enthusiastic about going through this process. Clearly, what takes place during this process is exploratory learning. The above findings suggest that "exploratory learning" can be an effective and attractive strategy for learning.

2.2.2 Exploratory Learning Environments

Exploratory Learning Environments (ELEs) are software that is specifically designed and developed for educational purposes. The difference between an ELE and an ordinary learning environment is that the former is open-ended and allows learners to work in a constructivist way. Learners discover knowledge by building constructions that depict scientific models (Amershi & Conati 2006, Chen 1995, Cocea et al. 2008b). They operate on these constructions, examine or change their properties and analyse the data that derives from these processes. Learners are actively engaged in the dis-

covery of knowledge through this process. These environments offer learners a lot of freedom to move around, explore and play, rather than being constrained or very directed in a highly controllable process. In ELEs learners are expected to work in a particular way. They are encouraged to investigate and explore a broad set of possibilities when looking for something. They are encouraged to construct and explore models by varying their parameters and observe the effects of these operations on those models (Cocea et al. 2008b). In terms of learning outcomes, ELEs have been shown to offer significant benefits to the learning process (van Joolingen & Zacharia 2009, Noss & Hoyles 1996). Since, exploratory settings provide for a rich educational environment for students (Amir & Gal 2013), modern pedagogical software tends to be more open-ended and flexible, allowing students to approach problems through exploration and trial-and-error. ELEs are by definition student-centric and that is perceived as an approach that requires less direct teacher involvement in the learning process. As a result of that, ELEs are generally used in big classes where teachers have a difficulty to monitor students and provide assistance when needed (Gal et al. 2008). For the same reason ELEs are becoming increasingly prevalent in developing countries where access to teachers and other educational resources is limited (Pawar et al. 2006).

ELEs are particularly useful for ill-defined domains (Lynch et al. 2006), where problems are not well structured and there are no clear boundaries between correct and incorrect approaches to solve the task. Some problems in these domains may have several valid solutions but none of them can be safely considered better than the others (Cocea & Magoulas 2010). As mentioned above, ELEs are built on the principles of discovery learning, which emphasises on opportunities to learn through free exploration and discovery rather than guided tutoring and that is aligned with the principles of the constructivism paradigm for teaching and learning. This approach has proved to be beneficial for learners in terms of acquiring deep conceptual and structural knowledge. However, discovery learning without guidance and support appears to be less effective than step-by-step guiding in traditional learning environments (Kirschner

et al. 2006). Provision of support in this context requires a lot of preparation, skill and intensive monitoring of the learning process. Guidance in ELEs requires monitoring and understanding of learner behaviours and the associated knowledge construction that is taking place (Morales Gamboa & Gamboa 2000). The nature of problems students have to tackle in ELEs combined with the fact that there are no clear boundaries between correct and incorrect approaches makes the task of providing support an excessively difficult, time consuming and resource-intensive process (Gutierrez-Santos, Mavrikis, Magoulas et al. 2012). It is a big challenge for teachers to keep track of student progress and assess their performance in such a context (Amir & Gal 2013). Research in the learning sciences suggests that freedom of exploration without a proper degree of support can be problematic due to distraction and loss of focus (Mayer 2004, Kirschner et al. 2006, Klahr & Nigam 2004). Furthermore, lack of support or guidance in an ELE can actually hinder learning (Kirschner et al. 2006).

ELEs are rich educational environments that offer a lot of learning opportunities to students but learning and teaching in those environments can also be very challenging. Teachers have a limited capacity to support students in those environments and that introduces a clear need for computer-based support (Gutierrez-Santos, Cocea & Magoulas 2010, Amir & Gal 2013). Considering all the above, it comes as no surprise that most of the effort in the community has been focused on the development of systems that offer student and teacher support for those environments. These efforts soon revealed another facet of the same problem.

ELEs' open nature makes it difficult to design a system that supports the user, as the operations that take place in them are mostly unstructured and the possible combinations of those operations quite broad (Cocea et al. 2008a). According to Murray (1999), ITS design, development and evaluation is, in general, a challenging and costly process anyway. The nature of student interactions in ELEs, and the constructivist intentions behind their design, make this even more difficult and costly in those systems (Gutierrez-Santos, Cocea & Magoulas 2010). Therefore, we have four distinct direc-

tions of research in this scientific area: systems that support teachers in ELEs, systems that support students in ELEs, design approaches for such systems and fully functional ELEs.

Teacher Support Systems

As mentioned in the previous section, there is a clear need for computer-based support for ELE users. ELEs tend to be used with large cohorts of students and students tend to work in unpredictable ways with ELEs. Teachers need support in identifying and recognising student activities and coordinate or facilitate learning accordingly. The techniques presented in Amir & Gal (2013) can be used to recognise student activities in ELEs. Once the activities are recognised, they can also be presented to teachers through visualisations. The approach is based on a plan recognition algorithm that uses a recursive grammar which takes into account repetition and interleaving of activities. The algorithm was evaluated empirically on an ELE teaching chemistry with good results. The system was able to correctly infer student plans when the appropriate grammar was available. The system also provides two methods to visualise student activities for teacher inspection. The first one visualises student inferred plans, and the other one visualises student interactions over a timeline.

Another project that focuses on tools designed to support teachers is presented in Pearce-Lazard et al. (2010). Possible student trajectories in ELEs are potentially infinite. In this context it is quite difficult to establish whether a student is moving towards a solution to a problem or how close they are to a discovery of something that carries some learning value. To resolve this issue there were landmarks placed in the system so that tools can provide information to teachers regarding student trajectories in relation to those landmarks. The purpose of those tools was to reveal the level of engagement on task dependent and task independent activities in MiGen. Tracking student trajectories would then enable real-time and retrospective interventions in the

learning process. Further work that focuses again on teacher assistance tools in MiGen is given in Gutierrez-Santos, Geraniou, Pearce-Lazard & Poulouvassilis (2012). This text presents the design and development of teacher assistance tools for the eXpresser microworld. The eXpresser is a component that resides in the core of the MiGen system and this is where students can construct models. The aim of this work was to find ways to identify common student misconceptions and situations where students may be in difficulty or may be disengaged from the given task. If this information is available to teachers then students may be able to receive more personalised support on how to reflect on their constructions and interpret the feedback given by the system. Teachers may be able to prepare their interventions in a more informed way and encourage students to work towards specific goals and communicate and share their constructions with others.

Another part of the same project that falls in the category teacher support tools is given in Cocea & Magoulas (2010). An interesting problem for teachers in ELEs is how to form groups of learners for collaborative learning sessions. This work presents a computational model for group formation specifically designed for open-ended exploration in ELEs. The technique is based on modelling the various strategies learners adopt to solve the same task. The underlying theoretical basis is underpinned by Group Technology (GT) techniques. The system uses learner strategies and the similarity among them as criteria to recommend homogeneous group formations that match the given pedagogy considerations. The formation of heterogeneous groups is also possible. The proposed mechanism is tested in eXpresser.

Student Support Systems

As shown in previous sections it is very difficult to exploit the learning potential of ELEs without adequate support for students. The level of support required is typically higher in ELEs as these systems are by definition student-centered and highly

complex due to lack of structure. For the same reasons, it is difficult to generate and provision automated intelligent support in those environments. It is also difficult to formulate a common strategy to develop support mechanisms that are universal for this type of systems. That, in turn, makes it difficult to perform fair comparative analyses between systems that employ different approaches. An attempt to provide some common ground on which different techniques for intelligent support can be assessed is given in Cocea et al. (2008a). This work is focused on the formulation of a modelling strategy for activities that take place in an ELE that teaches mathematical generalisation, called ShapeBuilder. The proposal is based on scenarios that cover different aspects of problems that take place during the learning process as well as the different possible methods of the required support. A case-based reasoning formulation is also given for the representation of learner behaviour during model construction.

The openness of ELEs imply that tasks in those environments can be approached in many different ways. However, learners are expected to be able to address certain things that characterise those tasks or have an awareness of them. The criterion to identify what the learners need help with may be their current level of understanding or more importantly the actions they perform as they are working on a task. Using solely these actions to understand the need and provide support may not have the same effectiveness for all learners due to differences in personal characteristics. Cocea & Magoulas (2009a) proposes a context-dependent personalised feedback prioritisation mechanism for ELEs to cover exactly this need. According to this approach, context can bring valuable information and help us provide more appropriate and effective personalised support. Context refers to different learning modes as individual or collaborative learning and the different stages within tasks. The proposed technique is based on the Analytic Hierarchy Process (Saaty 1990) which is a popular method in Multicriteria Decision-Making (Zopounidis & Doumpos 2002). Following this work there is an hybrid approach presented in Cocea & Magoulas (2009b) that is based on Case-Based Reasoning (CBR) and Multicriteria Decision Making (MDM) components

used for learner modelling and feedback generation in ELEs. This is a conceptual model that can be used to represent learner constructions and identify strategies learners follow in ELEs. These findings can then be used to prioritise different types of feedback. The Case-Based Reasoning component is used to diagnose what students are doing in the environment and the Multicriteria Decision Making component is used for learner modelling and feedback generation.

Successful exploratory learning is also related with certain meta-cognitive skills that students may or may not possess according to Bunt et al. (2004). Skills like systematic exploration, hypothesis generation and hypothesis testing are considered important in this process. One particular meta-cognitive skill that plays a crucial role for effective exploratory learning is self-explanation and this is where this work is focused on. Self-explanation means the ability to spontaneously explain to oneself things related to the activity that takes place in terms of the underlying domain knowledge. The ELE used in this project is called ACE and teaches mathematical functions. Support is based on a learner model that incorporates meta-cognitive skills to track and evaluate student understanding and exploratory behaviour. This project focuses particularly on how the model is expanded in order to track self-explanation behaviour of learners and improve the effectiveness of student exploration. Another system that focuses on the provision of automated adaptive support in ELEs is given in Amershi & Conati (2006). In this case machine (unsupervised) learning techniques are used to automatically recognise learner behaviours and identify patterns in those behaviours. These patterns can then be used to identify groups of learners that have distinguishing interaction patterns and similar learning improvements. This work then discusses how these findings can facilitate the provision of adaptive support in ELEs. Another system that focuses on adaptive support in ELEs is given in Bernardini & Conati (2010). In this work we have a learner model that is generated using data mining techniques and in particular Class Association Rule mining and the respective classifier. The idea is to automatically generate learner models that can then be used to facilitate adaptive

support in ELEs. The approach was applied on an ELE that teaches AI algorithms.

Systems Design

From what has been presented so far it seems that due to the nature of interaction in ELEs, it is difficult to design, develop and evaluate them. MiGen, a ELE presented in previous sections, is an intelligent, exploratory environment intended to support learning of mathematical generalisation. Pearce & Poulouvassilis (2009) presents the conceptual and architectural design of MiGen. The description gives a detailed technical explanation of a working proof-of-concept prototype of the architecture. This explains the motivation behind using the particular technologies and approaches chosen to implement the intended functionality in the context of this project.

Developing support for an ELE is not merely a matter of technical expertise. The system is complex, there are no generic architectural frameworks to streamline the process, the task is highly interdisciplinary and there are many different concerns by different people and from different viewpoints that need to be addressed. Scientists from different areas have to collaborate and in general there is a lot of interaction between technical and non-technical members of the research team. The nature of interaction expected in ELEs makes it difficult to design, develop and evaluate automated support that reflects student needs. An attempt to simplify this process is given in Gutierrez-Santos, Mavrikis & Magoulas (2010). This work proposes a model that addresses the above issues through a conceptual separation of concerns. This separation allows compartmentalisation of concerns and easier communication between technical and non-technical members of the development team. Another advantage of this layered approach is that it facilitates early evaluation of the system as it allows for both formative and summative evaluation of the components to be developed. Another project that focuses on the design and validation of intelligent exploratory environments is given in Mavrikis et al. (2013). This work is based on the premise that ELEs require ap-

appropriate support to lead to meaningful learning outcomes and that means that there should be a mechanism to operationalise relevant pedagogical strategies through appropriate computer-based support. Design of intelligent support components needs to be aligned with the theoretical principles behind exploratory learning. It also needs to be informed of what kind of interaction is conducive to learning so that it can lead to components that offer meaningful support. Development of intelligent support for ELEs is a long, complex and resource-intensive process. That means that there should be a framework in place that facilitates evaluation and identification of design and implementation flaws throughout the project and this was another objective of the work presented in Mavrikis et al. (2013).

ELEs

In this section we present some notable examples of ELEs that show what has been accomplished in this area. An early example of an ELE that teaches Lisp is given in Schmalhofer, Kühn, Charron & Messamer (1990), Schmalhofer, Kühn, Messamer & Charron (1990). This system was designed to allow exploration of the effectiveness of different learning approaches allowing receptive, exploratory learning and combinations of the two. The system was used in an experimental study to compare the effectiveness of three given learning conditions. In all three cases learning takes place in a basic exploration environment. The variable changed by the three conditions is the presence of tutoring. The first condition is to operate without a tutor. The second is to operate with a selective tutor and the last one is to operate with a constant tutor. The outcome of this study revealed that the selective tutor approach was the most effective of the three. Students working under this condition managed to acquire the intended knowledge and solve the given tasks in less time than the other cohorts. During the test they solved the criterion test tasks, while solving an equal number of programming tasks correctly. What this result shows is that combining the advantages of learning by

exploration and guidance that is available on demand seems to be the most appropriate tutoring strategy. Students learn better in an exploratory manner when there is on demand support that can help them overcome obstacles and misconceptions.

Another ELE, also presented in previous sections is eXpresser (Pearce et al. 2008). eXpresser is a microworld that lives in the core of the MiGen system and teaches mathematical generalisation. eXpresser is the component where learners can build and analyse general patterns through models. This system is equipped with sophisticated intelligent support components to assist both teachers and learners throughout the learning process. More details for both aspects are given in previous sections.

Another platform that is more generic, in the sense that it can host any exploratory learning task-based environment is Annie, (Thomas & Young 2011). Annie is a domain-independent platform that can host exploratory learning environments. An evaluation study that took place in this project, shows that using the platform had a strong, positive, and statistically significant impact on increasing the number of learning tasks completed by the students and reducing the time needed to complete them. The results did not show an impact in learning gains on domain knowledge.

2.2.3 Discussion

Evidently, learning in an exploratory context offers a lot of opportunities but it's not a panacea. The approach alone cannot overcome the inherent difficulty of people to learn programming. It needs to be supported with the right type and amount of help so that learners can exploit the full potential of this freedom. Teachers also need support in order to adequately guide students through this process. In both cases automated support is essential and can significantly improve the effectiveness of the process but design and development of intelligent and adaptive support for that type of learning is a complex, long and very resource-intensive process. The process requires a lot of domain knowledge combined with deep, low-level technical expertise in order to both

conceptualise and implement adequate solutions. Furthermore, solutions tend to be domain-specific and with limited reusability in other systems.

2.3 Tutoring Systems for Programming

It has been established that programming is considered hard by students (Bellaby et al. 2003, Virtanen et al. 2005, Lahtinen & Ahoniemi 2005). There have been numerous attempts to address this problem and identify ways to help students learn the concepts and develop the required skills. In the field of Technology-enhanced learning (TEL) there has been a lot of scientific interest in this area since the late 70's. A survey of tools that can be used to assist learning of programming is given by Deek & McHugh (1998). The categorisation proposed in this survey follows:

- Programming environments: Tools intended to allow users to familiarise themselves and experiment with specific languages and their features. Typically they are given as fully fledged programming environments equipped with editors, compilers, tracers and debuggers so that users can explore the features of programming languages and construct, compile, test and debug programs.
- Debugging aids: As the name suggests, these are tools limited to testing as well as observing program execution. Typically, these tools are intended to assist in detecting and correcting errors. Common components that belong to this category are code watchers, tracers, flags, and visualization and animation utilities.
- Intelligent tutoring systems: These are more sophisticated systems specifically designed to support training. Their aim is to provide automated and adaptable support to students that learn programming. Typically, they can interact with students and guide the learning process with instructions suitable to individual learner needs.
- Intelligent programming environments: These are programming environments enhanced with some features of intelligent tutoring systems such as adaptive instruction, automated support and assessment etc.

A presentation of systems that belong to the above categories follows. Some systems may belong to more than one category.

2.3.1 Programming Environments

Typical systems that belong to this category are equipped with code editors that support automatic syntax highlighting for different languages, syntax checking and verification as well as code formatting that follows language-specific conventions. Other features may include visual programming interfaces, dynamic compilation/ interpretation of code and dynamic visualisation of outputs. Some programming environments are designed to facilitate learning as well as development. In those systems usually there is tutorials about algorithmic thinking/logic, programming paradigms and concepts, language specifications and the respective semantics.

Pict

Pict (Glinert & Tanimoto 1984) is a visual programming environment for the Pascal language. It was designed to assist with algorithm design and program development in a way that is more natural for the human brain and not restricted to one-dimensional text-based representations. Programming in Pict is done with a joystick. Text input via a keyboard is not possible. Every element (atom) needed in program construction is represented by an icon and the user can select atoms and arrange them in a flow chart. The user can use the graphical representation of the program to observe the flow of execution. The system can identify syntax errors but there is no automated support beyond that. Pict was evaluated and found useful for novice students but limited and confusing for people that had previous knowledge of programming.

PECAN

PECAN (Reiss 1985) is a language-neutral system (suite of components) that provides

users with assistance in program development, algorithm understanding and execution monitoring of data structures. It has a layered architecture featuring basic, mid and higher-level modules. At the basic level there is the user interface where program construction/parsing/compilation takes place. At this level the user can use predefined templates and amend/augment with their own code. The mid-level module provides support for data manipulation and monitoring as well as program representation in abstract syntax trees. The higher level module utilises the mid-level services to provide various program views like semantic views, abstract syntax trees, flow diagrams, execution views and graphical representations of data structures.

SCHEMACODE

SCHEMACODE (Robillard 1986) is a system designed to promote better solution planning and code design. Development is done in pseudo code using the SPC language (Schematic Pseudo Code). The system can then translate SPC and automatically generate source code in other languages like Fortran and Pascal. The resulting code is self-documented using SPC specifications as comments. The design objectives were to provide a system that offers systematic and consistent documentation in the source code and enables better understanding of flow control that ultimately leads to better use of the language and problem solving. An obvious disadvantage of this system is that users need to learn SPC to implement things. Furthermore, SPC may not have the same level of abstraction as the target languages. This may be a limiting factor depending on what needs to be implemented.

DSP

The DSP system (Olsen et al. 1988) is specifically designed to support teaching of introductory programming. The system is a visual programming integrated development environment that supports code management, maintenance and reusability. The user interface provides access to code templates through a high-level visual language

intended to eliminate syntactical mistakes. There is emphasis given in modular development and working on different aspects of a problem / parts of the solution. The system can generate code in various languages like Ada and Pascal, execute the solutions and visualise memory maps to show data structures and variables. The user is allowed to access the underlying code, but in general this feature is neglected as emphasis is on visual handling, manipulation and understanding.

Amethyst

Amethyst (Myers et al. 1988) is a suite of software components designed to teach introductory programming. Its primary design objective is to teach programming through visualisations. Emphasis is given on the conceptualisation of data. There is visual representations of data structures as well as the operations on them. There is also different visual perspectives of code like outline, tree decomposition, and linear. Data visualisation is highly customisable allowing users to define ways to display complex data structures like linked lists and stacks.

University of Washington Illustrating Compiler

The University of Washington Illustrating Compiler (UWPI) (Henry et al. 1990) is a system designed to help students understand basic programming concepts through program visualisation. Code is written in a cut-down version of Pascal. The system can perform code analysis and maintain a semantic-rich internal representation with inferences of abstract data types, value ranges and idioms of code patterns. This representation is translated in a layout plan which is then transformed into a visualisation of the program. Mapping between the underlying code and the intermediary representation is done using a knowledge base.

BACCII

BACCII's (Calloni & Bagert 1994) is another system developed to teach introductory

programming by abstracting syntax details thought visual components. Students can develop programs in the form of flow diagrams and code is then automatically generated in various languages like Pascal and Basic. There is no support for other higher level tasks like problem solving. The system has been evaluated and has shown positive positive impact in learning.

ASA

ASA (Guimarães et al. 1994) is a development environment designed to teach algorithmic logic at an introductory level. It is equipped with a visual programming environment where students can develop code in the form of flow diagrams and generate code in a pseudocode language and other languages like Pascal. The system, also offers guided instruction in the form of tutorials with animations of relevant concepts and algorithms. ASA was evaluated and proved to be too instructive since students could develop correct algorithms without understanding their purpose. As a consequence of that there was lack of control over their learning process since there was limited information about their strengths and weaknesses.

POLKA

POLKA (Kraemer & Stasko 1998) is system designed to teach programming through the development of graphics and animations. Since graphics development is inherently difficult, high-level abstractions are used to reduce the level of expertise needed and make the development of animations simpler.

SUPPORT

SUPPORT (Zelkowitz et al. 1989) is a development environment intended to be used for programming in a cut-down version of Pascal. Code is written in a syntax-directed editor with function keys and menu buttons that correspond to statements and procedures. Text input that corresponds to expressions is processed by an internal parser.

The code is guaranteed to be syntactically correct because the system maintains and checks against an internal syntax tree and does not allow incorrect inputs. If the input is incorrect the system invokes automatically a component that helps the user to make the necessary modifications that will preserve the validity of the syntax tree. Other features include automatic logging that simplifies program output and testing, program tracing and debugging.

STRUEDI

STRUEDI (Köhne & Weber 1987) is a system designed specifically to teach programming to novice students. Coding is done in Lisp and students can work with a sophisticated syntax-directed editor to select predefined language constructs from a menu and synthesise programs. The template constructs are placed in the workspace and contain empty slots that need to be completed by the students. The main design objective of STRUEDI is to help students understand better the syntax and the underlying semantics. According to the evaluation the system seems to satisfy those objectives.

Example-Based Programming System

The Example-based Programming System (EBPS) (Neal 1989) is a system designed to assist novice programmers learn coding through examples. The idea is based on the premise that programmers tend to reuse previously written code and integrate it with their programs rather than develop everything from scratch. Typically they are looking for code examples that exhibit the same characteristics or behaviour with the function they are trying to develop. Sources of information may be textbooks, source code from previous projects and nowadays the WWW. The core of this approach is reusability of code, which if done properly, offers a lot of advantages like less risk (code is already tested), economy (less development effort) and a lot of learning opportunities (code understanding, integration). The system comprises a syntax-directed editor and an example library. The editor is supported by an example window. Example programs

given in the latter can be copied into the editor and processed there. The system has been evaluated and found useful in teaching introductory programming.

Software Design Laboratory

Software Design Laboratory (SODA) (Hohmann et al. 1992) is a system designed to support novice students learn Pascal. The focus of the system is to support specifically the software design process. The students are directed to follow a very specific design pattern in a highly controllable environment. First, they are asked to decompose the problem using references of existing program solution techniques. Then, they have to state explicit goals, specify plans, and then assemble the pieces into a working solution. Finally, they have to verify the solution using a debugging module. The system is supposed to help students modularise a problem properly and integrate the modules into a working solution. An obvious disadvantage is that it is very inflexible and does not give opportunities for exploration. It is a purely guided process that must be executed in a sequential manner even though some parts of the process may require an iterative approach or could be executed in parallel. The system has not been fully evaluated but there are indications that it affects positively student performance in design.

MEMO-II

MEMO-II (Forcheri & Molfino 1994) is a system designed to help novice students develop problem solving skills rather than learn specific languages or engineer programs. This is contrary to what other tutoring systems of its time were doing. The system was designed to help inexperienced users build correct problem-solving abstractions that may, then, be implemented in different programming paradigms and languages. Specifically, the system offers help in problem representation, specification validation, implementation, and program execution. MEMO-II comprises three major components: The first component is used to build solution specifications. That is done using an editor and a verifier. The students are asked to express a problem specifica-

tion in a specialised, syntactically complex language that consists of a set of predefined operators. The second component is a reasoner that is used to validate the specification base given in the editor. The third component is used to translate a valid specification into program code. Although the solution proposed seems to offer student activities that are essential for understanding and solving problems, the system may not be suitable for novice students. The obvious disadvantage is that students have to learn a complex language to perform the problem representation task and that seems not to be consistent with the original design objectives of the system.

2.3.2 Debugging Aids

These are tools intended to be used for testing programs, observe program behavior during execution, detect and correct errors.

LAURA

LAURA's (Adam & Laurent 1980) is a tool designed to identify semantic errors in programs and suggest possible corrections for them. Typically, debugging aids are tools used during the program construction process to identify syntactic problems and correct them. This tool is designed to be used after the program construction is completed and there is no interaction with the user during development. Therefore, the focus is not on syntactic errors but rather on semantic problems in syntactically correct programs. The tool requires information about what the program is intended to achieve in order to debug it properly and for that a translation of the code is performed into a language-independent form where syntax variations are removed and programs can be checked against models. The tool is using a knowledge base of program models as a foundation for that. New programs are translated into graphs by a debugger and then compared against the solution representations in the KB. The outcome of this process is diagnostics about possible semantic inconsistencies in the code. The tool has been

evaluated and found quite effective in identifying correctness or errors regardless of structure variations in programs.

The Debugging Assistant

The Debugging Assistant (DA) (Laubsch & Eisenstadt 1981) is a tool developed to help students learn introductory programming in the language SOLO. The code is first translated into a language-independent diagram that contains information about control structures, data flows and a description of the overall effect of the program. This diagram effectively depicts an execution plan of the program. At that stage the system can detect what is called "irrational code". That is instances where there is unbound variables, code that is unreachable and code that seems not to be used for anything. The next step is to classify the diagram according to a library of standard plan diagrams. The diagrams are seen as sequences of code (patterns) that are manifestations of certain plan diagrams in the library. A plan diagram comes with an effect description. The next step is to evaluate the diagram and analyse all possible paths. The outcome of this is an effect description that reveals the changes in the data as a result of running the program. The final step is to check for mismatches between the effect description of the program and the one in the library. If there are mismatches then the system tries to figure out what is causing the problem in order to suggest corrections. Various aspects of the program are investigated like variable bindings, missing steps, correctness of conditions in branches and the respective effects. The system has not been evaluated.

GENIUS

The GENIUS prototype (McCalla & Murtagh 1985) was designed to help in introductory PL/C programming. The underlying idea is based on the premise that knowledge-based systems cannot provide adequate support and guidance to novice programmers since complete coverage of solutions requires too much intelligence and knowledge.

The system proposed as an alternative is one that appears to understand and know a lot about programming where in fact it doesn't. Its purpose is to get the students discover the problems themselves by keeping them engaged with the problem for as long as it takes to identify them. The system is based on what is called "ignorance-based reasoning". Although it can only identify syntax problems it pretends it knows about the logic aspects as well. The tool is equipped with a simple natural language understanding module that can interpret student answers to questions and a domain knowledge module that contains hints and advise. The students are asked to provide the error code as input and the tool first tries to determine if it is a syntax or logic error. If it is the latter it tries to engage the student with the problem by repeatedly asking questions and providing general advice. The system has been evaluated and found not able to provide adequate support to students. In general it was established that as the complexity of problems increased, the perceived usefulness of the system decreased.

VIPS

VIPS (Isoda et al. 1987, Shimomura & Isoda 1991) is a tool developed to assist in ADA programming. It is a visual debugger able to perform dynamic code execution, visualise program flow and display graphical representations of debugging data. A unique advantage of the system is that it allows the user to determine the graphical representation of data. The disadvantage is that the user is supposed to learn a specialised language called Figure Description Language (FDL) in order to do it. The tool has not been evaluated.

Lens

Lens (Mukherjea & Stasko 1994) is a tool designed to help with logical mistakes that derive from incorrect coding. The tool provides code visualisations as a series of animations. The advantage of using it is that the user does not have to learn a graphics paradigm or write additional code in some specialised language. As the code is

graphically executed the user is able to trace the execution flow and recognise logical problems easier. That is especially suitable for learning introductory programming. Lens is an algorithm animation tool that utilises a traditional source-level debugger. Its functionality is limited since it cannot deal with very complex programs and map algorithmic principles like recursion. The tool has not been evaluated.

2.3.3 Intelligent Tutoring Systems

These systems are much more sophisticated than the ones in the preceding categories. They typically present sets of problems to students and provide continuous support until the solutions are developed. The solutions are automatically assessed and the systems recognise individual learner needs and adjust the level of instruction. Adaptive instruction along with automated intelligent support are the two primary characteristics of this category. The cognitive objectives of those systems are also much higher. Depending on the orientation of the system indicative examples include development of cognitive models and mental representations of problem domains. In terms of architecture there is typically three general components in those systems:

- A knowledge base that formally represents the domain of knowledge on problems, solutions, and known errors. This is knowledge inserted into the system by experts and typically used to recognise errors, identify possible corrections, evaluate student progress and consider possible solutions.
- A learner model that is used to represent the student learning process along with the current level of understanding. This is used to dynamically determine the material that needs to be taught, the level of support and possibly the teaching approach that needs to be followed.
- A virtual tutor. This is typically a component responsible for the orchestration of the learning process. It utilises the two previous components to collect information about goals, students and their current understanding and makes decisions

on how to interact with them to facilitate the process and achieve the learning objectives.

The BASIC Instructional Program

The BASIC Instructional Program (BIP) (Beard & Barr 1976) is a system designed to facilitate the study of tutorial methods using computer-aided instruction. The focus of this study was on feasibility and effectiveness of those methods. BIP is an instructional course on the BASIC programming language. It is intended to be used as an informal introduction to problem solving and the target group is high school or college level students with no prior knowledge of programming. The core of the system is an enhanced BASIC interpreter that can assess correctness not only in terms of language but also in terms of the given task. It does that by comparing certain patterns in the code with model solutions. If assessment fails, the student is given a lesson on the missing skill. Lessons derive from a library of 100 programming tasks associated with certain skill sets. Teaching is done through hints and examples. Those hints and examples come from a different library that contains text-based help messages and problem-solving hints. The system selects dynamically what lesson to present next based on student ability which is measured through error counters and self-reported statements. The system has been evaluated and shown no positive results in terms of performance. It has shown, though, that people using it can achieve higher throughput as they encounter fewer difficulties in completing the tasks.

The LISP Tutor

The Goal-Restricted Environment for Tutoring and Educational Research on Programming (GREATERP), also known as the LISP tutor (Reiser et al. 1985), is a system designed to teach the LISP programming language. The system itself is built in LISP. The design is based on the premise that private tutoring is more effective than classroom training as students tend to learn the same amount of knowledge faster. Another major

design consideration is the fact that students learn better when immediate feedback is always available. As students work on the tasks, the system remains silent in the background for as long as there is nothing in the code that indicates a deviation for the expected solution. If there is a divergence detected, the system interrupts the process to help the student re-align themselves with the correct solution path. The system offers explanation of the problem and the correct solution using templates stored as LISP production rules. The system has been evaluated and found less effective than human tutors and more effective than self-learning.

PROUST

PROUST (Johnson & Soloway 1985), is a tutoring system for Pascal programs. The system initially breaks down the program and tries to determine what the code is intended to be used for. Known problem-solving algorithm models are used to assist in the goal decomposition task. Once that is done the system examines the code against its own stored solutions for the task. If there are significant differences that cannot be reconciled a series of transformation rules are invoked and the code is decomposed further. Individual functions are checked, error messages are displayed along with explanations and help on request is offered. PROUST has been evaluated and found quite accurate in detecting bugs. It fails, of course, when students introduce unusual errors or even use creative ways to solve a problem.

The ACT Programming Tutor

The ACT Programming Tutor (APT) (Corbett & Anderson 1993) is a system designed as a programming environment to help students complete short programming assignments in LISP and PROLOG. The system was built to test the validity of the ACT* theory of skill acquisition (Anderson, 1983). According to this theory programming knowledge can be modeled as a series of “if-then” production rules that form an “ideal student model.” The system is using a knowledge base to supports this and the user

interface is presented as a set of graphical windows. The students are given exercises through the main window. Another window is used to display hints - feedback. A different window is used to present a graph that shows the probability the student has learned the skills being presented in the exercise. The system has been evaluated and shown moderate success. An obvious problem is that as the complexity of the problems increases, so does the complexity of the rules.

2.3.4 Intelligent Programming Environments

Intelligent programming environments combine features of intelligent tutoring systems such as adaptive instruction, monitoring and assessment of students' progress, and feedback and advice with tools that are used in the program development process. In addition to the domain knowledge base, student model, and tutoring agent, intelligent programming environments provide access to traditional programming environments utilities such as syntax editors, compilers, and debuggers. A detailed explanation of each system follows.

Bridge

Bridge (Bonar & Cunningham 1988) is designed as a learning environment for the novice programmer. The main design objective was to bridge the gap between the syntactic approach to learning a language and the cognitive processes needed to solve a problem, hence the name. The students start with the specification of a solution to the given problem that is expressed in English terms. The system is equipped with a natural language understanding module to translate these specifications into a programming plan. Finally, the students, supported by the system, translate the programming plan into Pascal code. The obvious benefit of this system is that the students are encouraged to think of algorithms in English which is much less formal than a programming language and that helps them to easier conceive the logic and formu-

late their ideas. The system is lacking support for problem decomposition and code design. It also does not support functions or procedural abstractions. Correctness of student solutions is verified against model solutions stored in the system. In general the system is limited to support tutoring on very simple problems in nature. Bridge has been evaluated but evaluation was very limited.

Graphical Instruction in LISP

Graphical Instruction in LISP (GIL) (Reiser et al. 1989) is a system designed to teach simple programming in Lisp. As highlighted by its name, the system allows students to synthesise their solutions in a visual programming environment. GIL is fairly flexible in the sense that it allows students to develop solutions in both directions. One can start from the data and work forward to the solution or start from the solution and work backwards. Another distinguishing feature is that it can determine the way the student develops the solution and adapt the way support is provided. It can use its own reasoning to suggest the next steps to be taken by the student. GIL has not been evaluated adequately but results look promising.

Intelligent Tutor, Environment and Manual for Introductory Programming

Intelligent Tutor, Environment and Manual for Introductory Programming (ITEM/IP) (Brusilovsky 1992) is a system designed to teach introductory programming to high-school or beginning college students. The language used is called Turingal (Brusilovsky, 1991). The students are given a task and asked to work with a code editor to develop a solution. The system maintains the student history and is able to make decisions as to whether a task must be repeated or not. Work is assessed by comparing student solutions against model solutions stored in the system. Support and guidance is provided by demonstrating the incorrect behaviour detected in the student solution. The system has been evaluated with good results.

DISCOVER

DISCOVER (Ramadhan 1992, Ramadhan & du Boulay 1993) is system designed to teach introductory programming. The language used is a simple pseudo-code language specifically designed for DISCOVER. The system is equipped with two modules that allow the student to work with or without guidance. The only support provided in the non-guided component is a memory visualisation. The guided component is more structured as it offers a set of predefined problems and monitors progress towards the solution. The student is given the specification of a problem and then they must find a solution by combining programming concepts from a menu. Solutions are assessed by comparison with model solutions stored in the system. Partial analysis of code modules is also possible. The system has not been adequately evaluated.

Episodic Learning Model Programming Environment

Episodic Learning Model Programming Environment (ELM-PE) (Weber & Mollenberg 1994) is a system designed for teaching LISP. The design is based on the fact that novice programmers usually need explicit examples of what they have to develop. Normally, references to similar problems successfully addressed in the past helps significantly. According to Gentner et al. (1985) novices have a difficulty to recall analogies between old and new problems. That means that there is an obvious need for support in that part. ELM-PE is built on that premise. It teaches LISP by showing relevant examples to students. Students develop programs in a code editor that provides support to prevent syntax errors. Tutoring is done by a specialised module that provides code examples that may be copied and modified by the student. There is also visualisations to show the flow of program execution. The core of the system is a component that performs cognitive diagnosis of the given solution and categorises the solution as good, sub-optimal and buggy. Based on this diagnosis a hints, suggestions and relevant code example may be shown. This is based on a rule-based system. ELM-PE has not been adequately evaluated.

Capra

Capra (Verdejo et al. 1993) is a system designed to teach program design at an elementary level. The designers recognised the need to teach something more than the syntax and the semantics of a language. The design is based on a three-step workflow that leads to a programming solution. The first step is to recognise the problem abstraction. Then, the second step is to identify the relationship to a class of solutions, and finally the third step is to refine and produce a final answer. The final answer will have to be known by the system. Only pre-stored model answers in the system's knowledge base are accepted. The core of the system is the modules that present students with exercises and check problem-solving activities in order to check their comprehension. The obvious drawback is that comprehension is always checked against known solutions and there is no room for creativity.

INTELLITUTOR

INTELLITUTOR (Ueno 1994) is a system designed to teach intermediate programming. It is equipped with a sophisticated editor that is able to utilise built-in knowledge about the language syntax to help students avoid mistakes. Once the solution is written, another component is used to analyse the code and recognise students' intentions. This is supported by a knowledge base of alternative model solutions. If there are errors identified, the system offers relevant advice on how to fix them. The system is also adaptive and makes dynamic decisions on what knowledge to present to the student based on a learner model. This model is continuously updated with information that derives from user activity and indicates the perceived level of understanding. The user model does not take into account the user knowledge though. The system has not been adequately evaluated.

2.3.5 Discussion

Tools used as programming aids and fully-fledged programming environments are traditionally seen as enhancements of the development process that provide programmers with significant help. Using these tools, one can streamline the development workflow, simplify and shorten the length of the procedures involved and make the process more easily manageable. The tools presented above offer templates to help with the conceptual design and generate code and offer different perspectives and visualisations. The ease to identify mistakes and debug code or test the validity of your solutions is invaluable but that is normally the case for people that do professional programming and they just want to automate the process as much as possible. These people are more concerned with efficiency and throughput rather than learning. Contrary to that, novice programmers need to go through this process as slowly as possible so that they can learn to recognise problems and learn from them. Programming aids and sophisticated development environments that do all the work for them most likely hinder the learning process rather than facilitate it and that is the case especially during the first stages of learning. Furthermore, programming environments that offer too much functionality in terms of enhancements impose a quite high learning curve to the end user. It is very likely that all these features will initially intimidate the novice learner and make learning of programming even more exclusive than it already is. The real value of these aids is to utilise them to gain as much insight as possible for the actions that take place during learning and the semantics around them. This data can then be used as feedback so that systems can have better information about student state and understanding and provide suitable assistance. Programming aids of this era are quite limited in terms of freedom and they lack support for intelligence and adaptability. ITSs on the other hand offer excellent examples of intelligent support and to some extent adaptability but in general they offer limited opportunities for exploration and user interaction takes place in an environment that is highly controllable

and intrusive.

2.4 Automated Support Authoring Tools

Technology enhanced tutoring systems naturally aim to provide learning materials to students that are highly interactive and provide as much automated support as possible. The level of sophistication of those systems is positively correlated largely with the cost of development. This becomes very evident in traditional ITS's. According to Woolf & Cunningham (1987) it takes roughly 200 to 1000 hours of development time to create an hour of ITS instruction. This is one of the reasons that justify the very low level of adoption of ITS technologies in schools and online learning platforms. The complexity of those systems along with the difficulty to support them and provide robust and stable services to students are additional problems (Murray 1997). The cost of development is typically related to the fact that knowledge needs to be inserted manually into these systems. That presupposes input from expensive resources like domain experts, knowledge engineers and specialised system developers. In typical ITS's these people combine their efforts to create monolithic, tightly integrated solutions that offer limited configurability without low-level programmatic interventions. One solution to the problem is to logically separate authoring of automated support from the development of platforms. Decoupling the two and making the authoring tools more easily approachable by non-experts may be the key to enable people utilise these technologies more. There have been attempts in the past to logically separate the two and provide more flexible solutions that allow independent development of materials with various levels of support. A description of these systems follows:

2.4.1 SQL-Tutor

SQL-Tutor (Mitrovic 1997) is an ITS designed to teach University undergraduate students SQL. The goal of this system is to achieve effective one-to-one tuition with students following a guided discovery learning approach. That means that the process

is guided but students are given a certain degree of freedom to find the correct answer through discovery. Students are supposed to use the system in conjunction with traditional classes and some previous knowledge of database theory is assumed. The scenario is that students work on their own as much as possible and the system intervenes only when students are unable to complete the task and ask for help. The system is not equipped with a domain model but it is using rich and computationally tractable student models to guide instruction. A constraint-based approach is used to develop the user models in order to minimise the inherent complexity in them. These models combine both student knowledge and domain knowledge. The assumption behind this approach is that student understanding is not hidden in the student's actions but rather in the current state of the solution given. There is predefined databases and problems in the system. The student is asked to deal with them and any solution state is checked against sets of constraints. If a problem solution violates the constraints then the system is able to intervene and provide support. A distinguishing feature of the system is that knowledge (constraints) is not built-in and it can be inserted into the system or modified through a GUI. That gives the flexibility to add constraints at any time and with minimal development and administrative overhead. Getting the knowledge is still a costly process and requires domain expertise and knowledge engineering but at least there is no development involved.

2.4.2 ASPIRE

ASPIRE (Mitrovic et al. 2009) is a complete authoring and deployment environment for constraint-based ITSs. It is designed to simplify the development of AI components needed in SQL-Tutor, EER-Tutor and Normit (Mitrovic, 2012). Although, the development of constraint-based tutors is less expensive in comparison to other ITS methodologies, manual insertion of constraints is still a demanding task that requires

substantial effort, time and expertise. An initial approach to simplify the development of these components was to provide a web-based authoring interface called WETAS (Web-Enabled Tutor Authoring System). ASPIRE (Authoring Software Platform for Intelligent Resources in Education) is a more complete solution that provides both authoring and deployment capabilities. Authoring is significantly simplified since the system is able to generate the domain model automatically. ASPIRE consists of two main components: The ASPIRE-Author is the authoring environment that enables domain experts to easily develop constraint-based tutors by providing high-level description of the domain along with examples of problems and the respective solutions. The ASPIRE-Tutor is the platform that enables the deployment of the generated tutors. ASPIRE has been evaluated and shown that around 90% of the constraints needed can be generated automatically. If domain experts have programming skills it allows manual insertion and editing.

2.4.3 Diligent

Diligent (Angros Jr et al. 2002) is an authoring system for automated tutors in simulation-based learning environments. The knowledge is inserted into the system through observation. An expert demonstrates the skills to be taught and the system observes and learns. The system processes the recorded observations in order to derive meaning and better understanding of the role of each step in the procedure. Training outputs can be directly modified by the author before the training process is completed and the learnt procedure is finalised. The author can also add the language that needs to be used for the system to explain the procedure to students. The generated model can be reviewed and modified even after the completion of the training phase. The model is presented in a graphical form and it is directly editable by the author. The author can also test the model by asking the system to use it to teach the respective skill. Diligent

has been evaluated and has proved very valuable especially in complex, error-prone procedures. Its applicability is, of course, limited in simulation-based learning environments.

2.4.4 Disciple

Disciple (Tecuci & Keeling 1998, 1999, Tecuci et al. 1998) is a framework that can be used to develop automated tutors for any domain of knowledge. The framework provides a very specific set of constructs and workflow that must be followed in order to develop a tutor (learning agent). A fully trained tutor may be used to interact with students, help them to solve problems and guide them through the learning process. Domain-specific knowledge is inserted into the system by an expert using a variety of methods like inductive learning from examples, explanation-based learning, learning by analogy and learning by experimentation. Disciple gives a basic infrastructure of a tutor that is like a template. The template has a very specific structure and requires customisation before the actual tutor is developed. There are two interfaces that need to be implemented. One interface is needed so that the expert can insert knowledge into the system and another one is needed so that the system can communicate this knowledge to the student during teaching sessions. Every instance of a tutor can only be developed for training that is domain-specific and the domain knowledge is encapsulated within a problem solver component. That means that the above interfaces and the underlying functionality are highly domain-specific and hardly reusable. The extent and the sophistication of the problem solver depends on the scope and the purpose of the tutor being developed. Development of those components require domain expertise, knowledge engineering skills and specialised developers to implement the components and apis. Depending on the complexity of the knowledge domain and the respective semantic network the whole process can be very lengthy, expensive and

require considerable effort.

2.4.5 Demonstr8

Demonstr8 (Blessing 1997) is an authoring system that can be used for the development of automated model-tracing tutors for arithmetic domains. Knowledge is inserted into the system by experts in the form of production rules. Training is done using a PBE (Programming By Example) approach and production rules are inferred by the system. The system also provides the means to authors to further abstract the generated productions if needed. A central component of the tutor is the student interface. This is what gets created first and forms the basis for further development. This interface is composed of cells that correspond to working memory elements (WMEs). Not all WMEs have the same significance. The most significant ones are called higher-order WMEs. Knowledge may be embedded in cells and therefore directly depicted in the interface or exist in the background as knowledge functions. These functions operate as maps and are implemented as 2D arrays that store values for every permutation of its inputs. The author selects an example problem and demonstrates the solution as a procedure. The system then selects a suitable knowledge function for each step in an automated fashion. It goes through all the functions trying to locate the one that produces the same result. If there are many candidate functions, the author is asked to select one. Every action performed by the author generates a rule. Once the rule is produced it gets displayed and the author can review and modify it by selecting a more general or specific condition for it. Every rule must also be associated with a goal and the skills it covers. The author must also provide four messages with increasing level of detail for every rule in the system. Development of tutors using Demonstr8 requires a lot of expertise and it can be a very lengthy and expensive process. The system is domain-specific and cannot be used in more open-ended, non-procedural domains of

knowledge.

2.4.6 CTAT

CTAT, Cognitive Tutor Authoring Tools (Koedinger et al. 2004, Alevan, McLaren, Sewall & Koedinger 2006, Alevan, Sewall, McLaren & Koedinger 2006) is an authoring environment that can be used for the development and deployment of model-tracing tutors. The tool allows two types of tutors to be developed: Cognitive tutors and example-tracing tutors. Cognitive tutors are equipped with a model (production rules) and can trace student activity while solving problems. They can be relatively generic and new problems can be added to them with very little effort. Example-tracing tutors are more problem-specific as they can contain a fixed trace from one particular problem. Development starts by creating the student interface which entails stating the problem and demonstrating the problem-solving procedure. The actions in this procedure take the form of a behaviour graph that is visualised in a tree-like structure - see figure 2.1. Then, the graph is annotated by adding hints in correct links and error messages in incorrect links. Links must also be labeled with the skill associated with the corresponding problem-solving step. Labels can then be used to generate a skill matrix that shows the knowledge elements required to solve the particular problem. CTAT has been evaluated and found very efficient compared with manual development of tutors. The ratio of development time to instructional time with is 23:1 on average. An estimate of the respective ratio for manual development is 200:1 (Koedinger et al., 2004).

2.4.7 Automatic Rule Authoring System for CTAT

(Jarvis et al. 2004) This is an authoring environment that can be used to generate automatically Jess rules for CTAT. The idea is to utilise existing domain knowledge in conjunction with examples of problem-solving actions to generate rules automatically. Problem-solving actions are inserted through PBE as before. The author demonstrates the process and correct steps are used as positive examples, whereas incorrect ones are used as negative examples. The system has been evaluated with data from three domains and found quite effective. Development time was reasonable but in some cases the generated models proved to be overly general (Jarvis et al. 2004). The system may be effective but development can be quite expensive as it requires a lot of expertise in cognitive modelling and model tracing. Modelling must be especially detailed and complete since the rule generation engine depends fully on the list of skills given on the links. An incomplete skill set would result in an incomplete set of rules and that makes the job of the domain expert especially demanding.

2.4.8 SimStudent

SimStudent (Matsuda et al. 2007) is another system developed to help generating production rules within CTAT. The purpose in this case is to predict student performance. The author demonstrates the problem-solving process and for each step there are three inputs given to the system. The first is the interface component that is being modified and has the focus. The second is the actual value entered by the author to that component and the third is the actual skill being demonstrated. The system takes this input and generates a production rule. The approach has been evaluated and shown promising results in predicting correct behaviour but disappointing results in predicting errors.

2.4.9 GIFT

GIFT (Sottolare et al. 2012) is a modular CBTS (Computer-Based Tutoring System) framework designed to simplify the development of automated tutoring systems for military training and education. The motivation for this system is based on the premise that modular design combined with standardised technologies can enhance reusability, support authoring and optimization of CBTS strategies for learning, and lower the development cost and skillset required for users to adopt CBTS solutions. GIFT targets domain experts with little or no knowledge of computer programming or instructional design. It provides various tools that enable rapid development of expert models and other domain knowledge that generate fully-fledged ITSs.

2.4.10 The FRAME Approach

The Feedback Reasoning Analysis Model Events (FRAME) approach (Gutierrez-Santos, Mavrikis, Magoulas et al. 2012) is a divide-and-conquer strategy intended to facilitate development and testing of automated support for ELEs. The motivation behind this approach is that feedback integrated in ELEs is usually purely reactive (as opposed to supportive) as it does not take into account learner profiles as well as interaction history with the environment to provide adaptive and more personalised support. The provision of support at that level is a very challenging task as interaction is highly unstructured and it is difficult to distinguish "correct" from "wrong" approaches. "Wrong" approaches in this context might have a better impact in terms of learning outcome. Furthermore, expertise from very different disciplines is required to develop that type of support and this fragmentation typically hinders the development process due to difficulties in communication and collaboration. The FRAME approach deals with these issues by reducing the complexity of the development process. It first recognises the three most important questions related to support:

- What is the situation now? (evidence)

- Which aspect needs support? (reasoning)
- How should the support be presented for maximum efficacy? (presentation)

These questions relate to different aspects of the general problem and thus different concerns regarding the development of support. The separation and compartmentalisation of concerns enables a more clear definition of the scope related to each component and allows the definition of clear communication interfaces between them. That allows contributors to focus on the component of interest and facilitates better collaboration between them as well as testing and validation of resulting components. A conceptual data flow diagram that depicts the development process is figure 2.2.

The development flow moves towards the opposite direction of the data flow. The first step is to establish the type of support required at the front end and how this support should be communicated with the learner. The next step is to find out the concepts related to the type of support required and the corresponding rules that need to be evaluated in order to activate it. The final step is to identify or derive the evidence that can be used to evaluate the corresponding rules and generate the related feedback strategies.

2.4.11 Discussion

Authoring intelligent support, especially in exploratory settings, is a very complex and resource intensive process. The cost of development is quite high and that makes these solutions not attractive and feasible for implementers. Typical authoring environments are domain, platform or technology-specific solutions that offer outputs of limited applicability to a wider range of problems. Authoring requires low-level technical expertise and is not intended for the low-skilled teachers and learning technologists. Furthermore, solutions are typically focused on guided learning scenarios and there is no system that offers the versatility to develop support for diverse and heterogeneous learning components in exploratory settings.

Figure 2.1: The CTAT system

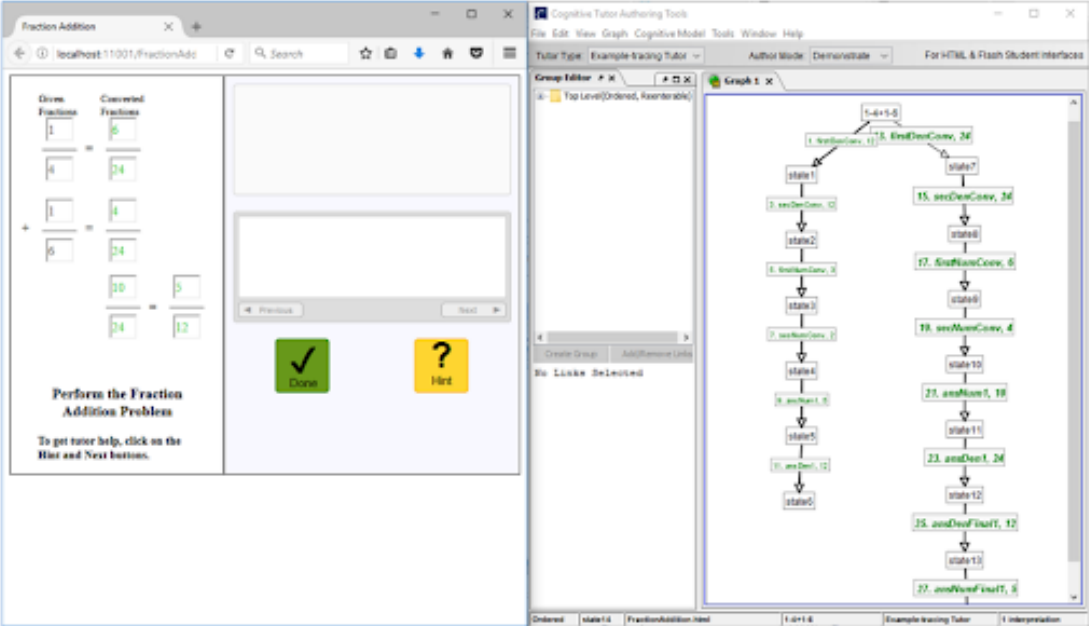
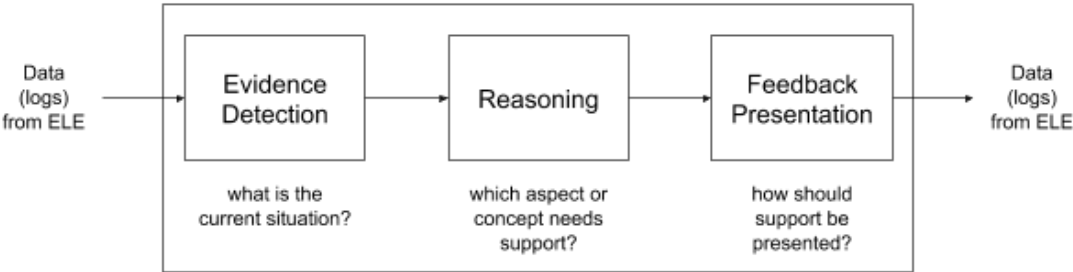


Figure 2.2: The FRAME Approach
Intelligent Support for Exploratory Learning Environments



2.5 Integration and Interoperability

In the early 90's, the advent of the Internet and the gradual advances of high-speed networks along with the increasing popularity of personal computers brought distributed computing to the forefront (Chung et al. 1997). People started to look for solutions that would give them the ability to reuse existing functionality potentially residing in disparate and heterogeneous systems that may operate in the same or remote networks. Different hardware, programming languages, operating systems and communication mechanisms were obvious obstacles in this process. Initially two frameworks emerged as standards to simplify network programming and distributed component-based software architecture. These were CORBA and DCOM. Throughout the years these technologies were superseded by new ones that came to remedy shortcomings and satisfy new business requirements. In this part we present the basic technologies emerged in this area.

CORBA

The Common Object Request Broker Architecture (CORBA) is a framework for distributed software component development designed collaboratively by a consortium of 700+ companies called the Object Management Group (OMG). This framework became a standard that was used to facilitate the interoperability of systems deployed on diverse platforms. The standard promises a system that abstracts hardware, operating systems, programming languages and programming paradigms. The programmer can work at a level that is neutral and provides methods that enable seamless integration and interoperability of local or remote components. The core of this architecture is a component that is called Object Request Broker (ORB). This can be thought of as a bus over which software components located locally or remotely can seamlessly communicate and interact (Vinoski 1997). CORBA itself uses an object-oriented model but that is not a mandatory requirement for systems that implement it. Pieces of function-

ality are exposed as abstract objects that have a specific interface with a set of methods. Objects are uniquely identified by object references. When a client component requires this functionality acquires a reference to the object that provides it through ORB. ORB handles all the processes required to locate the object, prepare it to receive the request, pass the request to it and return the reply to the client. CORBA has been criticized a lot and is currently not as popular as it used to be. Technical shortcomings was one of the reasons that affected acceptance. Technical complexity and specifically the complexity of defining APIs is a big hurdle. A notable example of this is the amount of code required for interface definitions in an object adapter. This is typically 200+ lines of code for something that could be expressed with no more than 30 lines. Problems with language mappings, the type system and insufficient features like the lack of support for versioning and security are issues that affected acceptance as well. Finally, another practical problem is the difficulty to communicate through firewalls. CORBA requires a port to be opened in firewalls for each service and network traffic is unencrypted. This may not be a problem for applications composed of components that reside within the same network but for internetwork communications this is in serious conflict with typical security policies.

DCOM

Distributed Component Object Model (DCOM) is another framework for distributed software component development. This is a proprietary technology designed by Microsoft. The model is an extension of another technology called COM (Component Object Model). COM is a model that enables a language-neutral definition and implementation of software components that can be reused across different applications, environments and machines. Internal implementation is encapsulated within the component and only a set of well-defined interfaces is exposed to provide different views of the object. DCOM is COM with an extra layer that takes care of remote procedure calls to support remote invocation and use of objects. As with CORBA, interfaces are

defined at a higher level using an Interface Definition Language (IDL). Client components acquire a reference (pointer) to one of the interfaces and invoke object methods through that pointer that may be executed in different processes on the same or different physical tiers. DCOM may be language neutral but it is not platform independent. It runs only on Windows operating systems and the supported hardware and obviously this is a serious limitation in terms of interoperability. It also suffers from the same practical problem as CORBA since it is difficult for components to communicate through firewalls.

SOAP

Simple Object Access Protocol (SOAP) is a messaging protocol specification for exchanging structured information via web services. The advent of the WWW along with the great success of XML and the decline of CORBA and DCOM lead Microsoft to design an XML-based messaging protocol that could be used over the web as an alternative. SOAP was the outcome of this effort and was submitted to W3C for standardization. The main design objectives were extensibility, neutrality and independence. Extensibility means an architecture that allows more features to be added without violating existing ones. Neutrality means network protocol neutral as SOAP can operate over HTTP, SMTP, TCP, UDP and JMS. Independence means that it can be used with any language, programming model, operating system and hardware. SOAP allows software components residing on disparate and heterogeneous systems to interoperate seamlessly. They communicate through messages expressed in XML. Message negotiation and transmission is typically done via HTTP or SMTP. HTTP and XML are like universal standards and as such they are available in any platform. This is a big advantage. Another advantage over DCOM is that messages can pass through firewalls with ease because the HTTP port is open by default to accommodate web services.

REST

Representational State Transfer (REST) is an architectural model that defines methods to combine existing standard web technologies to create web services. Web services that adhere to the model are called RESTful Web services (RWS). REST's big advantage is the fact that it doesn't define new technologies and frameworks but rather combines existing ones in certain ways to achieve communication and interoperability. REST is stateless and can only be used over HTTP with a limited set of standard requests (methods) allowed in services. Although, this might seem to be a limitation, it allows REST solutions to be simple, efficient and easily implementable. Communication and message passing is also faster since JSON is used to format messages instead of XML and JSON is more lightweight than XML. The payload for responses can be expressed in HTML, XML, JSON, or some other format. In general this is the prevalent technology today since it has superseded SOAP since it is simple, fast and more easily implementable.

GraphQL

GraphQL (Hartig & Pérez 2017) is a conceptual framework that can be used for the definition and development of web-based interfaces to databases. It consists of an open-source graph query and manipulation language and a runtime environment that can handle data operations. GraphQL was initially developed by Facebook but it is currently maintained by the GraphQL Foundation. The motivation behind this project was the need to resolve many of the shortcomings and inefficiencies of REST like the lack of flexibility and efficiency. A typical problem with REST is that data operations are exposed in the form of a set of web service endpoints that return fixed data structures. The assumption here is that it is very difficult to design an API in a way that is able to provide client components with the exact data needed and that often results in overfetching and underfetching of data. Overfetching occurs when the interface returns more information than it is actually required and underfetching occurs when less information is returned and additional requests must be made. GraphQL provides the

flexibility to client components to define with a high degree of specificity what data operations are required. All data operations can then be requested through a single endpoint. If multiple datasets are required, then all operations can be performed in one request. This gives flexibility and certainly increases efficiency but comes at a cost and that is limited caching and increased complexity. A requisite of this system is to have a properly defined data API using the GraphQL Schema Definition Language (SDL). It may not be worthwhile to use this framework for simple APIs.

oData

oData (Chappell 2011) is a protocol that specifies how REST can be used to define and build simple and standardised web-based interfaces to databases. It is a technology developed originally by Microsoft in 2007 and then standardised at OASIS (Organization for the Advancement of Structured Information Standards) in 2014. The principles are similar to GraphQL since the idea is to give the ability to client components to specify queries using URIs. Client components can consume, modify and publish data defined in data models through simple HTTP requests. It gives a lot more power to the client side than GraphQL though. In GraphQL the client component cannot ask for anything the server does not explicitly handle. Contrary to that an oData client is free to express any query, serialize it into a URI and execute it at the back end. This is seen as a potential security threat and creates obvious difficulties in traceability of code and optimisation.

gRPC

gRPC is an open-source high performance RPC (Remote Procedure Call) framework developed by Google. Its focus is performance and scalability as it allows bidirectional streaming over HTTP, flow control, multiplexing requests over a single connection, cancelation and timeouts. Services can be defined using the protocol buffers IDL (Interface Definition Language). Protocol buffers is a platform and language neutral

mechanism for serialising structured data. Interface definition, development and deployment is relatively fast and easy. gRPC is gaining popularity in the industry as it is currently being used by many major corporations worldwide.

OpenAjax Hub

OpenAjax hub is a lightweight Javascript library developed to facilitate the interoperability of web widgets within a web page. It is a technology defined and maintained by the OpenAjax Alliance. The hub operates like a security manager entity that follows the publish/subscribe paradigm. Individual web widgets that are integrated with the web page through Ajax can be isolated and operate in their own secure sandboxes. All the communication between these components passes through the hub which allows or denies publish or subscribe requests. The hub is designed around the concept of anonymous broadcasting. Producers and consumers are not aware of each other. Point-to-point messaging, cross-component property management and remote procedure calls are not supported. Since RPCs are not supported, asynchronous execution through callback functions is not supported either. The hub addresses the inherent security problems of integration of third party components in web pages but fails to provide an adequate solution for interoperability.

2.5.1 Technologies used in Learning Management Systems

Learning management systems (LMS) are platforms that host suites of tools to support online course creation, maintenance and delivery. They also support administrative tasks like student enrollment, cohort management, learning performance and reporting. They all have been designed to support extensibility in order to allow new functionality to be added so that the systems can align with new, emerging requirements and needs. The problem is, though, that architectural decisions, APIs, communication protocols and data formats are often proprietary and that is a serious limitation

in terms of extensibility. Initially, the LMSs developed in the early 90's were implemented as monolithic black boxes. Learning materials (courses) and other functionality could be integrated with platforms but the components were too tightly coupled with the host. Every integration was a special case that led to high cost and limited reusability. The first standards that emerged were about learning content representation. These were Dublin Core (Weibel et al. 1998), IMS Learning Resource Metadata and IEEE Learning Object Metadata (Barker 2005). Content creators could develop courses using those standards and integrate them with compliant LMSs. There was also a standard communication API developed for components that would like to send notifications to hosts regarding commencement and completion of learning activities. This was the AICC Computer Managed Instruction (AICC) standard. As LMSs started to mature and the need for more semantically rich integration was emerging new standards arose. These standards addressed issues like integrating more complex learning material than content like learning objects, whole courses and learner information. Examples of these new standards include Shareable Content Object Reference Model (SCORM) (Bohl et al. 2002), IMS Content Packaging (Wilson & Currier 2002) and IMS Learning Design (Leo et al. 2004). These standards enabled the integration of courses through a simple import/export mechanism. Another emerging standard at that period was IMS Learning Tools Interoperability (LTI) (Severance et al. 2010) that provided a technique for making tools hosted in different LMSs interoperable. In the late 90's LMSs started to follow the Service Oriented Architecture (SOA) paradigm and the systems became more modularised and flexible. That allowed a more clear separation between learning content, tools and learner information.

2.5.2 Epiphytic Integration Systems

This section presents a number of indicative systems that follow the epiphytic approach in integration. An epiphyte system is defined as a system that integrates with a

foreign system in a non-invasive manner (Paquette & Tchounikine 1999, Paquette et al. 1996, Richard et al. 2003). More specifically, there is no requirement for any changes in the target system in order to perform the integration. The target system may be completely unaware of the epiphytic application's existence.

SEPIA

The Sepia System (Ginon et al. 2014) is a set of tools that can be used to generate automated assistants to complement existing applications without the need for tight integration and interoperability. The aim is to have the ability to develop support for any native Windows or JVM application without accessing its APIs or recompiling the application to get direct access to its internal components. The term used to describe this type of integration is epiphytic. Thus, in this context, the automated assistants are called epiphytic applications or epi-applications whereas the applications they support are called target-applications. User interactions with the target applications are monitored by special components called epi-detectors. These are applications utilising accessibility libraries that can monitor events and log activity at any level. As evidence is collected rules are being evaluated and if certain conditions are satisfied the assistant performs the consequent actions to provide support. These rules are inserted into the system by domain experts through a special component called assistance designer using a specialised language called aLDEAS. The actual execution of the assistance operation is performed by another component called the generic assistance engine.

The Sepia system can only be executed as a native application and that carries all the inherent administrative overheads related to that approach. It is also unable to utilise domain knowledge that is embedded into the target system which may be necessary for more advanced assistance scenarios. The definition of rules requires a lot of expertise and working knowledge of a specialised language. Despite these shortcomings the system has been extensively used with various target applications includ-

ing learning environments and evaluations have shown that it has achieved its design goals which is the provision of assistance through a non-invasive loosely coupled integration with diverse applications.

Epiphyte Recommender System for Web Applications

This is a technique proposed to enhance websites with recommender systems that provide real-time support to users (Richard & Tchounikine 2004). The motivation behind this work is to promote a more effective use of the content presented in web pages and prevent cases where content is unused, underused or misused. The distinguishing feature of this approach is that the recommendation system can be deployed in a non-invasive manner as it does not require any change at all in the original web application. In fact the web application is completely unaware of its presence. The recommender system operates as a proxy. The user communicates with the proxy and the request is then being forwarded to the actual website. The proxy can operate in a different physical or logical tier. The proxy intercepts all the user requests and utilises a model of prototypical uses of the website to analyse the user trajectories and provide recommendations accordingly. Recommendations are given in a separate little window that is displayed along the main window of the webpage. There is no personalisation in these recommendations as no user models exist in this approach. Prototypical models are predefined in the system in the form of graphs. The system has been evaluated and results have shown there despite the fact that there is one additional tier through which web content must be transferred and processed there are no significant delays in the process. The system manages to provide intelligent assistance in a pure epiphytic way to web applications.

The iFrimousse Architecture

This is an architecture proposed to augment educational web portals like LMSs with services that can help educators monitor learner activity in real time (Carlier & Renault 2010). The architecture aims to do that in a non-invasive manner as the components that control logging and analysis of user trajectories operate as epiphytic applications. In order to achieve that an n-tier architecture is imposed in conjunction with a proxy so that clients cannot access the LMS directly. Since location based information is not available in the HTML content served by the LMS, the request is intercepted by the proxy and the content is processed through pattern matching and enhanced with special tags. These tags are then being used in subsequent requests to log user activity and extract location-based information. The data logs are processed and information about the current state of students becomes available in mobile devices through web services. This way tutors can have information about learner engagement in real time and intervene to remedy problems during the learning sessions.

Eclipse Student (In)Activity Detection Tool

This is a tool intended to augment the eclipse IDE and provide real-time monitoring of student progress (Karkalas & Gutierrez-Santos 2014a). The tool is designed to integrate with the IDE in a non-invasive manner as a plugin. After installation the plugin behaves as an integral part of the Eclipse platform and it gets automatically activated whenever the user starts interacting with the IDE. There is no visual indication in the interface that reveals the tool's operation other than the stream of messages that get displayed in the console tab. Once installed the tool can follow the roaming user profile and monitor user activity in different machines without interruptions. The tool consists of two components. The observer component is responsible for detecting user activity and generating the corresponding indicators. The user interface component is responsible for communicating the required information to tutors through native mo-

bile and web applications. Both components interoperate through web services that handle data logging and retrieval operations. The motivation behind this work is to provide real-time assistance to tutors and help them prioritise better the provision of help in computer laboratories. The tool has been tested in real laboratories for stability, fault tolerance and usability and found to have accomplished its design goals.

2.5.3 Discussion

Integration and especially interoperability has always been a difficult task that required technical ability and skills. Implementation is resource intensive and deployment entails a heavy workload in terms of administration and maintenance. Solutions of the past suffered from excessive complexity in interface definitions, problematic language mappings, insufficient type systems, lack of security (unencrypted network traffic), inability to pass through firewalls and access different networks, lack of extensibility without violating existing functionality and lack of platform independence.

Technologies have evolved a lot and modern solutions like GraphQL, oData and gRPC are now offering significant advancements in functionality, simplicity, efficiency, security and versatility. All of them can and should be used as references of good paradigms in software integration.

The prevalent standard in integration of learning components with platforms nowadays is IMS LTI. LTI is mature and robust but fails to accommodate situations where all its functionality is not required and introduces significant overheads. There is no lightweight version of it for cases where the actual component to be integrated is a simple front-end web widget with no dependencies or other functional considerations. This makes it an unattractive solution for simple components that are usually freely available on the web. OpenAjax used to be the standard for implementing interoperability of components in the context of a browser. Notable problems with it are the lack of point-to-point messaging, cross-component property management and remote

procedure calls. Asynchronous execution through callback functions constrain a lot, in an artificial way, the potential of combining different components together efficiently in a browser.

An integration and interoperability mechanism for educational content that is implemented as a simple front-end web widget should be able to accommodate the advancements in new integration technologies to allow existing functionality to be reused with no artificial constraints. Solutions should be simple and lightweight and able to travel across networks without restrictions and security concerns. Interface definitions should be small and functionality should be easily accessible and deployable in browser-based solutions to avoid unnecessary network traffic and roundtrips with back-end services. The component interfacing mechanism should be simple and standardised to allow easy integration but it should also be versatile enough to allow all the required functionality residing in diverse components to manifest itself and be exposed for external use. In most cases refactoring of third party components that are freely available on the web is not possible, easily implementable or even desirable. That means that epiphytic integration should be investigated for browser based solutions. Finally, solutions should be simple, efficient and easily implementable with minimal administrative overheads.

2.6 Synthesis of Related Work and Revised Research Objectives

In this research project we are touching on a lot of areas of knowledge covering learning theories, teaching and learning approaches, teaching and learning of programming, (educational) programming environments, intelligent tutoring systems, exploratory learning systems, authoring tools for automated support and integration - interoperability standards, techniques. In many cases there is no overlap or links between these areas. Throughout this project research needs directed us to touch on those areas and opportunities to create useful connections presented themselves. We started with a review on teaching and learning programming in order to understand the fundamental concepts in the area, realise difficulties and shortcomings and identify opportunities for meaningful contributions. We established that the value we want to achieve is to make teaching of introductory programming easier and one way of doing it is to develop a system that facilitates exploratory learning in programming and offers automated support and adaptability. A review on educational programming environments, specialised IDEs - debuggers, ITSs and ELEs gave us an in-depth understanding of the theoretical and technical background behind these systems and that helped formulating a more concrete idea about what we need to develop and how. We developed a prototype that teaches programming in an exploratory context and offers TI support. This experience gave us positive results, good insights and the encouragement to move on. The next component was to see how TD support can be given in an exploratory context. For this we did a review on authoring tools for the development of support and systems that offer it. We wanted to see what the state of the art is in those systems, identify limitations, deficiencies and weaknesses and recognise opportunities for improvement and innovation. In the attempt to identify requirements for the development of a new authoring system, the need to overcome a technical obstacle related to integration and interoperability became clear. During a technical spike to

address that problem we did a review on integration and interoperability standards and technologies to inform the technical underpinning of the solution. After a number of design iterations a prototype of an innovative authoring tool for the development of automated support was developed. In the text that follows we provide in detail all the links developed between different parts of the literature under each category when this is applicable to the case discussed.

Educational Programming Environments

Most of the systems reviewed under this category are microworlds employing constructivism as the teaching approach. A comparative analysis between them revealed interesting patterns and commonalities. One theme that seems to be repeating has the following characteristics:

- The environment comprises elements
- The elements may be created, deleted, modified
- The elements respond to events
- Different types of events may be related to different types of elements

We find these characteristics in logo, karel, toontalk, alice, greenfoot, malt+ and scratch. As these systems are typical examples of learning environments for programming that offer opportunities for exploration, these observations heavily influenced the design of authelo. One of the fundamental assumptions in authelo is that the learning environment being enhanced comprises different types of elements that may be associated with different types of events. This is reflected in the way authelo builds dynamically part of its user interface so that it corresponds to the particularities of the associated learning environment. Upon initialisation the learning environment is asked to provide details about the types of elements and types of events that may be found in its virtual universe and authelo utilises this information to dynamically create its interface. This way the author can use the same tool and the same concepts to define rules

for different environments. This is clearly related to the facilitation of reusability research objective.

Another common characteristic found in those environments is that most of them focus on reducing the complexities of the programming language used. Logo, for example, uses a relatively easy language with a small learning curve to lower the entry threshold and enable primary school children to engage with programming. The same pattern is seen in Malt+ and Scratch that uses a block-based language. Karel uses a small set of commands with very simple syntax for the same reason. Toontalk uses a self-taught programming environment to significantly reduce the cognitive load required to start programming and thus lower the entry threshold. The same concept of a self-taught visual programming language is also used by Greenfoot. The concept of reducing the cognitive load and thus reducing the entry threshold for prospective users is something that influenced a lot of our thinking throughout this research project. This is reflected in the design of FLIP and especially the design of authelo because the same idea can also be applicable to learning designers and authoring of automated support. In FLIP the cognitive load is reduced through adaptive automated support on common misconceptions. In authelo the cognitive load is reduced through the compartmentalisation of different concerns in the design process, the application of the example-tracing design pattern, the unification of the entire design process in a single logical tier and the ability to express authoring concepts using high-level language constructs. The intention to create different levels of expressing the same concepts in the context of authelo using LFT is following exactly the same reasoning. A high-level language specialised in authoring feedback aims to achieve exactly the same thing as logo in Logo microworld. This concept is related to the simplification of authoring research objective as stated in 2.1.

The fact that all the systems mentioned in the previous paragraphs follow more or less the same design pattern (elements, events) but use different languages to achieve the same objective reveals another interesting concept. The language can be thought

of as a component that is logically distinct and separate from the environment. That leads to an intriguing thought. What if we can mix and match languages and learning environments? People knowledgeable in one language would be able to use any system in this case and benefit from its comparative advantages. If the intention is to learn different languages using a system that is well established and of high quality, then it may be possible to reuse the same system with many different languages. The same argument also stands for newly defined languages especially designed to facilitate specific learning objectives. This notion of language neutrality is related to the facilitation of reusability research objective.

Another observation that was revealing is one of the main principles behind the design of BlueJ. This is the ascertainment that the learning environment should reflect the paradigm of the language being taught. This helps the learner better conceptualise the elements of the subject being taught because they use the same abstractions during the learning process. In the context of BlueJ this is classes and objects rather than lines of code or files in projects. In the context of authelo this is types of elements and types of events. In authelo this need becomes even more apparent and necessary given the potential diversity of the learning objects combined with it. The same notion is also reflected in the data format of the action indicators generated in the example tracing process to help the designer understand the current state of the learner and configure the rules for the automated support. In both cases authelo is able to adapt to the paradigm of the learning object being used and help the designer understand the semantics and conceptualise the process better and more accurately. This is related to both research objectives.

An interesting feature behind the BlueJ design argumentation is that learners should get immediate feedback in response to their actions during the learning process. Learners can visually inspect the results of their actions and that forces them to face problems immediately. This is something that influenced our thinking when designing FLIP and authelo. In FLIP there is an automated process that is continuously running in the

background to detect potential issues with the code. Once there is something worth discussing with the learner a discreet indicator appears that informs the user in a non-intrusive manner. Learning in FLIP is supposed to take place in a non-controllable way but the feedback is always immediately available for the same reasons as in BlueJ. The same concepts are also being considered in the design of authelo. Authelo offers an environment where the author can iterate over the same problem multiple times and get immediate feedback about the correctness of the approach followed without having to recompile, reload and repeat the learner actions for the particular task. This gives the author clarity about the flaws of the solution and streamlines the process. The same approach is also followed in the type of support developed through authelo for the learners. The outcome of authelo is formative and summative feedback that is immediately available at any time during the lifetime of the learning activity. Learners are free to consult this information at the time that is convenient for them to overcome obstacles in the learning cycle. This concept is related to the simplification of authoring research objective.

The user interface of Greenfoot was designed with the "gradual transition to coding" principle in mind. The idea is to let the learners engage with the content initially without code and once they feel comfortable with the concepts allow them to interact using lower level constructs. This reduces the initial cognitive load required as it smoothens out the learning curve and thus increases the chances of the learner to engage further and remain in the learning process longer. This principle gave us a direction when making decisions related to LFT and its potential uses. One of the main objectives behind LFT is to enable the definition of new high-level languages to hide details and allow easier manipulation of difficult concepts. Ultimately, at a GUI level these abstractions could be translated into graphical elements like the ones used in block-based languages to eliminate the need for the designer to memorise terms and semantics. This is related to both research objectives.

Another system that influenced our thinking is Salespoint. The assumptions and

considerations behind the design of Salespoint are that typical real-life projects are needed in the learning process to allow for ill-defined learning scenarios to be exploited through exploration. In this context it is especially difficult to support large student cohorts with limited resources. Salespoint is more a framework rather than a specific learning environment that has a narrow design orientation. Despite the fact that there was no microworld materialising these concepts in any of its incarnations we have seen, it inherently allows basic exploration and the discovery of knowledge in a constructivist way. Its design considerations are in line with established views in the literature claiming that ELEs are particularly useful for ill-defined domains (Lynch et al. 2006), they tend to be used with large cohorts of students where students tend to work in unpredictable ways. ELEs require significantly more support than other systems (Kirschner et al. 2006, Kynigos 1992, Mayer 2004) and support must be appropriate for that type of learning in order to lead to meaningful learning outcomes (Mavrikis et al. 2013). This influenced a lot our orientation towards exploratory learning and strengthened our belief that this is an area with a lot of potential as well as significant pragmatic constraints especially with regard to the development of automated support. Research in the area also revealed that it is possible to delegate part of this support to intelligent components (Bunt et al. 2001, Mavrikis et al. 2013). That, combined with the fact that very few attempts have been made to reduce the complexity of the authoring process and thus the entry threshold for both programmers and end-users (Blessing et al. 2007) lead us to the pursuit of a technology to complement that deficiency. This is related to both research objectives.

Programming Environments and Debugging Aids

These are software development environments equipped with enhancements that help users eliminate mistakes and build more concise and semantically correct code. A common theme that emerges from a comparative analysis of those systems can be summarised as follows:

Layered architecture: Most of these systems are equipped with different kinds of enhancements like automatic syntax highlighting, syntax checking and verification, code formatting, dynamic compilation / interpretation, dynamic visualisation of outputs etc. All of these components are being combined and used in different ways depending on the development scenarios implied by the different architectures used. In some systems like PECAN, MEMO-II, SUPPORT and ASA the notion of a layered architecture is more prevalent and well articulated. Especially in PECAN we see a system composed of three major categories of functionalities applied in an incremental fashion to gradually enhance the level of support required. This system inspired the layered architecture designed to support the development of web-based automated tutors in programming. This is related to the simplification of authoring research objective.

Language neutrality: Most of the systems developed in this era try to introduce a neutral language to express algorithmic concepts that are then translated into other real languages. We see this in PECAN, SCHEMACODE, DSP, BACCII and ASA. This is typically done through flow diagrams or even specialised pseudo languages like the Schematic Pseudo Code language is SCHEMACODE. The idea is to use a common language to help users systematically document code, understand better flow control and develop problem solving skills easier. None of these languages achieved the level of acceptance to become standards for these purposes. In the case of specialised languages the typical problem was the complexity and thus the steep learning curve implied. This notion of language neutrality inspired the work we have done with LFT. The premise in our case is that the de facto platform for the development of contemporary educational systems is the web browser. In the context of the web browser the de facto underlying and common language used in all systems is JavaScript. The idea to be able to define existing or new languages and use transpilers to generate JavaScript code dynamically to enable learners to take advantage of different learning environ-

ments with a known language or different languages with a known environment and achieve better learning or software development is the same as in the programming environments in the 80s. The difference here is that the learners are not expected to learn a different language to accomplish the learning tasks. On the contrary they are expected to use the language or system of preference for the learning activity and not stumble on artificial technical obstacles. The difficult part in this case is the specification of the new language. This task is done by a skilled developer only once. This is related to both facilitation of reuse and simplification of authoring.

Intrusiveness: Systems in this category typically interact with learners or users in a very controllable and intrusive way. Users' actions are being continuously monitored and once there is an element in the code that deviates from what the system expects there is immediate feedback that interrupts the development process and forces users to rectify it. Systems blamed for being too instructive in this respect are ASA, SUPPORT, STRUEDI, Example-Based Programming System and Software Design Laboratory. This for us was an example to avoid as it strengthened our belief that the intent should be exactly the opposite. Educational systems for learning programming should keep students engaged with the learning tasks for as long as possible. Interruptions in the learning cycle (Kolb et al. 1984) must be kept to a minimum and for that to happen help should be always available and given discreetly and in small increments in order to help the learners overcome issues that cannot deal with unsupported (Vygotskiĭ et al. 1978). Interestingly enough, a system that belongs to the same era named GENIUS aspired to achieve these very same objectives. The idea was to get the students to discover the problems themselves by keeping them engaged with the problem for as long as it takes to identify them. Support was given through a conversational component in the form of hints to suggest corrections. This system inspired to a great extent the design of FLIP. This is related to both research objectives.

Irrational code: Some of the systems in this category aimed to recognise patterns in the code that suggest some form of misconception and suggest possible corrections in an automated manner. This is a case where the code is syntactically correct but semantically incorrect. Typically systems that offer automated support for this utilise a knowledge base where these patterns are stored along with a reasoner that checks the representation of the current state against these patterns to determine the problem and generate the response. Systems that have related features to this are the University of Washington Illustrating Compiler, LAURA, The Debugging Assistant and GENIUS. These systems especially LAURA and The Debugging Assistant influenced a lot our direction to investigate further common student misconceptions and combine code quality tools with a knowledge base and a reasoner to provide support for them in FLIP. This is related to both research objectives.

Intelligent Tutoring Systems

These are systems specifically designed for educational purposes. They are typically equipped with sophisticated technology used to provide individualised support and adapt the level of instruction according to learner needs. These systems are intelligent and able to perform automated assessment and control the student trajectory throughout the learning process with the aim to achieve high level cognitive objectives. The three main architectural components typically found in those systems are a knowledge base to hold the domain knowledge on problems, solutions, and known errors, a learner model to hold information about the student and the level of understanding and a virtual tutor to orchestrate the learning process. This general architectural scheme at a high level influenced significantly the way FLIP was designed and implemented as all three components are present in the design. This is related to the simplification of authoring research objective.

Some of the issues identified in the previous category of systems can also be found here. For example these systems allow very little room for creativity as only known

solutions are accepted. An indicative example of that is Capra. This is, of course, identified as an example to avoid as it contradicts with the fundamental principles of exploratory learning. This had an effect on our decision to move away from strict guided learning and consider task dependent and independent support under exploratory settings.

Although in these systems it is recognised that students learn better when immediate feedback is always available, stated clearly in Lisp tutor, this support is given in an intrusive way that interrupts the learning process if there is something in the code that indicates a problem. As analysed in the previous section this is considered a deficiency in the process with a potentially negative impact and it was recognised as an example to avoid. This influenced the general orientation of the work presented in this thesis and thus it is not directly related to specific research objectives.

A common theme that emerges from the comparative analysis in this category is that these systems are fairly monolithic, tightly integrated solutions that offer limited configurability without low-level programmatic interventions. Thus, they are costly to develop and maintain and offer very little flexibility and reusability. This helped us realise that componentizing and decoupling AI components as well as creating external tools to develop them could move things forward. This would allow reusability of existing components as well as less expensive development and make these systems more approachable to people and educational institutions. This realisation influenced our decision to move towards reusability and simplification of authoring. This is related to both research objectives.

Finally, there is a system under this category that was quite inspiring and influenced the architectural design of authelo. This is the Graphical Instruction in LISP. The distinguishing characteristic of this system is that it allows the students to develop solutions in both directions. They can either start from the data and work forward to the solution or start from the solution and work backwards. The latter was quite unusual at the time but later on with the adoption of Test Driven Development and Behaviour

Driven Development it became the norm in software engineering. This reverse process is also used by the FRAME approach to facilitate development and testing of automated support for ELEs. According to this approach we first establish the level of support required, then we find the related concepts and the corresponding rules to activate it and finally we compose the evidence required to evaluate those rules. Both systems significantly influenced the architectural design of authelo. This is related to the simplification of authoring research objective.

Exploratory Learning Systems

In this category we have systems designed and developed specifically for educational purposes. These systems allow learners to discover knowledge through exploration and are typically equipped with intelligence and adaptability to support them. The number of systems we find in this category is not as big as in other categories and that possibly indicates that there is room for more work to be done. The key takeaway from the literature review is that they have very sophisticated technology that may include identifying student activities, proposing group formations, recognising learner behaviours, providing personalised support etc. Intelligence is typically built into the systems as there is no authoring interface available. Learning in this context offers a lot of opportunities but requires a significant amount of support. Automated support can significantly improve the effectiveness of the process but development in this context can be a very complex and resource-intensive process. Systems in this category tend to be domain specific with very little reusability allowances and there is very little work done in exploratory learning on programming. All these findings imply that there is room for contribution in this area and that played a significant role in the direction we followed throughout this research project towards exploratory learning.

One of the systems reviewed, Annie, is a domain-independent, generic platform intended to host exploratory learning environments. To an extent this inspired the design and development of MLP. This is related to both research objectives.

Automated Support Authoring Tools

These systems are generally ITSs with separate AI components and authoring tools to develop them. A common theme that emerges from the comparative analysis in this category can be summarised as follows:

High entry threshold: Development in most of the systems presented in this category typically requires expert knowledge and knowledge engineering skills. This characteristic can be found in SQL-Tutor, ASPIRE, Disciple and Demonstr8. This implies that even though there might be an authoring environment available, the difficulty to utilise it for the development of automated support may be prohibitive and not cost effective. This influenced our thinking into pursuing ways to lower the entry threshold for prospective users and make it more accessible to less skilled people. This relates to the simplification of authoring research objective.

Genericity: All the systems (SQL-Tutor, ASPIRE, Diligent, Demonstr8, CTAT) apart from two offer solutions that are domain and platform specific. Only Disciple and GIFT claim to be domain independent but they are platform specific. GIFT claims to be platform independent but it imposes several restrictions regarding integration and interoperability of external components and it is not possible to develop support for components residing outside of the platform. This is another characteristic that gave us a research direction towards solutions that are more domain and platform independent and thus as generic as possible. This led us to perform research on epiphytic integration systems like SEPIA and the iFrimousse architecture to resolve the problem of tight integration with platforms. This relates to both research objectives.

Authoring paradigm: All the systems allow manual insertion and editing of knowledge. Some also offer a more user friendly GUI (SQL-Tutor, CTAT). In some of them

knowledge is inserted directly by experts (SQL-Tutor, ASPIRE) whereas in others experts give a demonstration that is then translated into knowledge (Diligent, Disciple, Demonstr8). The only paradigm that significantly reduces the complexity of the process is the one given in CTAT. In this case knowledge is inserted by example. An expert does the learning activity as a student and this generates a tree that is then annotated to generate rules. This is very efficient compared to manual development. CTAT and all its derivatives played a crucial role in the design of authelo. A variation of the example-tracing approach was used to provide a domain independent alternative that can be used in an exploratory context. This was a significant research contribution that relates to the simplification of authoring research objective.

Authoring strategy: The strategy employed to perform the authoring process of automated support in all the systems presented in this category assumes that all of it is done by an expert in knowledge engineering that may also be a domain expert. This scenario may be adequate in systems that follow the guided learning paradigm but it is very limited in the context of exploratory learning. In exploratory learning authoring of support is a much more complex process of a multidisciplinary nature. Many experts may be required to combine their efforts at different parts of the process and this makes development even more demanding and challenging. This requires a strategy to alleviate this difficulty and this was addressed in the FRAME approach. This is a divide-and-conquer strategy intended to facilitate development and testing of automated support for ELEs. This approach reduces the complexity of the development process by separating and compartmentalising the concerns. That allows for a more clear definition of scope and communication interfaces and enables better communication and testability. The development flow moves backwards as it starts with the presentation of support, it then moves to the reasoning component and finally ends with the collection of evidence to support the reasoning. This strategy also played a crucial role in the design of authelo. This is reflected in both the architecture of authelo

as well as the authoring paradigm used. This is related to the simplification of authoring research objective.

Integration and Interoperability

This section describes architectures, frameworks and protocols designed to enable integration and interoperability of local and remote software components. An analysis of these technologies revealed common issues that should be taken into account when considering what to adopt or avoid and to what extent.

Abstraction and APIs: Although, even at the very beginning of this type of engineering, the design objective was to abstract everything including hardware, operating systems and programming languages and paradigms there were big hurdles to overcome heterogeneity. One of these hurdles was the high technical complexity in developing APIs for software components. We see this in both CORBA and DCOM. This along with other technical shortcomings were some of the factors that affected the acceptance of these technologies. The realisation of the difficulty to design a method to define simple and generic APIs with no implementation overheads that allow for diversity to manifest itself through them was crucial when we designed the interfacing of components in WIIL. This is related to the facilitation of reuse research objective.

Inter-network communication: Another shortcoming was the difficulty of software components to communicate with each other through firewalls. This problem was partially resolved with a compromise after the HTTP protocol started to be used for communication. HTTP is simple, it can carry complex data and even other protocols through tunnelling. Most people allow HTTP to pass through firewalls and that makes it a good candidate for data that travels across networks. In this project one of the basic assumptions stated is that the de-facto platform for educational software is the browser and consequently the protocol natively supported in this context (HTTP). This affected

the work performed during the technical spike to develop WIIL. This is related to the facilitation of reuse research objective.

Artificial overheads: This is related to the current and prevalent standard in integration of learning components with platforms (LTI). Integrating web learning components with web-based platforms typically requires a stringent procedure to be followed in order to ensure the robustness and stability of the intended system. Although it is difficult to argue with this statement there is a certain category of web components and a very populated one, that indicates the need for a lightweight alternative especially regarding the launch and initialisation protocol. This again played a role during the work performed to develop WIIL and is related to the facilitation of reuse research objective.

Cross-component management: The prevalent standard at the time WIIL was developed for interoperability of web components within the browser was OpenAjax. This system was like a central hub operating under a publish / subscribe scheme and anonymous broadcasting. There were a number of artificially imposed deficiencies in the name of safety and security diminishing significantly the potential of this system. These were the inability to perform p2p messaging, cross component property management, RPC and callbacks. This again affected significantly the design and development of WIIL and it is related to the facilitation of reuse research objective.

3

Exploring Possibilities

As explained in 1.3 this project follows the design thinking approach to make teaching programming easier. This is based on the following reasoning pattern:

WHAT + HOW → "make teaching programming easier"

The first component of this pattern that is to be addressed is the WHAT needs to be done so that we can achieve the desired value.

The material presented in this chapter is supported by the following papers: (Karkalas & Gutiérrez-Santos 2014*b*, Karkalas & Gutierrez-Santos 2014*c*, Karkalas & Santos 2014, Karkalas & Gutierrez-Santos 2015).

3.1 Literature Review and Domain Analysis - An Outline

This step, starts with a literature review in the areas of teaching and learning programming in order to get familiar with the fundamental concepts in the area of interest. Then, a domain analysis (Prieto-Diaz 1990, Ferré & Vegas 1999) follows on educational software for teaching and learning programming. Domain analysis is a common technique used in software engineering to help designers and engineers gain a better understanding of the level of applied technical expertise in existing software and make well-informed decisions about reusability of architectures, techniques, methods, technologies and components. This analysis typically involves the study of all the available material that derives from the software development life cycle including history of design decisions, testing plans, evaluations, manuals etc. The findings of this process follow:

- Learning programming is difficult (refer to 1.1.2 and 2.3)
- Teaching programming is expensive (refer to 1.1.2 and 2.2)
- Supporting learning programming is limited due to limited resources (refer to 2.2)
- Learning Programming in an exploratory manner offers more opportunities for learning and is more natural (refer to 2.2)
- The need for support in exploratory learning is much higher than in guided learning (refer to 2.2)
- A possible solution to this is to offer automated support (refer to 1.1.3 and 2.2)
- Current systems that offer opportunities for exploration lack automated support (refer to 1.1.4)

After this cycle we started formulating an idea about the WHAT needs to be accomplished. This is a system that facilitates exploratory learning in programming and offers automated support and adaptability.

3.2 Educational Ethnographic Study

At this stage we want to verify the above findings and have a closer understanding of the issues under consideration. The methodology chosen to accomplish this led us to immerse ourselves in the reality of these problems and have a first hand experience of the situation. We did an educational ethnographic study (Pole & Morrison 2003, Noblit 2003, Mills & Morton 2013) that involved the researcher taking part in three University courses as a TA and lecturer. The general aim of this was to collect qualitative information about the subjective reality of lived experiences of people involved in a specific educational context. As part of this study we also did a thematic analysis (Guest et al. 2011) in order to identify common themes and useful patterns in the data. The exact objectives for this follow:

- Verify findings in the literature
- Understand better the area of interest
- Understand processes, flaws and weaknesses in the currently used teaching approaches
- Identify potential opportunities for improvement, innovation
- Understand how learners approach learning
- Understand how learners solve their problems

This part of the research was conducted in computer laboratories where students were undertaking programming assignments and practical training. We were involved in the process as tutors and in this capacity we provided help and guidance. The approach was exploratory as there was no hypothesis to confirm. We used direct observation and non-structured interviews to record issues that took place in laboratory sessions of three introductory programming courses that involved 111 students in total. We used an observation sheet to summarise the findings of the session. Two of

the courses were postgraduate modules taught in Java comprising 42 and 44 students respectively and the remaining one was an undergraduate course taught in JavaScript. The research was conducted at the Department of Computer Science and Information Systems, Birkbeck, University of London during the academic years 2012-13 and 2013-14. The Java courses were fast paced, intense courses and covered much more material and in greater depth. Therefore, the data collection part for these courses took place only for the first four sessions of each term. The data collection was carried out by two teaching assistants and the lecturer. The teaching assistants collected the data which was then reviewed by the lecturer. I was a teaching assistant in the Java courses and the lecturer of the JavaScript course. The findings of this process follow:

- Students require a significant amount of support, especially during the early stages of learning
- Students have common initial misconceptions in programming
- Students require two types of support. TI (Task-Independent) support - support for common misconceptions and TD (Task Dependent) support - support for tasks given to them for practice
- Labs are very busy especially during the early stages of learning
- Support provided to the students is limited and may not be adequate
- The learning cycle is frequently interrupted and students may not be able to progress, feel intimidated and excluded

The outcome of this cycle is that we verified the research findings of the previous step, the need for exploratory learning and the need for automated support. We also saw how learners and teachers behave under stress and how that affects their self-efficacy. We saw how difficult it is to provide adequate, relevant and bias-free support to a very

diverse cohort and how difficult it is to prioritise support requests in a busy environment. The thematic analysis revealed the distinction between TI and TD support and the clear need of having an automated solution for the former, since, according to the literature there is no system that provides it.

3.2.1 The Data Collection Process

The purpose of this process was pure observation as the intent was to perform educational ethnography. We did not want to intervene with a purpose to change things, measure the impact of those changes etc, as that would fall under the scope of action research. The observation areas set were based on the objectives presented in section 3.2. These areas are depicted in the observation sheet presented in Appendix 3. As one of the concerns was to keep the data anonymised, the only demographic information recorded was the name and level of the course, the number of students and tutors present and the date.

The purpose of using the observation sheet was to facilitate the recording of incidents. Incidents are not only issues that relate to student problems that may require some attention from the available tutors but they may be observations related to other things like the effectiveness of a teaching approach, the tension or disappointment of students facing a difficult task, situations where tutors have to make decisions about whom to support when all students in the class have their hands up and time is pressing. These are just a few examples of the numerous issues that may be worth recording in such an environment.

The original planning for this observation was to use one observation sheet per session per observer and record all incidents that seem to have some value and relevance with the observed variables. According to the plan the tutors would record incidents on the spot, devote sufficient time to each incident and if deemed necessary

allow some time for a non-structured interview with the student involved. The plan at a theoretical basis seemed to be complete and adequate but in practice it proved to be very ambitious especially during the first sessions of the courses. The reality in those classes was that students were thrown a lot of material to cover and very little time to process it. Programming was not coming naturally to the majority of the students. Normally at the beginning of the process a mental leap was required as students engaged with mind twisting activities in the lab. The time for this new knowledge and skills to develop was very little and students quite often felt despair and tried to grab every opportunity to get some help in the process. Help of course is limited as the available resources are also limited (time and tutors). As a result of that tensions developed and tutors worked overtime even after the end of the sessions to accommodate these needs. When even that was not enough tutoring continued via email and even phone calls. Using the observation sheet under these conditions proved to be impossible. The only compromise that seemed to be sensible at the time was to let the session end and then spend some time to reflect, to recollect the issues of interest and record them a posteriori. Sometimes if an issue was stimulating enough there were discussions among the tutors to determine a generally acceptable opinion. Using the observation sheet like that was suboptimal as it was not following the plan but in the end this gave us enough insight about the points of interest.

One of the things that emerged from the process and somewhat changed the recording routine was the actual code produced by the students. After the first couple of sessions of the first course we realised that this may be a valuable source of information and we decided to record it as well. Instinctively the tutors started using the box under section 5 for it but after a while we decided to introduce a structural change in the sheet to make this more official and semantically correct. Until that time section 6 was what is now shown as section 7 in the present version of the sheet. It was quite interesting to see how divergent thinking led students to develop solutions we had not even imagined ourselves and to also think of the potential misconceptions these

may be related to. If the type of issue was already known we just agreed to write a descriptive title of the problem, otherwise a snippet of code along with some textual or diagrammatic indicators for the interesting parts were given. This component was then used to inform the concept inventory of common student initial misconceptions presented in 3.3.

Perhaps the most valuable learning from this process was the realisation that it is immensely more difficult for an observer to observe oneself than to observe others. There are certain things that require very careful and unbiased introspection to reveal themselves like the reasons and justification of certain behaviours. Spending consistently more time with one student and neglecting another is a typical example of something worth recording as this may not be related to the actual need of help required by these students. To uncover the true reasons behind this and try to see how this might affect the learning process is a very difficult task.

3.2.2 Managing Bias and Subjectivity

The observation areas considered during the data collection process are the following:

1. General info
2. Learners and the learning process
 - (a) Understand how learners approach learning
 - (b) Understand how learners solve their problems
3. Teachers and the teaching process
 - (a) Understand processes
 - (b) Identify flaws and weaknesses
 - (c) Recognise good practices
4. Verify findings in the literature

5. Further understanding of the area of interest
6. Student code
7. Identify potential opportunities for improvement and innovation

The data recordings expected in those sections are in almost all the cases qualitative. The only two exceptions are in section 1 where there are fields about student and tutor participation and in section 4 where findings from the literature review that preceded require confirmation. In both cases the expectation was to have a single quantitative value per field per session. In the remaining sections the expectation was to have potentially multiple qualitative recordings per field per session.

The fields in section 1 were used to monitor the tutor:student ratio in the sessions. The fields in section 4 required a boolean value indicating the subjective opinion of the observer for each statement - whether they agree with it or not given their recent experience from the session.

As mentioned in section 3.2.1 the completion of the sheet did not go according to plan. This was partially justified due to circumstances that we did not foresee during the design of the sheet like the very uneven distribution of workload across the different stages of the courses. One thing that made the situation worse was the ratio between tutors and students. The plan for this ratio was to be around

$$3 : 42 - 3 : 44 \approx 14$$

for the Java courses but in reality most of the time it was 2:44=22. These courses were quite intensive and a single tutor handling 22 people and observing at the same time was not a trivial task. We realised soon enough that a realistic approach for this was to let ourselves do the teaching first and reflect later. This worked quite well as long as we were disciplined and didn't leave things for the next day.

The initial plan for the parsing of the data was to be done once a week but this was

not effective. Even if tutors are very conscientious, consistent and they try to articulate observations carefully there is always a chance to not use the right wording to express something. This is especially true in multicultural and multilingual environments like courses taken by international students. In many cases the only way to ensure that the data reflects the intended meaning is to negotiate or verify this meaning with the observer. The longer the time between observation and verification the more likely it is for important information to be lost, overlooked or even worse, misinterpreted. When the information is recent and all the contextual details are fresh one can have a much more complete picture of the observation and draw well informed conclusions about causes, implications, consequences etc. For most of the observation sheets parsing was done the very next day if not after the session. This way the chance to obtain invalid or incomplete information due to miscommunication or misinterpretation was minimised. The primary objective at the stage of parsing was to determine the true meaning of the observation and retain as much of the contextual information as possible in the form of sidenotes. We did not perform any further processing on the data at that stage. The data was inserted into plain text files in a sequence. We used distinct files for each (sub)section of the observation sheet.

3.2.3 Thematic Analysis

After the observations were finished the data collected was processed to determine the main themes. The derivation of themes was done starting with a 1:1 mapping between observation areas and themes and through analysis of the data we tried to determine the components to change. Analysis entails regrouping of the data items and identification of relationships and patterns. That gave us new information and insight that led either to rearranging, merging, expanding or even defining new themes and thematic sections. The resulting themes follow:

1. Theme: Literature verification

-
2. Theme: Learners and the learning process
 - (a) Learner attitude
 - (b) Learner emotional state
 - (c) Learner behaviour
 - (d) Learning approaches
 3. Theme: Teachers and the teaching process
 - (a) Understand processes
 - (b) Flaws and weaknesses
 - (c) Good practices
 4. Theme: Potential opportunities for improvement, innovation
 5. Theme: Coding issues
 - (a) Task dependent
 - (b) Task independent

Summarization of the data collected under each theme gave us insight that influenced directions for further research. The findings in theme No 5 for example informed the work presented in section 3.3. The learnings from theme No1 combined with learnings in other themes gave a very strong verification of the research findings from previous steps. A lot of useful information was revealed and articulated and that helped us in the steps that followed. We quote some indicative examples:

Learner Attitudes

Learners usually have the preconception that the more help they receive the better for their learning. If they receive little help there is a chance to feel neglected or to think

that the tutor is not knowledgeable, motivated or conscientious enough to provide adequate support. The truth of course is that help should be just enough so that they can carry on with the learning task without interruptions. It is useful to know about this attitude in advance and handle the situation appropriately.

Learner emotional state

We observed that learners become frustrated when they think they run out of options and the code still refuses to cooperate. The emotional state can play a significant role in their learning and the learning of others when working in a group.

Learner behaviour

We observed that in every cohort there are some people that are excessively talkable and sociable. These are typically extrovert students that have no resistance to monopolise tutors' time given every opportunity, even if they don't really need the help. There are also people that behave in exactly the opposite way. This is even worse because in this case the problem does not manifest itself and requires intuition and empathy by tutors to uncover.

Learning approaches

Some students when they stumble on a problem they think that the solution is readily available somewhere and they try to solve it by trial and error using answers that they may not even understand. Other students try to understand the problem and once they have a possible solution they try it out. If this is successful they realise they learnt something and they move on to the next challenge.

Understand processes

Teaching and learning takes place under very pressing conditions. The material is substantial, time and resources are limited and the aim is to have completed everything

before the end of the session in order to be ready for the following one that might be the very next day.

Flaws, Weaknesses

Time is limited and tutors give short answers instead of thorough ones in order to support more students. Support may be biased and given in different ways to different people. Support may not be personalised enough due to lack of preparation and previous knowledge. Support time may not be evenly distributed among the students that require it. Students are given a lot of material to process under pressing circumstances as there is limited time and resources available. The work must be completed in time because the next session might be the very next day. Under these circumstances, tutoring takes place at a really hectic pace and is done in a reactive as opposed to a proactive fashion. In other words tutors end up chasing problems instead of anticipating and dealing with them in an orderly manner.

Good practices

Tutors that managed to monitor the progression of some students were able to provide more targeted support and see better results in their performance.

Potential opportunities

If there is a way to recall previous experiences and knowledge about student engagement and performance, support decisions will be much more well educated and appropriately targeted. If help is more available and better prioritised there will be fewer interruptions and less frustration.

Table 3.1: Procedural Programming Concepts

Procedural Programming Concepts			
	ID	Topic	D
1	PA1	Parameters/Arguments I	
2	PA2	Parameters/Arguments II	
3	PA3	Parameters/Arguments III	
4	PROC	Procedures/Functions/Methods	
5	TYP	Types	*
6	BOOL	Boolean Logic	
7	COND	Conditionals	*
8	SVS	Syntax vs Semantics	
9	AS	Assignment Statements	*
10	SCO	Scope	

3.3 Common Student Misconceptions in Elementary Programming

One of the main findings of the previous step was the conclusion that the students have common initial misconceptions in elementary programming. A by-product of the analysis and one with a very significant value to us was the misconceptions themselves. Although, this was not one of the initial objectives, it caught our attention early enough and we systematically recorded and categorised programming issues in terms of misconceptions. In that respect the process followed is similar to Grounded Theory (Strauss & Corbin 1997). We intentionally did not use findings in the literature throughout this process because we did not want to constrain our perception and therefore influence the results. Our findings were then compared with existing classifications of already recognised misconceptions in the literature.

After the data was collected we classified it using the Concept Inventory (CI) presented in Goldman et al. (2008). According to the results, the following categories were applicable in our sample 3.1, 3.2:

We did not expect our sample to exhaustively cover all the concepts identified by the Delphi experts in Goldman et al. (2008). The concepts covered were 16 out of 32

Table 3.2: Other Concepts

Other Concepts			
	ID	Topic	D
Object Oriented Concepts			
1	PVR	Primitive and Ref Variables	*
Algorithmic Design Concepts			
1	IT2	Iteration/Loops II	*
2	IT3	Iteration/Loops III	
3	IT4	Iteration/Loops IV	
4	IT5	Iteration/Loops V	
5	REC	Recursion	
6	AR1	Arrays I	*
7	AR2	Arrays II	
8	AR3	Arrays III	

in the CI (50%). The 3 concepts in the grey area were not part of the original CI proposed in Goldman et al. (2008). After all the actual misconceptions that emerge in a course may depend on many factors like the students' background, the language and programming paradigm used, the material covered, the intended learning outcomes and so on. Assuming that these things remain fixed for our courses, the aim was to identify what is the actual need of our students in terms of help. The fact that the concepts identified cover a large part of the Delphi CI and overlap at a proportion of 60% with the student misconceptions identified in Kaczmarczyk et al. (2010) is an indication that the elicitation process was effective. The asterisk under the 'D' column in the above table indicates that the same concept was also found in Kaczmarczyk et al. (2010). There were no object-oriented misconceptions identified (apart from one) since JavaScript is a prototype object-based (class-less) language and the related misconceptions would not be relevant. Furthermore, most of the difficult aspects of object oriented development, like inheritance, were not part of the JavaScript introductory course. Also, it was probably very difficult for the program design concepts to reveal themselves in the code, at least in this initial stage. There have been recorded incidents in the lab regarding design-related concepts but there was no usable code involved.

Therefore, this category was excluded from the set.

3.3.1 Importance of Student Misconceptions

Students are unique individuals with their own distinct particularities. They may have different educational, social, cultural backgrounds, different aspirations, preferences, skills and aptitudes. However, always, and regardless of subject, they tend to experience common problems during the learning process and especially during the early stages of it. This increases tutor workload substantially during those phases. Tutors learn these common student misconceptions through experience and they gradually become more ready to deal with those problems since they know what to anticipate with future cohorts. Undoubtedly, this is one of the skills that are perceived as being essential for competent teachers.

Of course, the actual misconceptions that emerge in a course may depend on many factors like the background of the particular student cohort, the language and programming paradigm used, the material covered, the teaching approach used, the intended learning outcomes, the level of support provided and so on. All of these factors may influence, to an extent the misconceptions that emerge but there is always a core body of concepts that the majority of the students have problems with.

These misconceptions are an essential tool in teaching. A teacher equipped with this knowledge is better prepared to deal with students' problems. Teaching can be more relevant, focused and adaptable to cover what students really need. Also, having this knowledge makes teaching more efficient, concise and relevant in the sense that similar problems are dealt with in the same way. Teachers don't have to waste precious time to explore the problem or improvise. They have a well thought plan as to how this needs to be approached and they execute it. Having this part solved, they only need to see how to adjust their teaching to the particular needs and wants of individual students. This is evidently an essential component of teaching any subject and

computer programming is no exception.

3.4 Understanding Challenges by Developing a Prototype

According to the results of the previous step we established that there are two types of support an automated system could provide. Observations revealed the common student misconceptions and the emerging need of providing support for them. Since this type of support is not related to specific problems or tasks given to students we could name it Task Independent (TI) support and distinguish it from its complement which is Task Dependent (TD) support. Another important fact is that, according to the literature, there is no system that provides TI support specifically. This is an ambiguous outcome in the sense that it may be an indication that there is no actual need for it. So, that is something that remains to be verified. One way to do that is to create a system and put it to test with real students (Henson & Knezek 1991, Wong 1993). That, as an objective, conforms with our need, at that stage, to see what the methodological, architectural and technical difficulties are in developing such a system from scratch. Therefore, the natural next step is to try to develop an exploratory system that offers free exploration and provides TI support on demand. The exact objectives for this follow:

- Study the technical feasibility for this type of support
- Discover and realise requirements for building such a system
- Explore opportunities to make a lean design that saves development time and effort
- See if there is a real need for it in an authentic educational context

The outcome of this process is a system called FLIP Learning that allows free exploration and offers automated TI support and adaptability. The design and development process of this system led to the following learnings:

-
- The web browser is the de facto development platform for educational software and that is the platform used for FLIP Learning.
 - Development of FLIP was technically feasible but not easy.
 - An architectural framework (layered architecture) that can be used as a roadmap for the development of automated tutors was designed.
 - The need to consider different parts of the architecture as distinct components was identified. This relates to the need for componentization and consequently interoperability between them.
 - An opportunity to reuse existing functionality and make the development easier was identified and exploited (code quality tools). This relates to the need for reusability.
 - There is an abundance of web components (widgets) that are freely available on the web and can be utilised for educational purposes. This, again, relates to the need for reusability.
 - Reusability of existing functionality implies integration and interoperability. There are many different approaches and technical challenges to consider.

Throughout this process we developed a more concrete understanding of what is needed to develop such a system and we realised the technical challenges for reusability and interoperability. We designed an architectural framework that is suitable for web-based automated tutors in programming and discovered opportunities for making this process less expensive. The materialised system derived from this process can now be used to verify the need for TI support.

3.5 Developing FLIP

This part starts with the premise that if common misconceptions are known and the teaching techniques to deal with them are also known then a software component specialising in those problems can be developed. This component can be integrated with automated tutors to give them the ability to deal with student problems in any context and regardless of the given learning task. Common student misconceptions represent a significant proportion of teaching, especially in the early stages and thus shifting this component to automated tutoring is important and adds value to the system. Automated tutors need to know what teachers know when they deal with common student misconceptions and for that we need a machine processable body of knowledge that encompasses all these misconceptions as well as the teaching decisions / actions to address them. The following sections describe a simple method to derive a usable component that enhances the teaching ability of automated tutors. This process involves the derivation of a concept inventory (CI) that captures these misconceptions, the design of a visual rule editor that can simplify the production of rules and the design of a simple reasoning mechanism that can process those rules to derive actionable decisions in teaching. There is an actual implementation of the rule editor and the reasoner that can be used to showcase the capability. Both components can easily be integrated with learning platforms and reused to enhance any customisable programming tutor on the web. That can simplify a lot the authoring process of such tutors.

3.5.1 Knowledge Elicitation

In practical terms student misconceptions in programming correspond to certain formations of source code. These formations (patterns) could effectively be represented as sets of characteristics in the form of logical expressions. The first step in the process of modelling is to identify the patterns that indicate such misconceptions. As stated before, what we need to cover is common misconceptions in elementary programming. It

is assumed that this knowledge covers the basic concepts in an introductory programming course and does not fully cover the complexities of the programming language used or the more advanced object-oriented or other features of it. The intention is to capture only the misconceptions that correspond to basic language use and elementary algorithmic thinking. The concepts elicitation for the construction of the CI was a derivative of a wider ethnographic research that took place in computer laboratories used by University students for practical training in programming.

3.5.2 Knowledge Representation

For this part a rule-based representation technique (Newell 1962, Buchanan & Duda 1983, Duda & Shortliffe 1983) was used. This technique is efficient, easily implementable and particularly suited for problem-solving contexts such as programming. It involves production rules containing if-then or situation-action pairs.

The following are typical rule components:

- Initial state
- Goal state
- Legal operators, i.e. things you are allowed to do
- Operator restrictions, i.e. factors which constrain the application of operators

In a rule-based system, much of the knowledge is represented as rules, that is, as conditional sentences relating statements of facts with one another to identify the initial state. There is also the consequent part that states what action must be taken in case the rule is executed. The assumption in this case is that rules are generated by experts and form the knowledge base of a system that can automate certain tasks based on this knowledge. In the domain of programming the conditional part of these rules corresponds to one or more characteristics identified in the code. These are the facts that

show the current student understanding of the task. This is where the CI comes into play when developing an intelligent tutoring system (ITS) for programming. Experts can translate the concepts in the CI into the conditional part of those rules. The consequent part of the rules corresponds to the action that needs to take place in case they fire. There is no definitive guide as to what this part must include. This part mostly reflects the educational side of the process and more specifically the teaching approach followed. In an ITS this could take the form of cues, references to the documentation, visual indicators in the code, code tracing and code refactoring. For this part a visual rule definition language was implemented and made available through a web-based editor - see figure 3.1. The name we use for it is rule editor (RE). Experts can utilise the RE to synthesise rules and then inject them into a system that is able to process them. Typically, the processing engine for such rules is a reasoner.

Figure 3.1: The Rule Editor

The Rule Editor interface is divided into several sections:

- Form Fields:**
 - Rule Name (ID): 900-4
 - Misconception: Understanding the difference between block scope and function scope.
 - Issue: A variable is declared within a block of statements.
 - Solution: Move the declaration before the block.
- Fact => Rule:**
 - subject <- property
 - property <- subject
- Consequent:**
 - Reference
 - Refactoring
- Buttons:** JSON, Exit, Submit Rule
- JSON View:** A text area containing a JSON rule definition for a misconception about block scope.
- Facts:** A table with columns: part 1, part 2, part 3, delete.

part 1	part 2	part 3	delete
v1	is	var	delete
v2	is	block	delete
v3	location	v1	delete
v4	location	v2	delete
v5	is	structure	delete
v6	relates	v5	delete
v7	location	v5	delete
- Rules:** A table with columns: part 1, part 2, part 3.

part 1	part 2	part 3
v1	=	empty
v2	=	empty
select a value	select an operator	select a value
v4	contains	v3
select a value	select an operator	select a value
v6	==	v2
select a value	select an operator	select a value
- References:** A table with columns: name, link, delete.

name	link	delete
- Refactoring:** A table with columns: item 1, operator, item 2, delete.

item 1	operator	item 2	delete
v3	before	v7	delete

As shown in figure 3.1, this tool has the following features:

- It gives a unique identifier to the rule along with the misconception it associates with from the corresponding CI.
- It also gives a description of the (practical) issue that corresponds to the concept along with the recommended solution for it. The issue and the solution can both be thought of as parts of the consequent.
- The conditional part comprises statements that identify certain properties in the code and statements that combine these properties together using operators. This part shows the current student understanding.
- The consequent, apart from the issue and the solution can include references that explain the problem in more detail (like documentation) and refactoring directives.
- The rule, as a whole, can then be exported in the form of a JSON object and inserted into a knowledge base.

The above rule represents a common misconception that typically reveals itself when novice programmers declare variables without considering scope constraints. This is a common problem in any language but it becomes especially evident with languages that support variables with function scope. A particular example is Javascript that supports the declaration of variables with the keyword `var`. These variables confuse novice programmers as they can refer to them even outside the block they are declared in without any issues. Let's consider the following example:

```
var x = [2,5,1,8,8,9];
for(var i = 0; i <= 5; i++)
{
    var sum = 0;
```

```
    sum += x[i];  
}  
alert(sum);
```

This code would produce the facts given in figure 3.2 regarding student understanding.

Based on this information an expert could express a rule that identifies the issue and offers some form of support to the learner. A flow diagram that could help in this process is given in figure 3.3.

This diagram reflects the rule we composed with the rule editor earlier and it reads as follows:

1. variable v1 (variable in the rule) is a var (var is variable in the code)
2. v1 is not null (if there is no v1 we stop the process)
3. variable v2 is a block (the area enclosed with curly braces in the code)
4. v2 is not null (if there is no v2 we stop the process)
5. variable v3 is the location of v1 (the location the variable that corresponds to v1 in the code)
6. variable v4 is the location of v2 (the location the variable that corresponds to v2 in the code)
7. v4 contains v3 (if the block does not contain the variable we stop the process)
8. variable v5 is a structure (the for loop in the code)
9. variable v6 is something that relates to v5 (something that relates to the for loop)
10. v6 is the same as v2 (the block identified earlier relates to the for loop)
11. variable v7 is the location of v5 (the location of the for loop)

This is the initial state of the rule that identifies the problem. The consequent comprises the parts shown in figure 3.4:

It is up to the learning environment to decide on how to utilise the consequent. Ideally these three components should be used as different levels of adaptive support based on some criterion. The refactoring component is a direct instruction on how to resolve the problem in an automated fashion. The hosting system should be able to utilise this information and refactor the code by changing the location of variable v1 (v3 is the location of v1 in the code) and place it just before the location of v5 (v7 is the location of v5 - the for loop). More examples like the above are presented in Appendix A.

3.5.3 Knowledge Processing

The processing engine for the above rules was designed and implemented as a rule-based expert system. This is the component that handles student misconceptions that need to be resolved. This component can help the host learning environment to interact with the student when a misconception is identified and support is requested. It takes the role of the teacher and repetitively exchanges information with the student in order to assess the current situation, identify the problems and provide individualised support whenever possible. An abstract view of this component is given in figure 3.5.

This system accepts two inputs: rules (misconceptions) and facts (current student understanding). Rules are inserted by experts and form the knowledge base of the system. The conditional part of these rules corresponds to one or more characteristics identified in the code. The consequent part of the rules corresponds to the action that needs to take place in case they fire. The rule formation takes place in the Knowledge Acquisition Component (KAC) and then the resulting structures are stored permanently in the Knowledge Base (KB). Experts can utilise the RE that is part of the User Interface (UI) to synthesise rules and instruct the KAC on how these rules should be

constructed.

Facts are inserted dynamically into the system as the learners insert code into the editor. The fact formation takes place in the Fact Acquisition Component (FAC). The insertion process in this case may not be direct. Users may initiate this by selecting a part of code in the editor and ask for help or the system can monitor student activity and initiate the process implicitly. The reasoner invokes FAC to generate the facts. The facts are objects that are formed as a result of static code analysis. The selected code is parsed and analysed and the resulting constructs (if any) depict the code status. The patterns identified in the code are effectively transformed into a vector of characteristics. These characteristics take the form of atomic formulae (variable/predicate pairs) and are submitted into the Working Memory (WM) as facts.

If there are facts that satisfy the conditional part of one or more rules, then these rules get selected by the Rule Activation Component (RAC) and placed in the Agenda (A). If there are more than one rules in A then the system has to select one. One way to do this is to select the rule that corresponds to the simplest concept in the list. That means the one with the lower number of references to characteristics of the code. Firing the rule entails the execution of its consequent. This can be either some form of output to the user through the Student Support Component (SSC) or the creation of a new fact. In this case the fact is inserted into WM directly by A. The process carries on until there is no active rule to be processed or in other words until there is no misconception to be resolved.

As teaching takes place, the system provides feedback to the user in the form of help and/or questions. If the user is asked a question by the reasoner, then the answer may result in a direct formation of a fact through FAC. Feedback may also be given in the form of justifications as to how the decisions have been made by the reasoner. This service is provided by the Explanation Component (EC).

3.5.4 The PoC

FLIP Learning stands for Flexible, Intelligent and Personalised Learning. FLIP is an exploratory learning environment designed for teaching introductory programming to University students through inquiry-based scenarios and open-ended ill-defined problems (Savery 2006). The main design objective in FLIP is to enable students to work on tasks continuously and learn without interruptions. The idea is to provide timely and individualised support on any issue that hinder the learning cycles even if the task the student is working on is not known. This is based on the premise that laboratory sessions are very busy especially during the early stages of learning. There is limited time and human support in those sessions and that combined with the difficulty that tutors have to prioritise properly and make bias-free and well-informed decisions about the type and level of support needed might have a negative impact on the effectiveness of the process.

Learning in a computer laboratory is the sequence of actions that take place during a learning cycle (Kolb et al. 1984, Konak et al. 2014). There is a pattern that students follow when they engage with a task. The sequence of actions they execute follows a cyclical process. In every round students attempt to code something that brings them closer to the completion of the task at hand. Sometimes this is interrupted by the inherent inability of the student to move forward. That can be a lack of knowledge or a misconception. In some cases the only way to overcome the problem is to receive enough and relevant help in a timely fashion. Typically, the tutor intervenes and provides the help needed so that the student can move on and complete the cycle. The student conceptualises the issue and then confirms the validity of the new knowledge through active experimentation. In busy laboratories these interruptions may be very frequent and support may not be enough. If these cycles get interrupted, then learning gets interrupted and if support is not adequate or it cannot be provided timely, then inevitably the learning process becomes less effective.

FLIP is designed to overcome these problems by moving a lot of the workload from human tutors to an intelligent agent that is able to provide individualised support on demand. Support is task-independent (TI) in the sense that it may not be related to the task the students work on. FLIP is equipped with a knowledge base that corresponds to the CI presented above and a reasoner designed according to the scheme presented in 3.5.3. Both are loosely coupled pluggable components that may easily be reused in any other learning system. Support is also adaptive. There is a component that encapsulates a simple user model that monitors the misconceptions encountered so far as well as the level of support requested. Initially, the level of support provided for a misconception is as little as possible. If support is not enough, subsequent iterations will provide more detail and materials. Support includes simple cues and suggestions, references to documentation, visualisations using code tracing and direct coding suggestions using refactoring in the editor. If the student has repeatedly used the reasoner for the same misconception at the highest level of support then the system generates an event that provokes the intervention of a human tutor to resolve the issue. The screenshot given in figure 3.6 shows a number of misconceptions identified by the reasoner. The student has selected the one that seems to be more relevant to the problem and an explanation is given. The explanation is not enough and the user requests more help. A reference to the relevant part in the documentation is provided - see figure 3.7. More help is requested and a tutorial is given that includes a video that describes how the coding should be done - see figure 3.8. If this is not enough and more support is requested the solution is shown - see figure 3.9. The next level of support is a code tracing visualisation that shows what is happening in the computer's memory as the code gets executed - see figure 3.10. Finally, the last level of support is code refactoring - see figure 3.11. All these levels of support can be changed by using the RE and updating the particular rule.

Figure 3.2: Facts

(index)	subject	relation	property	location
0	"x"	"is"	"array"	"{"start":{"lin..
1	"x"	"location"	"{"start":{"lin..	"{"start":{"lin..
2	"x"	"length"	6	"{"start":{"lin..
3	"s1"	"is"	"structure"	"{"start":{"lin..
4	"b1"	"is"	"block"	"{"start":{"lin..
5	"b1"	"location"	"{"start":{"lin..	"{"start":{"lin..
6	"sum"	"is"	"var"	"{"start":{"lin..
7	"sum"	"location"	"{"start":{"lin..	"{"start":{"lin..
8	"l1"	"is"	"literal"	"{"start":{"lin..
9	"l1"	"location"	"{"start":{"lin..	"{"start":{"lin..
10	"l1"	"value"	"0"	"{"start":{"lin..
11	"sum"	"value"	"l1"	"{"start":{"lin..
12	"b1"	"includes"	"sum"	"{"start":{"lin..
13	"b1"	"includes"	"sum"	"{"start":{"lin..
14	"x"	"subscript"	"i"	"{"start":{"lin..
15	"b1"	"includes"	"x"	"{"start":{"lin..
16	"b1"	"includes"	"i"	"{"start":{"lin..
17	"s1"	"relates"	"b1"	"{"start":{"lin..
18	"s1"	"location"	"{"start":{"lin..	"{"start":{"lin..
19	"f1"	"is"	"for"	"{"start":{"lin..
20	"i"	"is"	"var"	"{"start":{"lin..
21	"i"	"location"	"{"start":{"lin..	"{"start":{"lin..
22	"l2"	"is"	"literal"	"{"start":{"lin..
23	"l2"	"location"	"{"start":{"lin..	"{"start":{"lin..
24	"l2"	"value"	"0"	"{"start":{"lin..
25	"i"	"value"	"l2"	"{"start":{"lin..
26	"f1"	"init"	"i"	"{"start":{"lin..
27	"o1"	"is"	"operator"	"{"start":{"lin..
28	"o1"	"location"	"{"start":{"lin..	"{"start":{"lin..
29	"o1"	"value"	"<="	"{"start":{"lin..
30	"f1"	"test"	"o1"	"{"start":{"lin..
31	"f1"	"test"	"i"	"{"start":{"lin..
32	"l3"	"is"	"literal"	"{"start":{"lin..
33	"l3"	"location"	"{"start":{"lin..	"{"start":{"lin..
34	"l3"	"value"	"5"	"{"start":{"lin..
35	"f1"	"test"	"l3"	"{"start":{"lin..
36	"f1"	"update"	"i"	"{"start":{"lin..
37	"sum"	"value"	Array(1)	"{"start":{"lin..
38	"f1"	"includes"	"sum"	"{"start":{"lin..
39	"f1"	"includes"	"sum"	"{"start":{"lin..
40	"f1"	"includes"	"x"	"{"start":{"lin..
41	"f1"	"includes"	"i"	"{"start":{"lin..
42	"f1"	"location"	"{"start":{"lin..	"{"start":{"lin..

Figure 3.3: Sample Rule



Figure 3.4: The Consequent

Issue:

Solution:

Refactoring

item 1	operator	item 2	delete
<input type="text" value="v3"/>	<input type="text" value="before"/>	<input type="text" value="v7"/>	<input type="button" value="delete"/>

Figure 3.5: The Reasoner Architecture

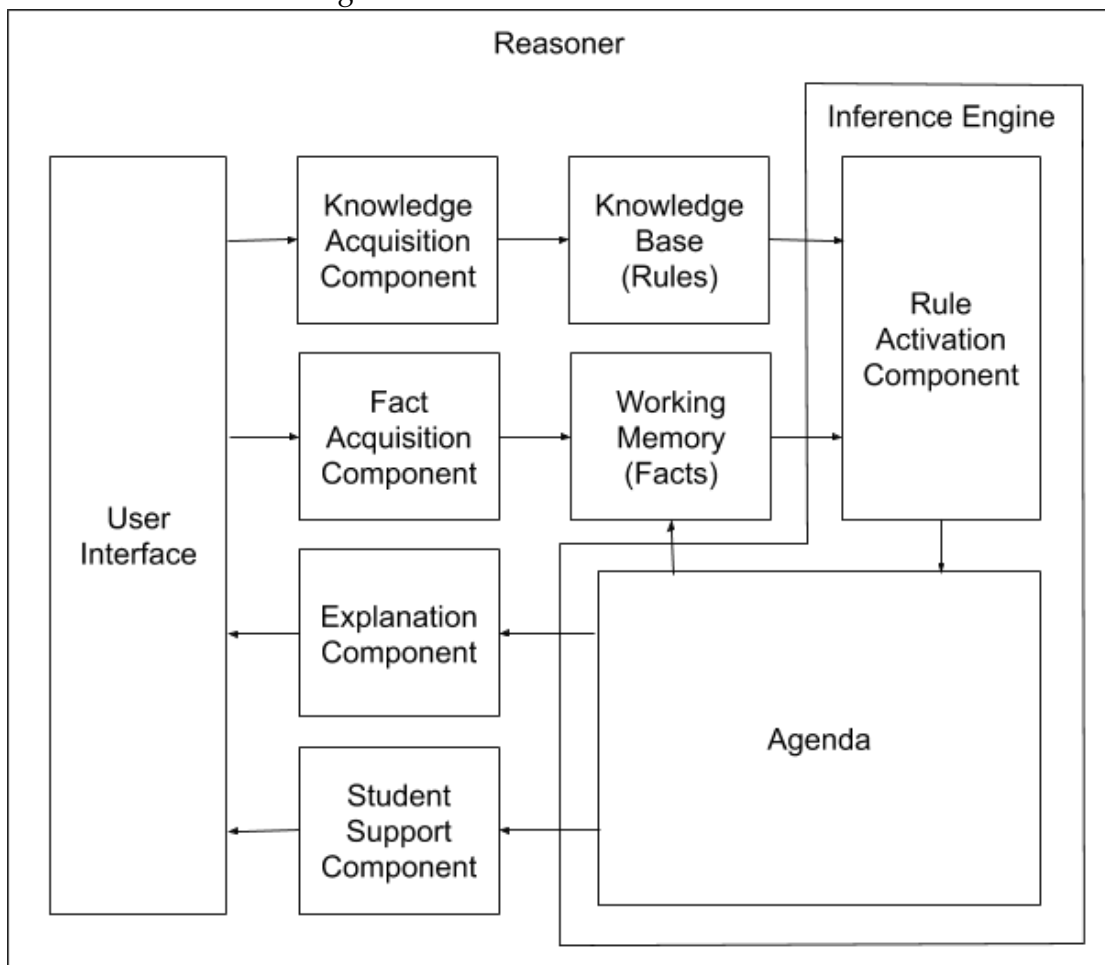


Figure 3.6: Misconceptions identified by FLIP

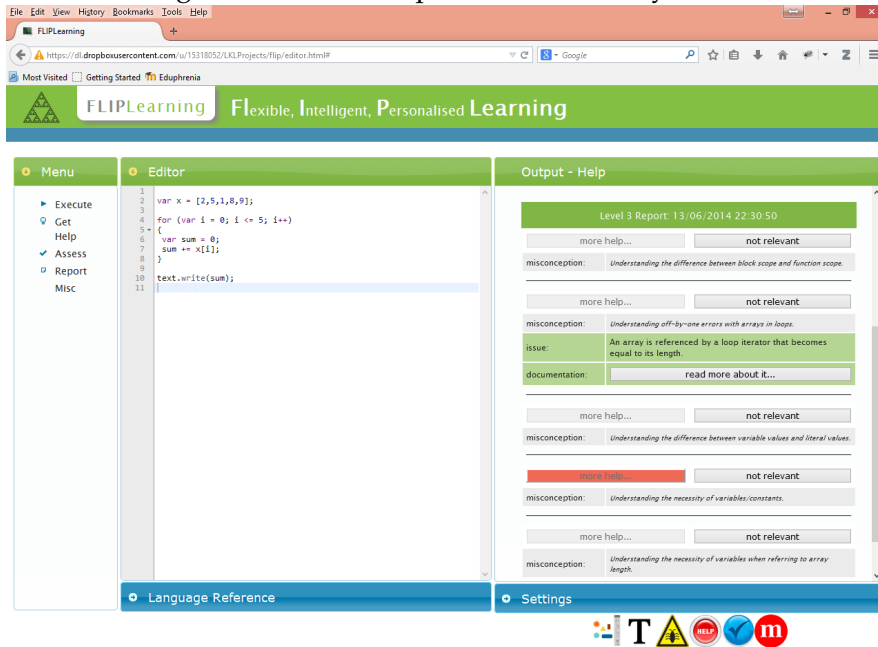


Figure 3.7: Initial Support

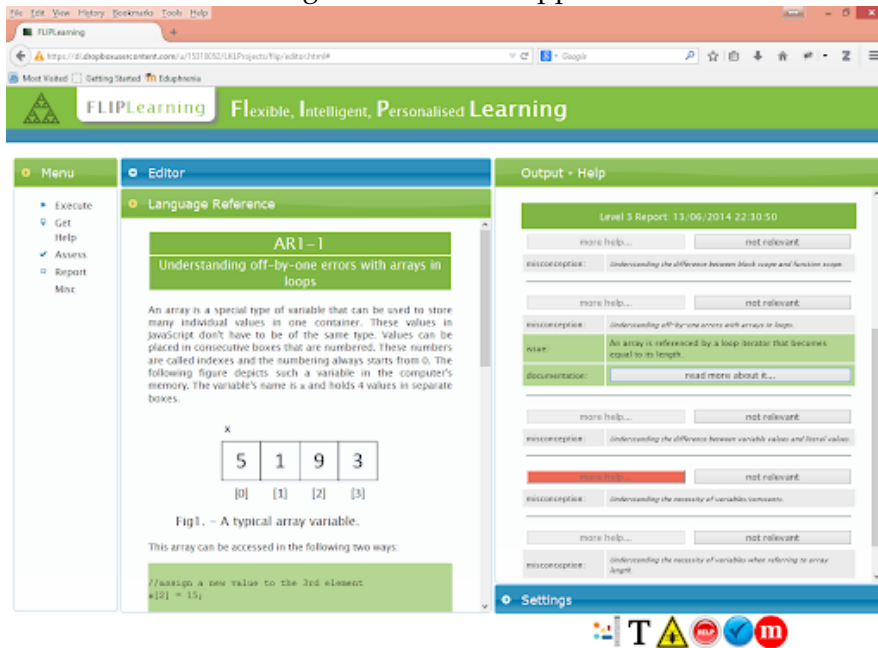


Figure 3.8: Third Level of Support

The screenshot shows the FLIP Learning interface. The Editor pane contains the following code:

```
var i = 0;
for (i = 0; i < 4; i++)
{
  alert(x[i]);
}
```

Below the code, there is a text explanation: "The value of *i* initially becomes 0 and after every iteration it gets incremented by 1. When it becomes equal to 4 the loop terminates and the final value (correctly) is not used to reference an array element. It is not uncommon for beginners in programming to use by mistake the <= operator in the loop condition. This would extend the range of values of *i* by one making the very last value an invalid index. In this case the value 4 would be used to reference the 5th (nonexistent) box in the array. The following animation clarifies the issue further:"

An animation shows an array with 4 elements: [5, 1, 9, 3]. The loop variable *i* is shown incrementing from 0 to 3, with corresponding elements being displayed. The loop terminates at *i* = 4.

The Output - Help pane shows a report for "Level 3 Report: 13/06/2014 22:30:50". It lists several misconceptions and issues:

- misconception:** Understanding the difference between block scope and function scope.
- misconception:** Understanding off-by-one errors with arrays in loops.
- issue:** An array is referenced by a loop iterator that becomes equal to its length.
- documentation:** read more about it...
- misconception:** Understanding the difference between variable values and literal values.
- misconception:** Understanding the necessity of variables/constants.
- misconception:** Understanding the necessity of variables when referring to array length.

Figure 3.9: Fourth Level of Support

The screenshot shows the FLIP Learning interface. The Editor pane contains the following code:

```
1 var x = [2,5,1,8,9];
2
3
4 for (var i = 0; i <= 5; i++)
5 {
6   var sum = 0;
7   sum += x[i];
8 }
9
10 text.write(sum);
11
```

The Output - Help pane shows a report for "Level 3 Report: 13/06/2014 22:34:13". It lists several misconceptions and issues:

- misconception:** Understanding the difference between block scope and function scope.
- misconception:** Understanding off-by-one errors with arrays in loops.
- issue:** An array is referenced by a loop iterator that becomes equal to its length.
- documentation:** read more about it...
- solution:** Replace <= with <
- misconception:** Understanding the difference between variable values and literal values.
- misconception:** Understanding the necessity of variables/constants.
- misconception:** Understanding the necessity of variables when referring to array length.

Figure 3.10: Fifth Level of Support

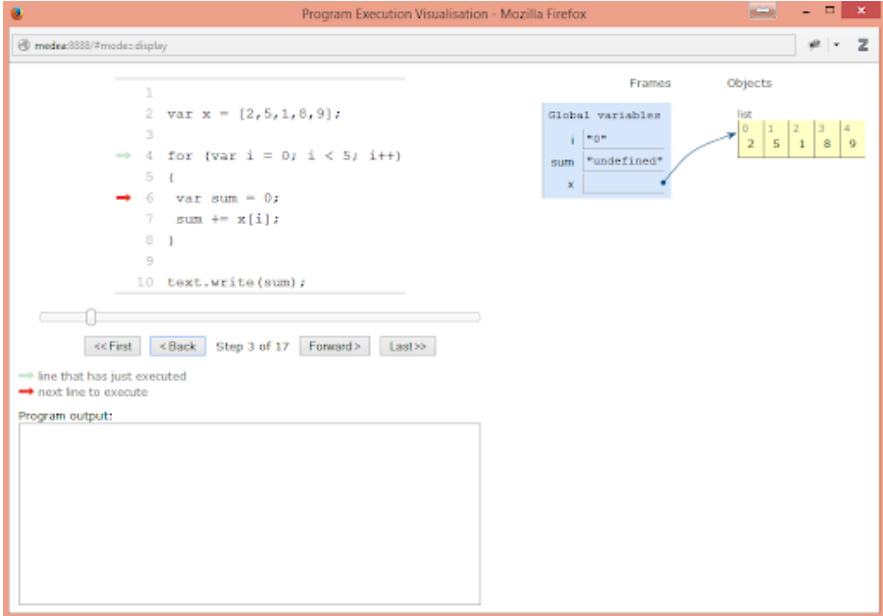
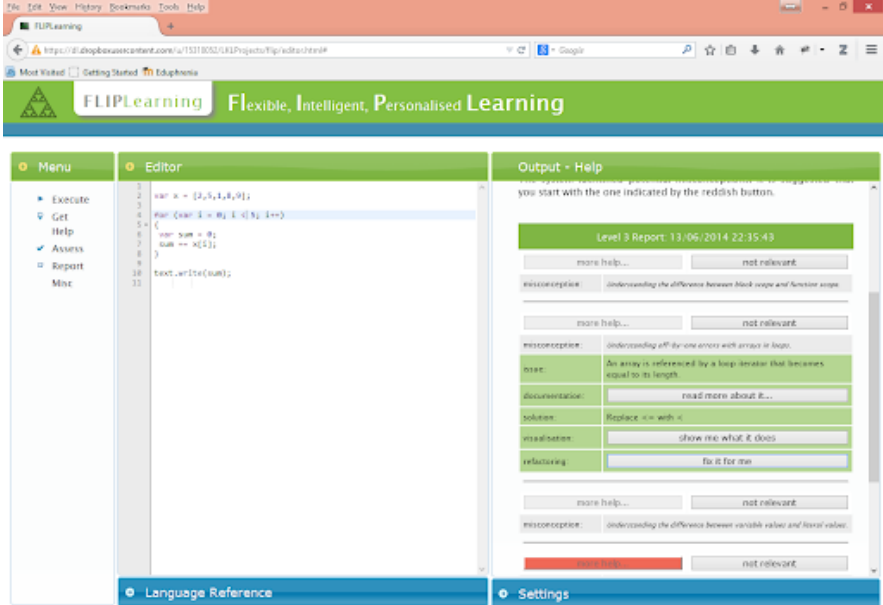


Figure 3.11: Sixth Level of Support



3.6 An Important Outcome: The Intelligent Tutor Layered Architecture

Nowadays the web browser has become the de facto target platform for software applications and educational applications is no exception. This inevitably influences the way we design those systems. One of the first things we need to consider is the fact that the machine language of the browser is JavaScript. Any other language developed to work on a browser must be ultimately executed by a JavaScript engine. Teaching a programming language in this context unavoidably is affected by the particularities of this language and its runtime environment. Students have to execute code in this environment during the learning process and get feedback that will help them complete their exercises. The nature and level of feedback as well as the time that gets generated highly depends on how this environment operates. Server-side approaches that may utilise other runtime environments introduce latency and don't scale well. Therefore we don't consider them as viable approaches. JavaScript is an interpreted language. As such it does not provide messages that you would normally get from a compiler about syntax problems. You expect to receive such messages at run time. It is self-evident that the sooner a coding problem becomes known to the programmer the better. This way the problems are more evenly distributed in time and the programmer does not have to face all of them at the end of the development cycle. The proposed architecture for the development of intelligent tutors in this context comprises six layers. The first five layers were used for the development of FLIP Learning.

Layer 1 (L1) : Syntax Checking A system that teaches programming typically comes with a code editor because students need the means to insert code into it. That is why such systems, normally, look like specialised integrated development environments (IDEs). Depending on the level of sophistication of the system, its editor might be equipped with features that help in coding. In our case this also means help in the

learning process. Therefore, in an automated tutor, the first layer of support is typically implemented within the code editor. Modern code editors can be thought of as components that may be integrated with learning platforms to offer a service. Typical code editors natively support syntax checking based on other web components like code quality tools. These tools perform static code analysis and recognise the structure of the code and the individual language constructs used. This can then be used for automated code indentation, highlighting and syntax checking. All of these can take place as the learner types in the code. Ideally, it should be possible for these features to be switched on and off to accommodate different teaching approaches and learning scenarios.

Layer 2 (L2) : Code Quality Checking The level of support implemented in the editor, normally cannot go beyond syntax checking. In other words, the editor may be able to do what a compiler does in a compiled language. Highlighting syntax errors is essential but not enough and that is because syntax errors may not be related to potential student misconceptions. For this a different component must be used. The second layer should be implemented by a separate component that performs full code quality analysis. This can be a similar component to the one used in the code editor but in this case it is used directly and its API is not hidden within the editor. Direct access entails more options in terms of configurability. Typically, in this layer we are looking to capture cases where there are suspicious patterns in the code that might indicate student misconceptions. The assumption here is that the code in terms of syntax is valid and we may not have logical errors either, but the code indicates that the student has implemented something without following good programming practices. An example of that could be a variable declaration that indicates that the student has not understood the notion of variable scope. The misconceptions captured in this layer correspond to a proper subset of the CI presented in 3.3. This layer corresponds to a preprocessing phase that takes place before execution. Problems identified at that stage can be ig-

nored by the learner as they will not prevent execution. The level of support that is possible at that stage includes cues, references to documentation, code segmentation, visualisations and code refactoring.

Layer 3 (L3) : Beyond Code Quality Checking The third layer is expected to cover all the cases that a typical code quality tool cannot identify. This, again, must be implemented as a separate component or a composition of components. At this level we must go deeper and utilise tools that can parse student code, perform static analysis on it and transform it into an equivalent representation that gives us all the structural and language properties of the code in a form that can be processable by reasoners. We need to define rules that cover all the remaining cases in the CI that the previous layer fails to detect. As indicated above this component is expected to respond to known student misconceptions / problems that are not necessarily related to bad coding practices and therefore may not be detectable by code-quality control tools. Examples of such misconceptions are off-by-one errors when using arrays in iterative loops, good use of variable types, implicit type conversions and unnecessary code repetition.

Layer 4 (L4) : Task-Specific Problem Checking The assumption in the two previous layers is that the student is given support on known common misconceptions that have to do with the understanding of language rules and algorithmic thinking in general. This is what the CI covers, but students in practical sessions may be given certain tasks to complete that relate to some project or assignment. This imposes an additional objective that is obviously not covered by layers 2 and 3. In this case support must be specific to the requirements of the task or as we say it is task dependent (TD). Respectively, we can think of all the cases covered by the CI as task independent (TI) support. This level of support is more difficult to conceptualise and implement. There is nothing static or known in advance and it is difficult to anticipate certain student behaviours, especially if the level of freedom in the working environment is relatively high which

is typically the case in exploratory learning environments. In this case we need to start from the actual requirements of the given task. The requirements typically may cover the following two aspects of a problem: The task specifies what needs to be achieved and it may also specify how the solution must be implemented. So, it is the the **what** and the **how** that we need to provide support for. In inquiry-based learning scenarios the latter might not be very restrictive. Nevertheless, we need to have a process in place through which adequate support may be provided for both.

The **what** is something that indicates our expectation about what the target situation should be. This is normally reflected by the outcome of the whole process. That is the output generated by executing the code and the artifacts created in the learning environment as a result of that. The **how** reflects the decisions the student made throughout this process as to how to approach the problem. In terms of code this reflects certain algorithms that might have to be used so that the resulting system behaves or performs as expected. For that part we may have in our disposal indicators of student activity that reflect the strategy the student followed in order to get to this point. All of this information needs to be translated into a form that provides us with a representative view of the system at its final stage so that we can assess both the implementation details and the outcome. The outcome typically can be assessed through dynamic code analysis and testing. Dynamic code analysis presupposes that the code gets compiled and executed in a runtime environment. That can provide indicators about the actual memory footprint and processing power needed to execute the code. If responsiveness and overall performance are requirements then these indicators could give some feedback about how well those requirements have been addressed. Dynamic testing could be used to assess the correctness of the output in terms of data. The system is given data as input and examines the resulting data after the processing takes place. This component can also examine the artifacts that get generated as a result of the processing in the learning environment. That could be a geometrical shape and all of its characteristics in an environment like Geogebra. This component typically checks

flaws in the **what** part of the requirements. The **how** may be assessed using static code analysis and machine learning or other techniques. Static code analysis can provide us with a representation of the code that is machine processable and shows whether certain programming patterns have been used. Machine learning techniques could be used to process a series of programs in order to learn how to identify those patterns and then use the resulting models to tell us, with some degree of confidence, whether new code is similar to the pattern we expect. If, for example, there is a requirement that says that the bubble sort algorithm should be used in a solution, then this component may be able to provide the student with feedback that shows the extent to which the approach followed is in line with the given requirements.

Layer 5 (L5) : Adaptability Having the means to provide automated support in teaching may not be enough if this support does not take into account the individual characteristics of students. Not all students learn in the same way. Individual students experience different problems during the learning process and deal with those problems in different ways depending on their idiosyncrasies and abilities. Each student is a different case, and the things that differentiate them from each other can significantly influence the learning process. Students expect individualised support that reflects their particular approaches and practices. Tutors, on the other hand, are expected to make bias-free and well-informed decisions about the type and level of support needed in every case and respond accordingly. The latter presupposes that tutors have a deep knowledge of students' profiles and the ability to analyse previous activity on the spot in order to provide suitable and adequate support in every case. Support in this context is a multi-faceted and complex task that requires a lot of preparation, expertise, time and resources. Decisions must be based on a multitude of criteria and a considerable amount of data about students. Human tutors, regardless of expertise, have a difficulty to recall such information and make these decisions on the spot with accuracy. Failing to provide individualised support to students can be a serious draw-

back as it may exclude some students from the learning process. In multicultural and diverse environments this may be especially problematic.

Automated tutoring can be made adaptable based on a multitude of criteria. The following is by no means an exhaustive list:

- Learner profile (assuming profiling techniques have been used and there is a learner model used to represent students in the system). This might be needs, aspirations, preferences, competencies, interests, knowledge, priorities, social-cultural background, gender, age, profession, skills, hobbies, values and attitudes. It may also be personality and behavioral traits.
- Learner past activity. This may be the types of actions the learner decided to take at certain points in time in their attempts to deal with tasks.
- Frequency and intensity of support already provided. There are students that tend to ask for more support than others and that may reflect their attitude and not a real need for support. There are also students that feel intimidated and tend to ask for less support than what they actually need. The system should be able to detect those behaviours and prioritise accordingly.
- Level of support already provided. If support has already been provided to a certain level and more support is required, the system should be able to adjust the level of help to accommodate the learning need.
- Time the support is requested. There are cases where there is a time limit beyond which the learners are not allowed to continue. As the task progresses towards completion the system might have to adjust the level of support to enable the students finish in time. There are also cases where different types of support are adequate or suitable to different phases of the learning process. Time-related adjustments should be possible in an adaptive system.

Adaptable support is an essential feature of ITSs. The intention must be to provide the students with individualised help to the greatest possible extent so that they can safely diagnose their problems, understand their misconceptions and consequently embed the new concepts into their knowledge structures.

Layer 6 (L6) : Analytics

Learning analytics modules have become essential features of ITSs. The general aim of learning analytics is to provide learners and tutors with formative and summative feedback about the learning process. At that level of abstraction this layer may be confused with the intelligent support presented in layers 1-4. The difference is crucial though and it lies primarily in the intention. Intelligent support help aims to detect potential misconceptions and help students to overcome them. Analytics, on the other hand, are update reports that aim to increase student awareness about their current state in relation to what is required/expected by the learning environment. Students are supposed to utilise this information to continuously align themselves with the expectations in terms of how the environment should be utilised and the learning objectives that have been set.

Analytics may also be implemented for teachers and learning designers. Teachers can utilise formative feedback to make better and more well-informed decisions about how to intervene during the learning process. They can also utilise summative feedback to see what the students achieved, their issues during the process and prepare themselves accordingly for subsequent sessions. Learning designers are primarily interested in summative feedback. The intention in this case is to analyse student activity and system utilisation in relation to the learning objectives and performance in order to revisit the design and make evidence-based decisions on how to improve it. Areas of interest in these analyses may be learner trajectories, platform usage, volume and intensity of learning environment usage. In exploratory learning environments where constructivist approaches are highly encouraged it may also be expected to measure

the level of creativity the students exhibit.

Another aspect of this layer that is worth mentioning is the possibility to utilise intelligent support to enhance the analytical capability of the analyst. By analyst here we mean learner, tutor or designer. Information presented by analytics dashboards can be overwhelming and that makes interpretation of the results difficult, time consuming and possibly less actionable and effective. Components that help the analyst focus on the information of interest and recommendations about appropriate actions can reduce the cognitive load needed for the analyses and result in a more effective utilisation of the system.

3.7 Usability Testing

At that stage FLIP is a system that is readily available for testing and experimentation. The next step is to conduct a usability testing (Barnum, 2020) to verify hypotheses regarding the need for TI support. The exact objectives for this part follow:

- Test FLIP with real users and verify that it is usable
- Verify that there is an actual need for TI support in programming
- Verify that users are willing to take advantage of TI support provided in automated manner

3.7.1 Participants

FLIP was tested with a cohort of MSc students at the Department of Computer Science and Information Systems, Birkbeck, University of London. The prerequisite for this test was to have participants with some or little experience in programming and not a strong background in IT. Students of a particular conversion MSc were targeted based on the assumption that they fit this profile. These students did not have a strong background in computing and had already some exposure to programming but not much. This combined with the fact that we had relatively easy access to them made them good candidates. An open invitation was addressed to students with an interest to attend two free introductory practical sessions in JavaScript. Students joined this short course on a volunteer basis. The decision to follow a non-random sampling technique for the selection of participants is justified due to the fact that we were looking for a population with certain characteristics that we had access to. This is referred to as volunteer sampling in the literature (O’leary 2017).

3.7.2 Method

The purpose of the test was to verify that there is indeed a need for that type of support and that students are willing to utilise automated support to satisfy that need. Two sessions took place in June 2014. The material given to the students was a series of exercises with incremental difficulty split in two worksheets (one per session). These worksheets are given in appendix 4. The same cohort attended both sessions but participation dropped in the second session. The first one involved 13 students and the second one 5 students. The sessions were automatically recorded by FLIP and there were 4062 user activity indicators intercepted. The indicators have the following form:

ID: Timestamp

DATE: Date

USERID: Unique identifier for the user

ISSUER: [system|user] Entity generating the data

MISCONCEPTION: Unique identifier for the misconception identified

STATE: [activated|fired] State of the rule that generates support

MLEVEL: [1-5] Level of support provided

As the student experiments in FLIP the code is continuously monitored by the system and suspicious code patterns are identified and recorded in the form of these indicators. If the user utilises support on a specific issue, the corresponding rule gets executed and its consequent does the tutoring. If support for the same issue is requested more than once, then the level of support changes as well as the consequent. In this experiment we wanted to measure the level of user engagement with support related to common misconceptions. That is the identification of misconceptions in student code and the number of times a user utilised support for each identified misconception.

Table 3.3: Misconception Indicators

	users	indicators	indicators/user	1st level	2nd level	3rd level	L1%	L2%	L3%
session 1	13	1283	98.6	996	316	5	74.98	24.63	0.4
session 2	5	556	112.2	413	138	5	74.28	24.82	0.9

3.7.3 Results

These indicators revealed that 23 distinct problems (code patterns) were identified. After reviewing the data we concluded that 14 of those were important and related to common misconceptions in the CI. That was 1839 indicators in total. As depicted in the table that follows, the average number of indicators per user in both sessions was around 100 and only 3 out of 5 levels of support were used by the students. The statistics don't seem to deviate between sessions as in both cases level one support equates to approximately 75% of the indicators, level two support is around 25% and level three support is a bit less than 1%. The average level of support given by the system for these misconceptions was 1.25 out of 5 which is relatively small. That is probably because the users had already some programming experience and could resolve the problems in just one or two iterations without further help. The level of human support during those sessions was minimal and the students seemed not to have problems communicating with the automated tutor to overcome issues. The fact that students managed to complete their exercises with minimal human support in a language they had no experience of is an indication that the system worked. The fact that there was utilisation of the automated support clearly shows that the CI and the implemented rules are valid and address common student problems. It also shows that the students need that kind of support and are willing to utilise automated tutors to receive it.

3.7.4 Discussion

In conclusion, FLIP was used effectively by University students in two sessions. Students were able to accomplish the tasks given to them with minimal human guidance and support. Usage data was collected and analysed. Findings indicate that TI support

was used and learners were happy to direct their support requests to an automated tutor.

We consider this result a good first indicator that gives us a positive outcome regarding TI automated support in exploratory learning. Although this satisfies our objectives at that stage as it gives us a good result and enough insight for the parts that follow, we recognise that it requires further research in order to achieve a more credible outcome. The sampling method used for this experiment cannot satisfy representativeness. That makes the above results not generalizable as it is likely that the students who decided to join this experiment were somehow different than the ones that didn't (convenience sampling). Although sampling on a volunteer basis methodologically contradicts research credibility, it is an acceptable way to test a system that has very particular requirements (O'leary 2017). In our case there were specific selection criteria used that satisfy specific requirements for the test. This makes the outcome valid and transferable to cohorts with the same characteristics.

This step effectively concludes phase 1. The experiences and findings acquired throughout this phase strengthen our belief about what is needed to make teaching of programming easier. The WHAT is clearly a system that allows learning programming through exploration and provides TI and TD support. The equation takes now the following form:

"ELE with automated support" + HOW → "make teaching programming easier"



Completing the Working Principles

As mentioned in 3.7, at that point we are fairly confident that what we need in order to achieve the desired value is an ELE that teaches programming and offers automated TI and TD support. In the previous chapter the research findings along with the experience acquired through participatory observation and the development of a prototype gave us a very good insight about the problems in teaching and learning programming as well as the respective challenges at both a theoretical and a technical level. The outcome of the usability testing, although not generalisable, gave us a very useful indication of the applicability of this type of support in a real world educational context and the level of acceptance and usability of that type of support for a certain category of learners. This makes the WHAT component of the reasoning pattern more concrete. We tested a real system with real users and we learnt something from the process. This system was developed from scratch based on a domain analysis of existing systems and all the educational and technical challenges along the way gave us knowledge and insight about how to build such applications. Therefore, from the work already done we also have findings that give some substance to the HOW component as well. The layered architecture, the need for web-based design, componentization, reusability, interoperability and ultimately the need to find ways to make tutors in an easy and cost-effective way are learnings that we take away from phase 1 and we feed them back into phase 2 to help us discover the rest of the HOW component. The HOW component at that stage needs to be complemented with elements related to the provision of TD support in the context of exploratory learning systems. Therefore, the aim for this phase is to investigate this area further and find ways to tackle this type of support along with the technical aspects of componentization, reusability, interoperability and authoring of intelligent tutors.

The material presented in this chapter is supported by the following papers: (Karkalas, Mavrikis & Charlton 2015, Karkalas & Mavrikis 2016, Karkalas et al. 2016, Karkalas, Bokhove, Charlton & Mavrikis 2015).

4.1 Literature Review and Domain Analysis - An Outline

At that point we focus on the TD side of automated support. For this type of support there are several systems that have been developed in the past. These systems can provide us with a lot of material in terms of features, architectures, technical requirements, teaching and learning approaches. We need this information so that we can get more familiar with the area and see if there is room for improvement and innovation. The aim is to find out more about the HOW component of the reasoning pattern. That is how we can achieve the WHAT. The exact objectives for this part follow:

- Familiarisation with the area
- Identification of limitations, deficiencies and weaknesses of current approaches
- Identification of opportunities for improvement and innovation
- Identification of architectures, methods, techniques and technologies that may be reused and combined to remedy problems

To achieve all the above, the review was directed in three areas:

- Existing systems that offer automated support (automated tutors)
- Existing systems that offer exploratory learning
- Existing authoring systems for the development of automated tutors

The outcome of this review is that there are a lot of learning systems that offer opportunities for exploratory learning and abstractions to keep the learning curve as shallow as possible yet these systems lack automated support and adaptability. On the other hand there are Intelligent Tutoring Systems (ITSs) that utilise very sophisticated technologies to provide support and adaptability but they offer limited opportunities for exploration and the learning process typically takes place in a quite controllable manner.

On the authoring side, we could say that authoring tools for automated support are typically oriented towards guided learning. From a technical point of view, these systems seem to be tightly coupled with specific platforms and technologies and offer domain-specific solutions. In terms of usability these tools are intended for people with a high level of technical and domain expertise. That means that, in general, the entry threshold for end users is quite high and development of automated support is quite expensive. The findings of interest can be summarised as follows:

- There is very little work on authoring automated support for exploratory learning (refer to 2.2 and 2.4)
- There is no authoring tool that can develop domain-independent support for any learning environment (refer to 2.4)
- There is no authoring tool that can develop automated support for diverse web widgets that may not be compliant with integration and interoperability standards (refer to 2.4)
- There is work that simplifies the authoring process for guided learning that may be transferable to exploratory learning settings (CTAT) (refer to 2.4.6 and 2.4.7)
- There is work on systems architecture that simplifies the authoring process in exploratory learning settings (FRAME) (refer to 2.4.10)
- There is work on systems architecture that may be able to give answers to the dependency problems on specific platforms and domains (SEPIA) (refer to 2.5.2)

4.2 User Centric Design through a Requirements Elicitation Workshop

At that point we have enough information regarding previous work and we need a fresh look at the design considerations for building an authoring tool. All that is a significant step forward but we need the users' perspective to have a more holistic and user centric view of what is needed. Therefore, the next step is to conduct a requirements elicitation workshop (Millard et al. 1998). In this workshop (see 4.3.1) we used participatory design methods to elicit the challenges potential users would face in developing support for three well known exploratory learning environments. Participatory design is a well-accepted method for attempting to solve a complicated design problem with the active involvement of people from different backgrounds and different expertise (Vines et al. 2013, Bødker & Kyng 2018). We chose three different web-based environments that are quite diverse in nature and not compliant with any integration and interoperability standard. The participants were designers and educators with expert knowledge on the respective environments. They were given a scenario describing the context of the exercise so that they can understand how to position themselves as authors of automated support in that context. They were asked to design and develop a learning activity for their respective environment giving details about misconceptions expected, landmarks that can indicate important states of the constructions and the expected feedback. The outcome of this process was a number of use cases of specific learning activities developed in three well-known ELEs: GeoGebra, Malt+ and FractionsLab. The experts provided us with complete usage scenarios for each activity that include potential student misconceptions, landmarks and the respective feedback that the system is expected to provide to students.

The information collected from this workshop helped us form an initial idea of the development requirements both at high and low level. The high level requirements relate to how automated support is supposed to be authored and presented and what

is required in terms of analysis and reasoning so that the required level of support is produced. That in turn gave us insight about the requirements related to the data acquisition and processing so that we can have the required artifacts to do the reasoning. That led to the specification of the lower level requirements concerned with the technical components required to accommodate the composition of evidence. The high level requirements follow:

- compatibility with any web widget that may carry some educational value
- ability to intercept user actions and configure logging on the widgets
- ability to develop automated support based on the logs for widgets operating in open and exploratory settings
- ability to generate support based on rules and analysed evidence
- ability to make support available on demand in a non-intrusive manner
- ability to operate with limited technical and domain expertise

The following list gives us the low level requirements:

- need for methods to make the widgets generate and expose the required data
- need for methods to transfer this data between tiers
- need for methods to efficiently store that data and make it processable so that it can be used for answering queries to the reasoning part

Naturally, the low level requirements need to be addressed and satisfied first before we move on with the high level ones. Therefore, the decision at that stage is to focus on the second list.

These requirements seem to be quite reasonable especially in a context where there are many interoperable and diverse components involved. Even the authoring tool

itself could be thought of as a component and as such it should be able to integrate and interoperate with any learning environment in a loosely coupled manner as an external entity. In a context like this the architectural design should not be a concern for potential authors. Integration should be seamless and all the underlying technical details regarding APIs and data interchange should be hidden from the author. The author should not be concerned with how the data gets collected, transferred, stored and processed. All of this should be very simple and transparent. Having these things in mind we carry on with a technical spike in an attempt to resolve these issues.

4.3 Implementation of WIIL

The low level requirements identified in the previous step led to the development of a technique that can be used to overcome web component heterogeneity and achieve seamless integration and interoperability between any web component and its host environment. After a literature review on the subject the technique was designed and a prototype (Henson & Knezek 1991, Wong 1993) was implemented and named Web Integration and Interoperability layer (WIIL). It was developed to provide the means to combine web components with hosting platforms like learning platforms and authoring tools and increase reusability of both.

This part of the project concludes with a new technique that is designed, implemented and tested successfully. This technique enables seamless integration and unrestricted two-way communication between web components and their environment with minimal technical overhead. This output gives an answer to two problems: reusability of any web component that carries some educational value and reusability of web-based authoring tools with those environments. This gives us a more concrete picture of that HOW in the reasoning pattern.

4.4 The Web Integration and Interoperability Layer (WIIL)

This section presents a technique that enables integration and interoperability of web components with learning platforms. Traditionally, integration and interoperability is an issue related to merging third party learning components with Learning Management Systems (LMS). LMSs have been adopted and used for decades by large educational institutions like Universities. These institutions have heavily invested in them and consequently they have become heavily dependent on them. LMSs are nowadays being treated as large-scale enterprise-wide applications and considerations like stability and reliability are primary concerns in this context. A natural consequence of that is that integrating new functionality and instructional content became more difficult (Severance et al. 2010).

The need to integrate LMSs with third party components in a controllable way and maintain stability, security and reliability led to an architectural approach that offers the ability to decouple functionality into independent and self-sufficient components that interoperate via standardised communication protocols potentially over a network (González et al. 2009). That, in turn, led to the development of a new market for learning components which are typically self-sufficient, fully-fledged web-based applications equipped with their own security infrastructure.

The need for these applications to integrate with LMSs without sacrificing stability led to the development of standards like the IMS Learning Tools Interoperability (LTI) specification and OpenAjax (see 2.5). Strictly speaking OpenAjax is not specifically related to educational software applications but it has been used a lot in the EdTech domain. It is an industry initiative to enable interoperation of web components in the context of a web browser. These standards solved a problem but LMSs became too restrictive for educational innovation (Mott 2010).

Educators want the freedom to easily develop formations of components that are available on the web and make them part of their educational practice with little or

no configuration overhead in a way that resembles systems like Gurrum et al. (2008) which is a browser-based application composition environment and run-time system that simplifies development on top of existing complex systems. Systems like Personal Learning Environments (PLE) (Severance et al. 2008) or Personal Learning Networks (PLN) and Open Learning Network (OLN) (Mott 2010) seem to be more in line with what educational innovation needs as the logic behind them is radically different and promises greater flexibility, portability, adaptability and openness. In this section we propose a technique that is a lightweight alternative to IMS LTI and OpenAjax and is especially suited to simple client-side widgets that have no back-end dependencies and potential security risks. This is based on the premise that nowadays there is a wealth of widgets that are freely available on the web and could be utilised for educational purposes but existing integration and interoperability mechanisms are too stringent and restrictive to allow easy development of educational web mashup applications. The technique is called WIIL and provides learning content authors the ability to utilise any type of web component with minimal development and administrative overhead. It also promises robustness, better functionality than OpenAjax and efficiency.

4.4.1 Web Components

As explained above, the motivation for this work was to enable teachers and learning technologists to make use of simple web components that are available on the web to synthesise new and compelling learning activities. The technique can be used with any web component, regardless of whether the component was originally designed for educational purposes or not or the component is implemented as a simple web page or it is a fully-fledged web-based application. However, the technique is primarily intended for a certain category of components, specifically the ones that cannot be easily integrated with learning platforms using standardised techniques like IMS LTI. A typical candidate component has the following characteristics:

-
- It is either an individual widget or it is a part of a JavaScript library that offers a logically interrelated collection of tools.
 - It is freely available and no copyright or license issues abide. Potential users are free to execute, copy, amend and distribute the software.
 - It offers an API through which its functionality can be made available to the users. Through this API it is possible to load, initialise, get/set its state and intercept user/system interactions with it.
 - It executes solely in the browser and there are no dependencies on other components at the back-end. Back-end dependencies may be acceptable only if there is no need for administrative control and potential security risks.
 - It may include a visual part but that is not strictly necessary.
 - There is no registration requirement for using the component.
 - It is hosted in a public Content Delivery Network (CDN) or it is downloadable and able to be hosted locally.
 - It is not possible or feasible to amend its implementation to make it compatible with a potential host or extend it with some interoperability method remotely.

4.4.2 Design Considerations

The primary design objective of this work was to overcome component heterogeneity and minimise the technical support and administrative overhead for integration with a platform. The technique should be generic enough to accommodate any widget regardless of its design idiosyncrasies. It should be equally easy to integrate components that implement the standard W3C widget interface and components that come with non-standard widget-specific interfaces. The technique should impose no constraints

on the type of interface the component supports, and of course, the component's interface should not determine in any way the functionality that the component is able to expose after the integration.

A secondary design objective was runtime efficiency in terms of responsiveness and resource utilisation. The method should be able to support efficient two-way communication between the hosting platform and the component without the need for expensive round trips to a back-end server which is always the case with LTI. Even widgets that have no back-end dependencies have to implement server components in order to become LTI-compliant. Back-end dependencies require round trips between the tiers and that introduces network traffic and latency. Adding artificial and unnecessary dependencies on the network and the back end is problematic if the components to integrate are simple front-end widgets. Efficiency is also related to the memory footprint and the processing requirements of the technique. This becomes even more important if we consider that the deployment platform is inherently poor in resources - the browser. Therefore, implementation should be lightweight enough in order not to burden the browser excessively.

A third consideration was cross-component communication. The approach we followed was that cross-component communication should be safe and guarantee the security and the integrity of the interoperating components but at the same time it should not be artificially constrained which is typically the case with OpenAjax. It should be up to the designer / implementer to decide what functionality is exposed from component interfaces and how it can be used by the rest of the system. Issues like component packaging, deployment, description and general architectural issues regarding distribution of learning widgets for the web (Wilson et al. 2007, 2008) were not considered in this design. The considerations that played a crucial role in this part follow:

- **Component heterogeneity:** Nowadays the plethora of components in the web

is overwhelming. These components are disparate and heterogeneous and their exposed APIs are always dissimilar. Integration with a platform requires a technique that is generic and independent of widget-specific functionality. The method should be able to overcome variability by providing a very simple interface, usable by any type of component.

- **Platform compliance:** Another consideration is the potential to re-use the method in future platforms as well. The method should not be dependent on platform-specific functionalities and idiosyncrasies.
- **Registration:** The integration process should not require the execution of protracted and cumbersome procedures. It should be possible to register the component with the platform with minimal effort and technical expertise.
- **Communication:** Once the component is embedded in the platform, it should be relatively easy and inexpensive (in terms of resource utilisation and complexity) to exchange messages with its host. Passing messages should be based on a connectionless communication protocol and rely on system stability at both ends of the channel.
- **Roles:** A crucial question is whether it is acceptable to consider the host (platform) and the guest (component) nodes as two equal entities in this relationship. In this case, implementation is simple and can be used globally by both sides in the same way. In the case that host and guest need to be treated unequal, the method should be based on the assumption that there are certain host and guest-specific functionalities that must be implemented. If this is deemed unnecessary, it obviously must be avoided.
- **Browser security restrictions:** Modern browsers are not very tolerant with pages that intermix content from different domains and most web components will

most likely originate from foreign domains. The method should be able to overcome security constraints and browser specific idiosyncrasies.

- **Performance:** Building a system as a dynamic and arbitrary collection of heterogeneous components, implies that these entities have their own space and distinct purpose in the system. It makes sense for these components to be able to operate in parallel and communicate asynchronously with the platform. In multi-processor systems this is not just a matter of asynchronous behaviour, but it can also make a huge difference in the overall performance of the application.
- **Memory:** Memory footprint is becoming a serious issue in browser-based fat-client implementations. The interoperability part must not be a substantial burden in the memory balance.
- **Security:** Security is always a major issue when integrating foreign and potentially non-trusted components with a system. The tendency is to create integration methods with artificial barriers in order to prevent developers from making dangerous mistakes. This approach obviously may have a major impact on the functionality that is eventually exposed and reused. The method in this project should allow for maximum flexibility in terms of what is exposed and what is not. It should allow both secure containment of unsafe material and unrestricted exposure of data and operations wherever needed. It should be up to the designer/developer to decide what is secure and what is not.

4.4.3 Browser Security

One of the design considerations mentioned above that deserves some clarification and analysis is browser security. Nowadays, one of the major issues in web development is security and since web browsers are the natural deployment platform for this type of development special attention has been given to the way content and functionality

operates in them. Traditionally, one of the major threats in web development has been the mixing of content coming from different domains. Allowing a foreign component to take control of the browser or even other foreign components hosted in the same application has been a major concern in web development. That led to the development of stringent rules and conditions under which foreign components must behave and operate in the browser. One of these rules is the so-called Same-Origin Policy (SOP) enforced by all modern browsers. This is a policy that aims to prevent unauthorised access to confidential information by malicious scripts and thus protect data integrity. Intermixing learning components - objects from different domains to develop mashup educational applications is like trying to contravene this policy. The question is how can that be done in a way that is appropriate from a technical point of view and does not violate what the policy is supposed to prevent. One other consideration here is that the method must not be browser dependent. Although browser vendors are trying to adhere to standards and they design and implement browsers accordingly, there are many cases where browsers exhibit non-standard behaviours that are difficult to predict and accommodate.

A very common method that is being used a lot by web developers is to simply reference the components directly as JavaScript libraries in the host page. The components in this case inherit the domain of the containing page and that alleviates the SOP problem without hassle. This way, without any elaborate integration mechanisms, components can operate under the same origin in the same space and have direct access to each other. Direct and unrestricted access, though, may not be safe or desirable. Mixed content coming from both secured (HTTPS) and unsecured (HTTP) origins is another problem with this technique. The browser may not allow that. The actual behaviour is browser-specific and that makes the approach somewhat problematic. Another problem is that participating components, regardless of origin, will operate under the same context as a single-threaded application since JavaScript engines follow the single-threaded execution model. From a performance viewpoint this is obviously

not desirable. Another important consideration is code organisation. In this approach, code coming from different sources, gets executed in the same global namespace and that is potentially risky. Implementers of these components may not be aware of each others' design and coding assumptions and practices. Accidental name clashes that invalidate operations and data are very common consequences of that method.

Another method for performing cross-origin requests is JavaScript Object Notation with Padding (JSONP). This method is essentially based on the same premise as the previous one. Instead of making an HTTP request to retrieve the code of the foreign web component we integrate it as a script. JavaScript scripts inherit the same origin as the enclosing page and therefore code or data received as a result of this request automatically becomes an integral part of the receiver. The difference between this method and the previous one is that it is presumed that the service to be integrated is JSONP-aware and that implies that this method presupposes a great deal of control over the component source and JSONP-aware services. This violates one of the design considerations mentioned in 4.4.2 and, in addition to that, the method suffers from the same problems mentioned above.

A very clean way to circumvent the SOP check is to not make any cross-origin requests at all in the browser. Instead, the requests can be made by another logical tier that operates elsewhere. SOP restrictions do not apply on server-side components. Requests are sent from the browser to a server-side proxy that has the same origin as the page. The proxy handles the interaction with the services and sends back the results to the page. This approach is much cleaner than the previous ones but the downside is that it requires an additional server-side component. That introduces unnecessary complexity, network load, latency and dependencies. According to the design considerations presented in 4.4.2 this is not desirable. Furthermore, as before, the code coming from different sources ends up being executed in the same context and that suffers from the same problems mentioned above.

A fourth alternative to the above is Cross-origin Resource Sharing (CORS). CORS

assumes that we have access to the server side part of the component and we can configure the HTTP response with a header that instructs the browser to ignore the SOP. This is obviously not possible with components that we cannot download and host locally in our own servers. Additionally, even if we do have access to the server, there is no guarantee that a firewall will not remove the additional header. Finally, browser incompatibility may be an additional problem. Another alternative is to not use HTTP at all and use websockets instead. SOP is only applicable in HTTP and that makes websockets an attractive solution. In this scenario it is the websocket server that does the security checks and allows the caller to receive the content. However, that seems to open the door to SOP violations and we expect that it will be resolved in future browser implementations. Therefore, it may not be a viable, long-term, solution. Furthermore, it assumes access and administrative control of the server side part of the component which may not be possible in most of the cases.

The only method that seems to overcome all of the above issues is to embed the component to the page as an external page in a sandbox. In practical terms this can be done in HTML using an IFRAME element that encapsulates a separate page containing the external component. The component is kept isolated in a sandbox and executes in its own context as a separate application that operates in parallel with the enclosing page and other components. That satisfies both the requirement of code safety and performance - parallel execution. The component can interoperate with its host through a messaging system that is inherently supported by HTML5 (Järvinen 2011). Components and the hosting page can operate concurrently and communicate with each other through bi-directional asynchronous message passing. In this scenario execution and data interchange takes place entirely in the browser and there is no network and server overhead involved.

4.4.4 The Technique

The technique proposed can be logically divided into two parts: node interfacing and communication protocol. We use the term node as a generic term to refer to the components to be integrated. Since integration in this context involves a platform and its external components we also use the terms host and guest to respectively identify both. The latter naming convention does not carry different semantics in terms of implementation. The nodes are treated as equals and two-way unrestricted communication between them is assumed. Conceptually, each part becomes more abstract as it is encapsulated within a wrapper layer that decouples its interface from its internal specifics. The wrappers provide the ability to selectively externalise the node functionality in a way that is generic - not node-specific. Node functions are exposed through the definition of a public interface that maps internal implementations to publicly available methods. These methods can then be callable by the communicating parties through message passing. Message exchange is facilitated by the wrappers that alleviate the SOP problem and provide a seamless integration mechanism.

Component Registration and Public API

From an architectural point of view the wrapper is just a software component that encapsulates another software component with the intention to extend it, reduce it, change it and impersonate it. In software engineering there is a well known OOP technique that is being used to associate classes with objects and other classes that is called composition. This exact paradigm is being used as the architectural basis for the wrapper.

In more practical terms, the wrapper is expected to be a JavaScript library that is referenced by the two components to be integrated. This library can be either local or external. The very same library is referenced by both, regardless of whether the component is a host or a guest component. As explained in 4.4.4 both parts are treated as

equals so implementation of the wrapper is always the same. The library materialises an instance factory that can be used to instantiate a singleton object to represent the wrapper. The instance factory handles requests of wrappers. Every time it is asked for one it checks whether the singleton exists and returns it if it does. If not it creates it first. The factory makes sure that only a single wrapper exists per node and that keeps things simple and easily manageable. Once the wrapper is instantiated, it can be used by the implementer to declare the node and its interface. The implementer is typically a learning designer with technical skills and enough knowledge at a technical level to understand how the node is organised internally in order to register it, declare its public interface and make decisions about the data format for inbound and outbound requests. Integration entails two registrations. The host must register its guest and respectively the guest must register its host. This way both components know about each other and establish a trust relationship between them. Without this trust, communication between them will not be possible. Depending on the case the integration might be performed by the same person or two different implementers. This depends on whether the implementer has access to the source code of both systems. If one of them resides on a remote and inaccessible server and migration is not possible, then the only way to perform integration at that end is to get somebody with access to that system to do it. Integration scenarios vary depending primarily on the level of sophistication of the guest component and the nature of its implementation. If the component is implemented as a library then the implementer will have to turn it into a web page first so that this page can then be referenced by the host's iframe. If it is already released in the form of a web page then the only action required is to add a reference of the WIIL library to this page and add a few lines of JavaScript code to perform the registration. Registration is performed as part of the component initialisation process where the component registers itself with the local instance of WIIL, declares its public interface and registers a trusted peer component along with the domain it belongs to. The exact same process must be performed at both ends for the integration to be

successful. In most cases there is only one person that performs integration at both ends because web components are typically plain JavaScript objects with known and accessible APIs that can easily be embedded into web pages. Once both registrations are successful the two components can start exchanging messages.

The wrapper provides a very basic API for those operations:

- `setComponentID(componentID)`
- `register(peerName, peerComponent)`
- `setForeignDomain(domain)`
- `pushFunction(functionName, functionRef)`

The first one can be used to register the component with the WIIL wrapper. The argument is just an identifier given as a text string.

The second one can be used to register a peer component by associating an identifier with a reference of the object holding the component.

The third one can be used to inform the wrapper about the foreign domain that needs to be trusted.

The last one can be used to define the component's public API. Typically, the latter is used multiple times to declare public methods and associate them with the component's internal API.

The following code snippet is a typical example of how this API can be used to register a component with WIIL and expose its public API:

```
//This statement uses the publicly available instance factory named
//Wiil to create an instance of a wrapper object
var wiil = Wiil.getInstance();

//This informs the wrapper that the component it will be handling
//is called 'malt'.
var componentID = "malt";
wiil.setComponentID(componentID);

//The following statements register the enclosing page (host) as
//peer with name 'host' and informs about the trusted domain
var peerName = "host";
wiil.register(false, peerName, parent);
wiil.setForeignDomain('somedomain.com');

//This is the public interface of the component

//This method is used to initialise the state of the component
//state is an object holding all the necessary info to initialise
//the component
wiil.pushFunction
(
    "setState",
    function(state) {
        if (state.isEmpty()) {
            maltWidget.resetState();
        }
        else {
```

```
        maltWidget.setState(state);
    }
}
);

//This method returns the state of the component instance as an
//object
wiil.pushFunction
(
    "getState",
    function() {
        return maltWidget.getState();
    }
);

//Finally, this method returns an object that shows the types of
//elements we can have in a malt instance and the events they can be
//associated with
wiil.pushFunction
(
    "getMetaData",
    function()
    {
        return maltWidget.getEvents();
    }
);
```

As shown above, the public methods refer to the component's internal API to actually perform the operations. This allows us to selectively expose the portion of the API that is needed for the purposes of the application we are building. It also allows us to semantically enhance this API or transform it to accommodate security or other requirements imposed by the design. What is also important to understand here is that there is no restriction as to what we expose. There is no standardised interface in terms of the component itself. This provides flexibility as we accommodate and facilitate component diversity so that we can take advantage of whatever functionality may be available and useful in components. Promoting diversity does not necessarily mean heterogeneity. Nodes interface with each other through a very simple and standard API and that means that all interoperating parts appear to be homogenous in the platform in that respect. The API of the wrapper is not the same as the API of the enclosed component that is exposed through the wrapper. This is explained in more detail in section 4.4.4.

Security

As indicated in section 4.4.3 the most suitable approach to integration is to use sandboxes to isolate implementations of foreign components and make them interoperable with the platform through wrappers that facilitate interfacing and communication. In a browser context sandboxes are typically implemented as IFRAME elements. As these elements may contain foreign code the SOP restriction is also applicable to them and that means that somehow this needs to be addressed and handled. In this scenario it all comes down to trust. If both interoperating components trust each other then message passing is possible. In terms of implementation, there are two ways to do this.

The first approach is to explicitly declare that the nodes have the same origin. This can be done programmatically by setting the property **document.domain** to the same domain name at both ends. A system called Subspace (Jackson & Wang 2007) is using

this technique to implement cross-domain communication for web mashup applications. In this system communication takes the form of a closure exchange between nodes. A closure is a function that remembers its lexical context or otherwise the environment it was created in. If it is called in a foreign environment - another node, it may still have access to the resources it remembers. Unfortunately, the actual behaviour of this method depends on the browser and that makes this approach potentially problematic. After experimenting with this we have also established that resetting the domain property may not have the expected result as the port number may implicitly be set to null (empty) by that process. This behaviour is again browser-dependent and that makes this approach unacceptable.

The second approach is to make the interoperating parts aware of each other so that each one checks the legitimacy of requests received before executing something or responding to the sender. This is the approach selected as it satisfies the design considerations mentioned in 4.4.2 and does not violate in any way the security restrictions imposed by SOP. The wrapper should provide the ability for a node to register the nodes it trusts so that communication between them can be allowed. Component registration is just another operation that is implemented as part of the wrapper's functionality. It is perfectly possible for this operation to be externalised as part of the public interface of the node. That opens the door to remote registration of components. In other words a foreign component should be able to register itself with a node as a trusted component without the implementer of this component knowing about it. This potentially facilitates easy integration but is obviously insecure, unless there is a list of known domains registered with the wrapper that can be used to restrict access to unknown and untrusted components.

Node Interfacing

As mentioned in section 4.4.4 component diversity is allowed but hidden within the wrapper. That provides simplicity in terms of node interfacing and at the same time promotes component diversity. The wrapper provides a very generic interface through which basic communication can be carried out. The interface comprises the following two methods:

- `sendMessage(message)`
- `receiveMessage(event)`

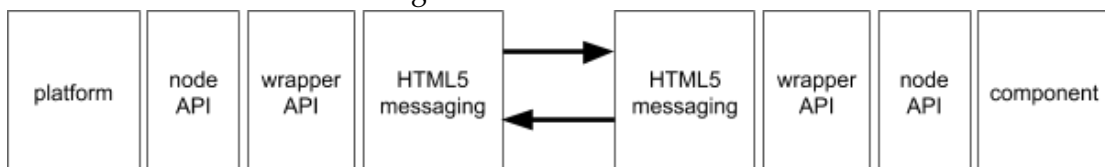
This simple system allows bidirectional communication between the nodes. Requests and data are passed to wrappers in the form of a **message** object. This message gets constructed and sent by the sender as an object. The receiver gets notified about the message with an **event** that carries it as one of its properties. Message passing in the HTML5 system is carried out using events. WIIL is using a standard format for the messages and provides a public method that can be used to construct one according to a template. The format follows:

- **origin**: This property serves as the unique identifier of the component that sends the message. It is not the same as the homonymous property of the event object that carries it when in transit. The latter corresponds to the domain of the sender. This property is just some text that uniquely identifies the component in the system.
- **content**: This property can contain data of any type. The purpose in this case is to send some data and let the receiver decide what to do with it.
- **command**: This is an instruction (command) that is possibly sent along with some data in the form of arguments. The intention in this case is to utilise

receiver-specific functionality and perform some processing there. This functionality is exposed in the form of a public interface that the receiver makes available to other communicating parties through the wrapper. The command can only be given as a string (text).

- **args:** This is an array of values that accompany the command. These values are addressed to a method of the receiver's public interface. That method is what the command property refers to.
- **callback:** This is a string value (text) that corresponds to a function exposed in the public interface of the sender. The receiver, upon receipt of the message, performs the requested operation and then sends the resulting value as argument to the callback function of the sender. This property permits the asynchronous continuation of the same logical process in the sender after a remote call in the receiver is completed.

Figure 4.1: The WIIL stack



The method exposed by the wrapper to create a message is:

```
createMessage(content, command, args, callback)
```

The format of messages allows nodes to exchange information and pass it to one another for internal consumption or send instructions for remote execution of functions. If callback is provided then the result of remote processing can be sent back to the sender of the request for further processing. Interoperating parts are secure because they don't have direct access to each other's functionality. Direct references to functions are not available. Processing requests are sent in the form of text and get evalu-

ated by the wrapper before execution. The receiving wrapper uses this text to identify the actual function that needs to be called and request the operation. That provides a level of control that allows the integrator to decide what should be available and how much scrutiny is needed before execution.

Since the format of the message is fixed, there has to be some validation in place in case the interoperating parts violate it.

A simple set of validation rules apply. A message object is not valid if:

- command is not a string
- args is not an array
- callback is not a string
- both content and command are not given
- args is given without a command
- callback is given without a command

These rules are enforced by wrappers by default. As explained above components are expected to be very diverse and therefore their public APIs exposed through wrappers are expected to be diverse as well. On the other hand platform - widget interoperability should be based on a standardised and uniform way of communication. Using message objects to enable interprocess communication satisfies this requirement. This technique allows us to keep a simple and uniform interface that facilitates basic communication for any type of node (component or platform) and at the same time we accommodate the utilisation of node-specific functionalities without constraints. Once a message is received by a node, WIIL internally processes the request. If it is a request

to perform an operation and not just a message that needs to be logged, it passes it on to a special executor method that selects the appropriate method from the component's public API to execute. WIIL is also equipped with an alternative executor method that is intended to be used internally by the integrator. If the integrator wants to utilise the component's public API from inside in order to avoid accessing directly the actual API, the following method can provide direct access - not through messaging:

```
executeLocalCall(id, args)
```

This way WIIL can also be used as an API management service that provides a standardised interface to components usable locally or remotely.

Communication Protocol

There is no standard communication protocol proposed for interoperating components. The specifics and the complexity of the protocol used depend on the particular use case and problem we are trying to accommodate. We quote a list of four communication scenarios that illustrate how a communication protocol could be used:

- a. The guest component wants to inform the host platform that it is properly initialised, fully functional and available.

```
content: 'ready'  
command: null  
args: null  
callback: null
```

This scenario assumes that once the guest becomes fully functional, an uninterruptible communication channel based on HTML5 messaging becomes available. If the hosting platform knows that the guest component exists and is available, then it is safe to

assume that the component will remain available throughout the whole session. In reality, of course, that may not be true because it is possible that the guest crashes or the element holding the sandbox is removed from the host's DOM for some reason. The communication protocol could be used in a more connection oriented manner in this case and check for availability at certain time intervals to ensure stability and robustness.

b. The component simply wants to send a message to the other party. There is no instruction for processing through a method call in this case. It is entirely up to the receiver component to decide what to do with the information received. The message format is exactly the same as before. The difference of course is that the previous content carries special semantics that may be part of the protocol.

```
content: 'some content'  
command: null  
args: null  
callback: null
```

c. The component wants to utilise a service provided by the other party. The request is to execute a remote method that gives access to the service. The resulting value may be required to be passed back to the caller.

```
content: null  
command: 'add'  
args: [2,3,4]  
callback: 'display'
```

In this case the caller asks the callee to execute the method **add** and perform addition of three integer values. The result is expected to be sent back using another message

as an argument to the method call 'display' which is going to be executed by the caller.

d. The component wants to send some data and instruct the other party explicitly what to do with it. In this case there is no callback as the sender is not expecting anything back:

```
content: null
command: 'logActions'
args: [{action1},{action1},...,{actionN}]
callback: null
```

The above scenarios illustrate simple uses of the message object to accommodate different communication requirements. The frequency of message exchange and the semantics of the values being exchanged may be used to define more complex communication scenarios to accommodate different needs.

Component Installation

Third-party components are considered external to an application and therefore an installation is typically required prior to their use. This type of process can take many forms in web-based applications. The physical component may be needed for local installation. In Moodle, for example, plug-ins must physically become part of the application codebase. Another case is to get a reference to a component that is hosted externally which means that it resides in a remote machine. If this component is LTI-compliant, there is a registration process that provides configuration parameters and a method of authentication, typically OAuth. Configuration parameters include the URL referencing the component, the user credentials under which a trust relationship can be established and launch instructions. User credentials are a consumer key and shared secret in Moodle. WIIL is designed to work with components hosted remotely

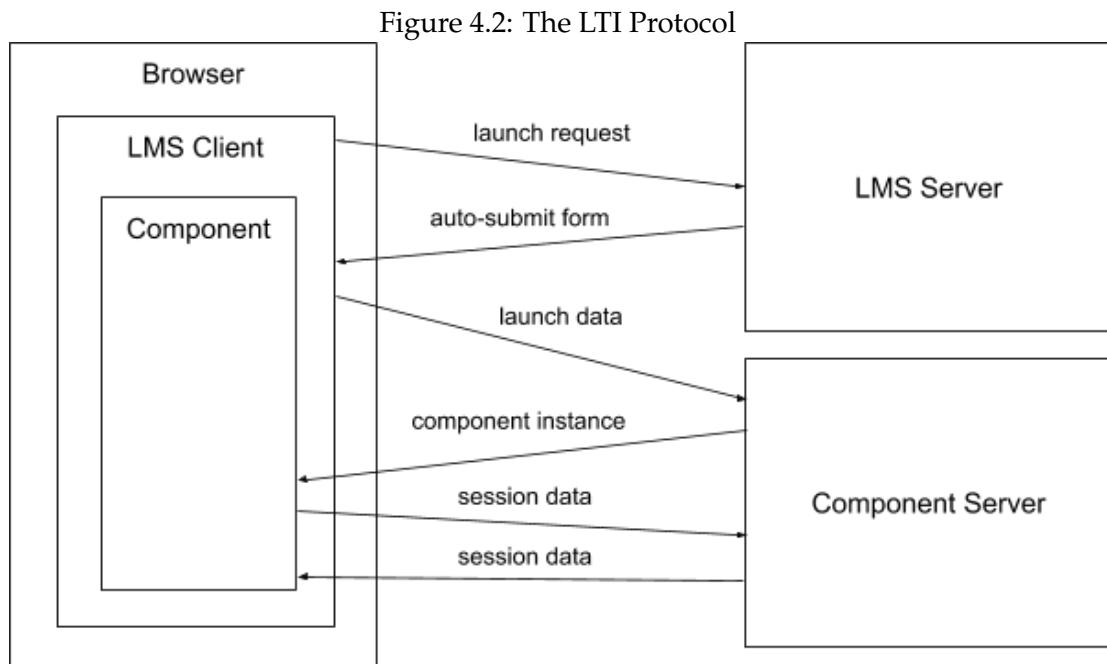
that are not LTI-compliant. In this context installation is much simpler and registration is more lax since authentication is not required. A mutual trust relationship can be established by injecting the trusted foreign domain names to the nodes using the wrappers.

The Launch Protocol

One of the things we are trying to avoid with WIIL is the unnecessary steps required for component launch in the LTI world. This is based on the premise that not all components are the same. Treating simple client-based web components as fully fledged web-based applications is like introducing artificial complexities that hinder the integration process with no apparent benefit. A comparison between the LTI launch process and the WIIL launch process will clarify things. The LTI approach assumes that there is always some back-end functionality implemented as part of the component to be integrated. The process follows:

1. The component to be integrated is selected by the user in the LMS client environment and a launch request is sent to the LMS server.
2. The LMS server knows what information is required for the particular component and prepares a message as an HTML form. This is typically information regarding authorisation to use the component combined with initialisation settings. This form is then sent back to the LMS client.
3. As soon as the form is received by the LMS client, it gets automatically submitted to the server-side part of the component to be integrated.
4. If all the information required is there, the component provider sends back to the LMS client an instance of the component. This is the client-side part of the component that is pre-populated with all the necessary values to instantiate and initialise itself in the LMS environment.

5. Session information is maintained through messages being passed via cookies during server round trips.

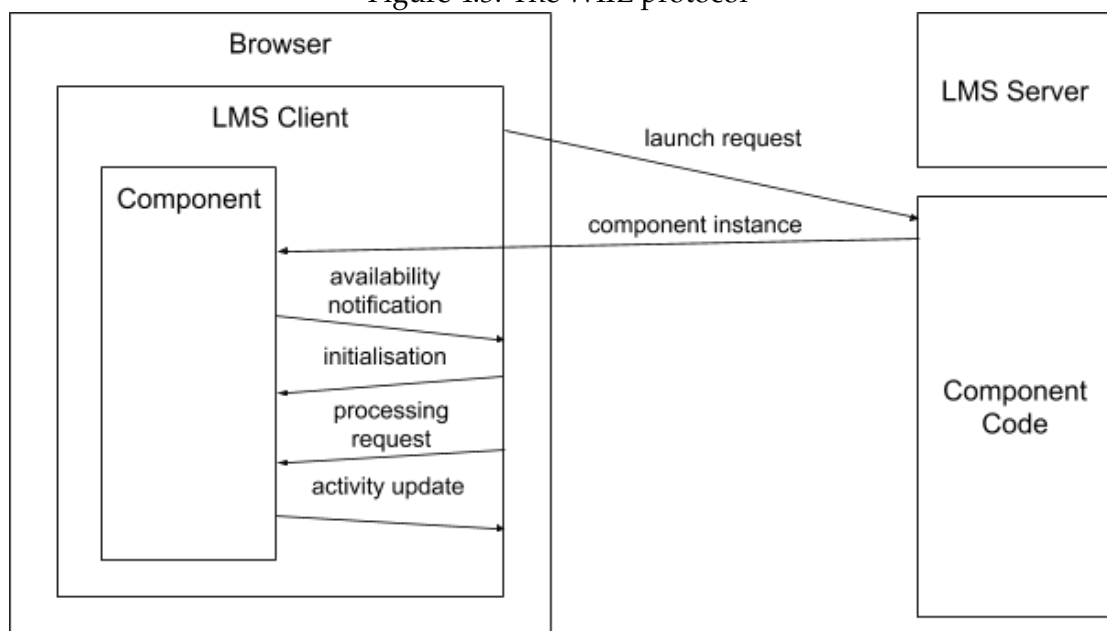


This launch protocol is obviously a very long-winded process for simple components that don't include any native server-side logic to process authentication, authorisation and initialisation information. According to LTI, in any case, this information must be submitted as a form through a HTTP POST request to a back-end service. That implies that there has to be some component at the back end to receive and process the request before releasing the code for the component. For simple web widgets with no server-side code that means the deployment of an extra processing tier to play the role of the server and unnecessary network traffic between tiers. All that introduces extra cost in terms of complexity, latency and resource utilisation. In the WIIL world things are much simpler. An alternative launch protocol follows:

1. The component to be integrated is selected by the user in the LMS client environment and a request to get its code is sent to the hosting server.

2. The code is released by the server and the component gets loaded and instantiated within a sandbox (guest) in the LMS client environment.
3. Once the component is available and fully functional it notifies the LMS client that it is ready to be used.
4. The LMS client sends a message with initialisation information to the component and the component gets initialised.
5. Subsequent correspondence between the LMS client and the component is local and may involve processing requests and activity updates.

Figure 4.3: The WIIL protocol



This launch protocol does not involve any server-side processing because there is nothing to be processed. The only thing that may be processed is initialisation data that is sent directly to the component at the front end. Once the component is instantiated at the client side it starts operating in a disconnected mode as if it were an integral part of the original platform. All subsequent communication is local and takes place directly

between the component and the enclosing platform. Network traffic and latency is minimal, processing is faster and responsiveness is better. Implementation is simpler and much faster.

Cross-component Communication

Typically the framework used for this part does not deviate a lot from the basic principles of OpenAjax. In the OpenAjax world cross-component communication takes place through managed or unmanaged hubs. A component may take the role of a producer that publishes messages to the hub and/or a consumer that subscribes to receive messages from the hub. The hub is designed around the concept of anonymous broadcasting. Producers and consumers are not aware of each other. Point-to-point messaging, cross-component property management and remote procedure calls are not inherently supported. In WIIL communication between components is possible only through the platform's wrapper. In that respect this wrapper plays the role of a hub. Components live in their own secure environment (sandbox) and exchange information with the platform through message passing but this is where the similarities end.

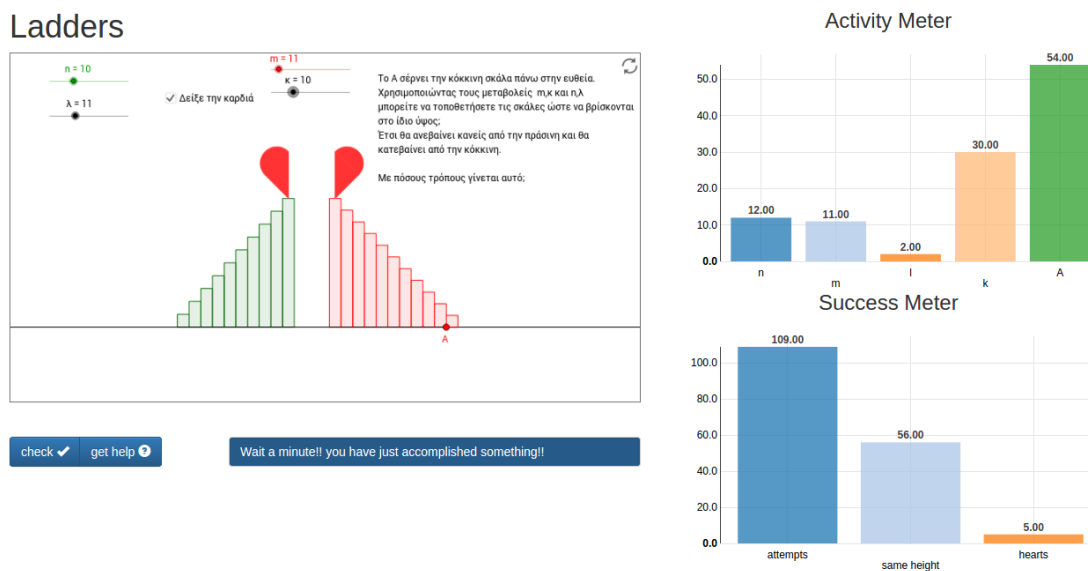
An important difference in WIIL is that the platform itself is a component. In this case communication is direct and unrestricted. System integrators are allowed to expose a widget functionality (or part of it) and make it available to its host and vice versa. Components are able to exchange messages and to make remote procedure calls. Property management is also possible through the same mechanism. Communicating parties are fully aware of each other's exposed functionality and are free to utilise it. A distinguishing feature of this system is the ability to perform asynchronous operations in the sender after a remote procedure call is completed via callbacks.

4.4.5 The Ladders Activity - A PoC

In this section we present a sample web-based application that shows how different web components can be integrated with a platform and interoperate using WIIL. A learning activity has been developed using Geogebra and uploaded to Geogebra Tube with the name "Ladders"¹. Geogebra Tube allows sharing and delivery of the component through a URL. Referencing this link from a web page instantiates the component and exposes its API to the page. Once the component is instantiated user activity can be intercepted through events and made available to the host platform through WIIL. If the host wants to be notified every time the user changes something in the construction it exposes a method to be called by the guest every time the corresponding event handler of the widget is invoked. In this activity the student uses the sliders to change values in variables. User activity data along with the current state of the construction are sent through the wrappers to the host. Once the data is received the wrapper transparently handles unpackaging of the content and delivers the data directly to the requested method. The host inserts the data into a local in-memory JavaScript database that is linked to visualisations on the page. As the student interacts with the tool and the data changes in the database, the host displays real-time user activity and performance statistics in histograms. The visualisations are themselves separate components hosted in their own guest sandboxes and communicate with the host using the same mechanism. The platform is also equipped with a rule-based expert system that provides real-time intelligent support to the student. If the student does something that seems to be in the right direction the host platform displays a message to reinforce this action. The same system can also be used by the student to request help during the activity and check whether the objective has been accomplished or not. The platform is merely a coordinator in this process and all the functionality becomes available through WIIL from the underlying components.

¹<https://www.geogebra.org/m/s48ThHqF>

Figure 4.4: The Ladders Interface



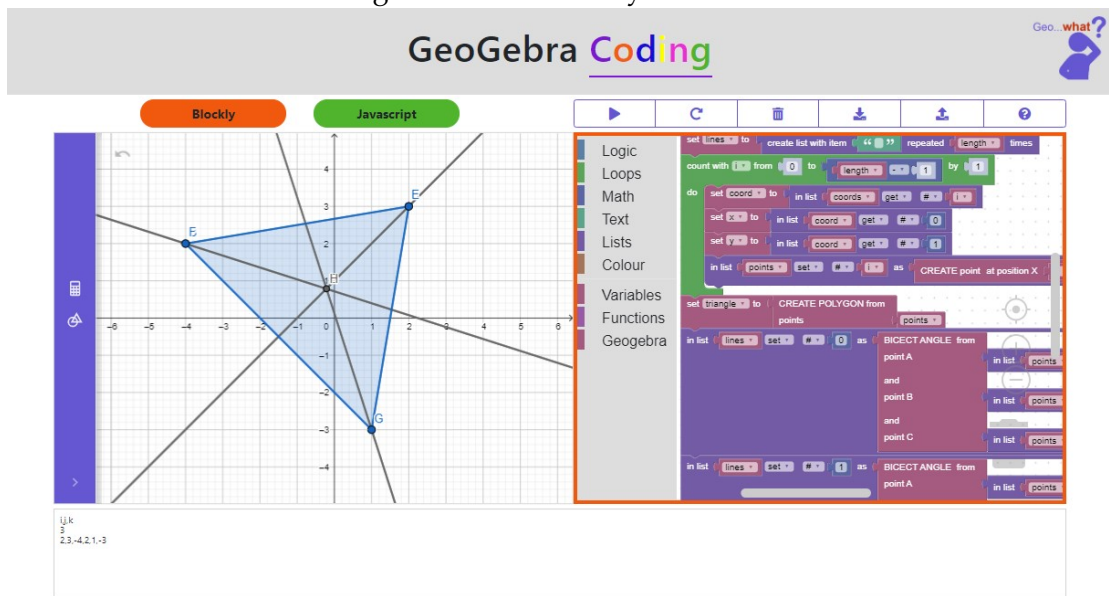
4.4.6 GeoGebra Coding - A PoC

In this section we demonstrate a different integration scenario. Again, we use a GeoGebra component, but this time the component lives natively in the hosting page. In this case it is not GeoGebra Tube that is being used as a factory of activities. GeoGebra Coding² is a fully fledged learning environment designed to help people learn programming. It features a GeoGebra component that exposes its functionality through a custom JavaScript API and a Blockly component that provides full JavaScript functionality as well as a custom category of blocks to facilitate operations in GeoGebra. Learners are given the option to use either JavaScript or Blockly to code their projects and GeoGebra in this context is used as a microworld or creative playground for learners that want to learn programming through experimentation with geometry. The GeoGebra Coding widget exposes its own public API through WIIL. The hosting platform is able to communicate with it and retrieve information about elements, events and operations that take place in the environment. The current student state can be exported or imported in the form of XML or JSON. Indicators of student activity notify the host

²<http://geogebraworld.s3-website-eu-west-1.amazonaws.com/>

every time something changes in GeoGebra or the code editors. The host receives this data through its own methods that have been exposed via WIIL. The data is inserted into a local in-memory database that lives in the browser. It is then processed by the platform to provide basic support in learning activities on demand.

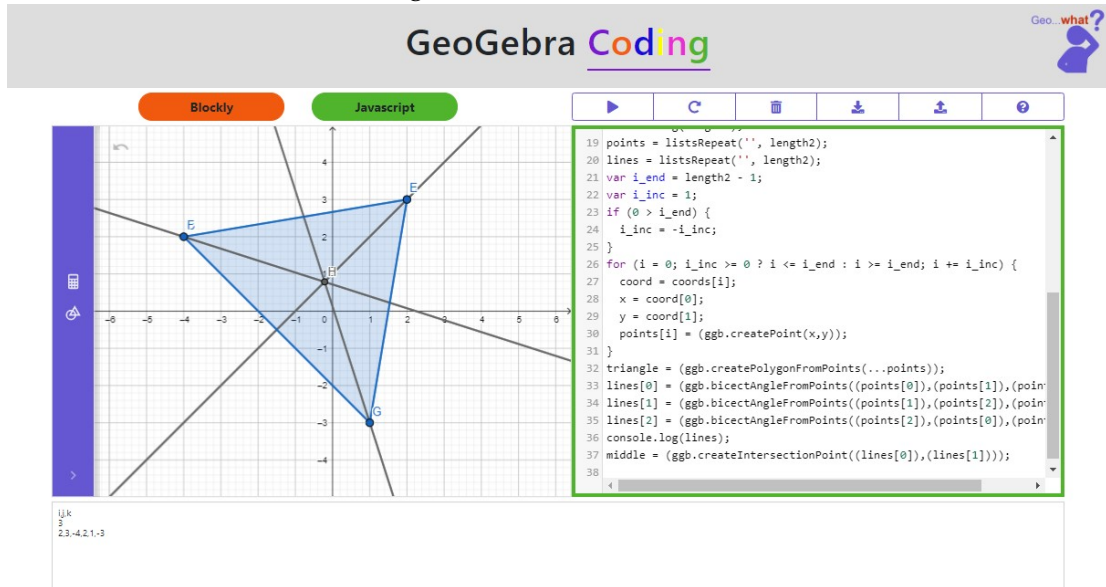
Figure 4.5: The Blockly Interface



4.4.7 Results

Test results with various experimental platforms showed that the method works as expected and fulfils its original design goals. The method is able to effectively deal with component heterogeneity and seamlessly integrates disparate components into a seemingly homogeneous whole. Registration, instantiation and initialisation of these components is simple and efficient. Operation is safe and the system performs well when the components asynchronously communicate with the platform. The method overcomes browser security restrictions and the overhead in terms of memory and processing power needed is minimal. It is estimated that the experimental implementation has successfully processed approximately 37,000 events so far. Sample tests

Figure 4.6: The JS Interface



showed that messages are being exchanged with 0% loss at a speed that allows a very smooth interaction between different components. In future versions of the system we envisage to implement an on-line editor that simplifies the integration process by inserting wrapper boilerplate code to the nodes and by providing the ability to visually manipulate them.

4.5 Learning Environment vs Platform

Many times the terms environment and platform are being used interchangeably in the learning domain. This is a potential cause of confusion and thus we need to distinguish between the two and clarify the differences. We can think of a learning environment as a virtual space within which learners perform actions with the intention to achieve learning. eXpresser³, Geogebra⁴, Scratch⁵. are examples of learning environments. Typically, a learning environment specialises in a specific domain of knowledge and utilises certain environment-specific approaches, tools and interfaces to achieve its purpose. A learning platform, on the other hand, is a system that hosts one or more learning environments and other components or services that potentially interoperate with each other with the intention to support learning. A learning platform in this respect can be thought of as the infrastructure that facilitates the integration and interoperation of these components so that their synthesis can achieve its purpose. In practical terms a learning platform covers a very broad range of systems that may vary in terms of integration, or other architectural aspects. Typical examples of learning platforms are LMSs like Moodle and Blackboard, but in reality a learning platform could be just a simple web page that combines geogebra with other complementary components that provide services like intelligent support.

³<https://migenproject.wordpress.com/>

⁴<https://www.geogebra.org/>

⁵<https://scratch.mit.edu/>

4.6 An architectural aspect of Learning Platforms

Currently there are many learning platforms available offering a wide range of services to educational institutions. From an architectural point of view these platforms can be categorised as follows:

4.6.1 Tightly Coupled

In this category we have systems that implement proprietary platform-specific APIs and the components and dashboards they provide are tightly integrated into the hosting environments. Usually tools of that category are well-tested and robust solutions but customisation and reusability is limited if non-existent. These tools are usually black boxes that provide standard functionality in a very fixed context. The intention of these systems is to provide learning material as well as feedback to students and teachers. Systems like that include Blackboard (Bradford et al. 2007), Khan Academy and several other bespoke platforms tightly coupled with the application they support. Khan Academy⁶ offers some extensibility through plugins but these plugins are tightly integrated with the platform. A notable example is ALAS-KA (Ruipérez-Valiente et al. 2015).

4.6.2 Loosely Coupled

These are systems that have the same objectives and general characteristics mentioned above but implement a more component-based architecture. In such systems the learning platform is composed of pluggable components that implement standard interfaces. Typically it is possible for these components to be used in different platforms and form alternative environments but quite often these interfaces are not compatible across platforms. A system that follows this approach is Moodle. There are plugins that can be used with Moodle and provide similar functionality as above. An example

⁶<https://www.khanacademy.org/>

is Moodog (Zhang et al. 2007).

4.7 Learning Platforms as Ecosystems of Diverse Components

As pointed out in section 2.1 nowadays there is a wealth of web components that are freely available on the web and these components as well as their combinations offer opportunities for supporting learning. Teachers and learners want to have the freedom to synthesise their own learning environments (PLE, OLE) in a way that resembles web-based mashup applications. In this context the objective is to reuse already existing functionality wherever possible and intermix it with new technologies in order to synthesise new and innovative solutions. The benefit of increased reusability is not just increased functionality but also reduced design/development time and cost. Therefore, this becomes increasingly important where there are time and budget constraints imposed. Constraints in this case should not be seen as obstacles but as unique opportunities to rethink about architecture and find new ways to design learning environments.

In this section we are using the term ecosystem instead of platform to show that in an ideal system we expect to have a community of independent components that may be used in various different formations to provide potentially meaningful and innovative solutions. These components are expected to be heterogeneous and disparate but able to integrate with the ecosystem and interoperate with each other. The various compositions may appear as different learning platforms that have their own logic, objectives, constraints and idiosyncrasies. The ecosystem acts as a hub for foreign components. These components may be stand-alone components or parts of other systems that live on the web and offer freely themselves and their services. These include components that may or may not have been designed for educational purposes. Facebook components that offer collaboration services, online code editors and output consoles, OWL reasoners for AI applications are just a few notable examples.

These components may have been designed for different purposes and may follow different architectures. As a consequence of that they may not be able to directly

integrate with the ecosystem. The ecosystem, in this case, provides the infrastructure that enables these components to live together and operate effectively within its bounds. The integration process naturally involves homogenisation. After this process is completed, these components become services that live in the ecosystem. Services may be utilised directly or they may be combined to synthesise other more complete or enhanced services. For example, we may need a service that provides us with an intelligent code editor that features debugging, visualisations and task-independent (TI) support in coding. This could be a composite service that utilises three individual components that offer the functionality required.

4.8 Implementation of AuthELO

Having the integration and interoperability hurdle out of the way, we move on to designing and developing a prototype (Henson & Knezek 1991, Wong 1993) for the authoring tool. According to the findings from 4.2 (User Centric Design) the tool needs to satisfy the following high level requirements:

- compatibility with any web widget that may carry some educational value
- ability to intercept user actions and configure logging on the widgets
- ability to develop automated support based on the logs for widgets operating in open and exploratory settings
- ability to generate support based on rules and analysed evidence
- ability to make support available on demand in a non-intrusive manner
- ability to operate with limited technical and domain expertise

The first of those requirements has already been satisfied with the development of WIIL. We know that SEPIA can be used as the high level architectural framework to overcome monolithic solutions. We also know that CTAT is a promising approach that with some modifications is worth pursuing further in the exploratory terrain. CTAT is simple but not simple enough in exploratory learning settings and for that reason we complement it with FRAME to make the process more compartmentalised and consider different parts separately to lower the cognitive load.

The outcome of this process is a prototype, named AuthELO. AuthELO was designed and developed following the above considerations as general guidelines. From an operational point of view the approach is exactly the same as the one used by CTAT. The difference is that instead of visualising the possible paths we visualise the intercepted user data and facilitate the authoring process through a view that follows the

FRAME architecture. The tool is able to dynamically learn the characteristics of interest from any learning environment attached to it and build dynamically its interface to facilitate easy configuration of data logging. It is also able to perform dynamic and continuous testing of automated feedback in a sandbox with minimal administrative overhead. The outcome of the process is easily deployable as a plug-in that accompanies the learning activity in any context. What remains to be done is to evaluate the tool (Nieveen & Folmer 2013) and see whether it makes development of automated tutors for exploratory learning easier. If that is the case, then that is a significant step forward towards the desired value. In fact, at this stage it seems that even the desired value can be made more specific to better describe the contribution of this work. Therefore, the new form of the reasoning pattern changes to the following:

"ELE with automated support" + "HOW" → "make authoring of automated support for teaching programming easier"

4.9 AuthELO

Programming is by nature an exploratory activity and as such it is learnt better in environments that support and promote creative expression and exploration. For this reason we consider task-dependent intelligent support only in the context of exploratory learning environments that can be used to teach programming like the ones presented in 2.1. These environments typically present themselves as virtual worlds (microworlds) offering the ability to create, modify, manipulate, delete objects, respond to events and manipulate the environment itself. Educational tasks in this context are highly interactive, exploratory activities that can be very effective in supporting learners' development of conceptual knowledge but they require a significant amount of intelligent support. Research in the area has demonstrated that it is possible to delegate part of this support to intelligent components (Bunt et al. 2001, Mavrikis et al. 2013). However, development of automated support requires technical skills and there have been very few attempts to reduce the entry threshold for both programmers and end-users (Blessing et al. 2007).

As shown in section 2.4 there have been many systems developed in the past that can be used for the production of learning material that is interactive and provides automated intelligent feedback to students. All of these systems are typically domain-specific solutions that require low-level technical expertise and usually offer fairly limited and not easily generalisable output. That seriously limits the applicability of these tools to a wider range of learning scenarios and imposes a high entry barrier for low-skilled authors. The aspirational goal behind the development of authoring tools for many years has been to enable users with low technical expertise to create and modify content, including ideally their adaptive features according to their own pedagogical strategies (Gaffney et al. 2010, Murray 2016, Harris 2002). However, the usability of such tools and particularly the time required to invest in learning them, are factors that affect teachers' engagement in the design process of authoring (Karoui et al. 2016). It is

important to understand that teachers have different expertise, needs and motivations and authoring tools should meet their different expectations.

What we see as a viable approach in the context of exploratory learning is a system that is generic (not domain-specific) and versatile enough to accommodate automated support in situations where interaction with the learner is not fully structured. The system should be able to provide an authoring interface through which users with limited knowledge-engineering and programming skills can develop automated support for any educational task. This tool should be able to simplify the authoring process to the extent that the cognitive load needed for the author is minimal so that authoring activities can be focused on the problem at hand and not wasted in the low-level technical details needed for the implementation.

4.9.1 Design

At a conceptual and architectural level we see an authoring tool for intelligent support in this context as a system that resembles SEPIA (refer to 2.5.2). SEPIA is designed so that automated support can be added in the form of an epiphytic application that is external to the learning environment. Integration does not require changes in the target environment and interoperation is not based on domain specific models and tools. From a more practical viewpoint our approach is very close to the example-tracing approach (refer to 2.4.2) introduced by Alevan et al. (2009, 2016) as this approach seems to require the least amount of cognitive load for the author. According to this method, the author develops feedback by doing the learning activity like a student. The system generates a tree-like diagram that depicts the current state of the student. The author can utilise this information to infer the current student understanding about the given task. Then, the author can decide about the type and level of support needed and annotate the diagram accordingly to determine the behaviour of the intelligent tutor. Although this approach is simple and allows non-expert developers such as learning

designers or teachers to author automated support, it is fairly limited in the sense that it is domain-specific and not generalisable beyond structured tasks. Structured interaction may be adequate for traditional ITSs but in the context of exploratory learning it is problematic since it covers only a very limited range of the possible alternative paths. In an exploratory learning environment these paths are potentially infinite since typically there are very few restrictions that constrain the learner trajectories and ability to do things.

Our approach is a variant of the example-tracing method. The method is essentially the same but we are not using the visualisation part that shows the potential paths simply because it is impossible to represent visually all the possible states in such diverse domains and open environments. The authoring tool must be generic enough to accommodate dynamic learning scenarios that take place in exploratory learning environments. In these environments support is expected to be task-dependent but tasks may not be fully controllable and structured. In this context, authoring must be based on activity indicators (data) that become available as the author interacts with the environment. The author tries to cover as much as possible of the potential student trajectories by executing the activity multiple times and gather data that is representative of those paths. This information can then be utilised to form sensible rules that determine the generation of automated feedback. Author-generated data as well as student-generated data may be used to revisit the initial design so that we can achieve, after multiple iterations, a model that is representative of the common student misconceptions for the particular task.

The example-tracing approach along with the fluidity of the data-based modelling approach tackle the domain specificity problem and unstructured interactions but the entry threshold for non-tech savvy authors is quite high. Since, there is no visual programming involved anymore and knowledge engineering techniques must be applied on user activity logs the technical difficulty seems to be prohibitive for non-skilled users. In our attempt to remedy this problem we first decided to decompose the au-

thoring process and thus reduce the complexity of the authoring task. Our methodology was based on previous work presented in the FRAME approach (Gutierrez-Santos, Mavrikis, Magoulas et al. 2012) - see figure 2.2 (refer to 2.4.10). According to this approach, the complexity of the authoring task can be reduced and made more manageable through the compartmentalisation of different concerns regarding the different aspects of the problem that may require different approaches and expertise. Considering these aspects separately reduces the skill threshold required to deal with the problem in its entirety. In practice the development of support using this method goes backwards and resembles to an extent the Model-View-Controller (MVC) architectural pattern in software design. Development tasks move towards the opposite direction of the data flow as it is presented in the diagram 2.2. Authors start from the presentation component and gradually move towards the underlying concepts that need to be addressed and development of components that intercept user or system activity and produce evidence.

Our design is influenced by the view that an exploratory learning system should not intervene in the process in an intrusive manner (Mavrikis et al. 2013). Support should not be provided in order to manipulate the students and control their behaviour which is typically the case in traditional ITSs. Support should be as discreet as possible and inform the users for potential issues but not interrupt the learning cycles and the educational process. That, of course, does not contradict with the fact that support should always be available on demand. Learners may not be able to exploit the full potential of exploratory learning environments if there is not enough support available to direct them (Mayer 2004, Klahr & Nigam 2004, Kirschner et al. 2006).

Taking all of the above into account we used a design-thinking approach that involved the development of a prototype and multiple re-design iterations. The starting point in this process was to identify the challenges potential users would face in developing functionality for each part of the authoring process and especially the parts that deal with reasoning and the acquisition of evidence that can support it. For this

we used participatory design methods involving expert designers and educators. For the reasoning part we collected a number of use cases of specific learning activities developed in three well-known ELEs. The experts provided us with complete usage scenarios for each activity that include potential student misconceptions, landmarks that can indicate important states of the constructions and the respective feedback that the system is expected to provide to students. This information helped us form an initial idea of what is needed for the reasoning part and they also were transformed into batteries of tests for a technical evaluation of the prototype tool. Having this information about the reasoning part enabled us to identify requirements for the evidence part.

Decomposing the authoring process is a crucial step to reduce the complexity but it is not enough if low level technical knowledge is needed regarding the architecture of the learning platform and coding is needed to acquire and process user data and domain knowledge. The architectural design of the learning platform should not be a concern for potential authors. Understanding the architecture of the system and coding for integration and interoperability is typically work for programmers and not learning designers and teachers. Acquiring and processing data and domain knowledge is, though, work for authors of automated support but if coding in a 3GL is needed, then that part becomes especially difficult for them. Compartmentalising the concerns and providing a different authoring component for each eases the difficulty but not enough. There have to be two additional levels of abstraction to make this process more accessible to users with limited technical expertise. The first level is to design and implement high-level, ideally declarative, languages for each component. This gives the opportunity to skilled learning designers that want to have a lot of control to produce fine grained solutions to accommodate sophisticated support scenarios. A tool that may be used for the production of such languages is presented in 6.4 (LFT). A first attempt to identify the requirements for a high-level specialised language for the data acquisition part was made in a discovery workshop we conducted

in 2018 at UCL Knowledge Lab (Mavrikis et al. 2019). Participatory design methods were used again, involving three experienced ICT teachers with enough programming background to teach computing but not necessarily enough to develop applications as professional programmers. They were all skilled in basic JavaScript and they were supported to develop 15 different activities in MALT+ and the corresponding automated support using the prototype tool. One of the objectives of this workshop was to see how authoring takes place and identify difficulties, commonalities and patterns in their solutions that could provide the basis of a higher-level language for evidence acquisition. The findings revealed the following requirements:

- No of actions performed by the learner since the beginning of the session
- No of actions involving a given element
- The first action that involves a given element
- The last action that involves a given element
- All the actions that involve a given element
- The time elapsed since the session started

The second level is a visual programming environment that encapsulates the abstractions and the functionality provided by the first one to enable development without the need to learn a programming language. This could be a block-based interface like scratch or Blockly. That gives the opportunity to low skilled designers or teachers to develop simple support scenarios with ease and gives easier and faster access to the same functionality to skilled learning designers. This requirement was confirmed in the same workshop at the UCL Knowledge Lab. For this part six newly qualified teachers who were studying at the UCL masters in Education and Technology were carefully selected based on a non-random sampling strategy. These students had a range of expertise in using technology in educational settings, but no programming background.

One of the key findings of the session was the need for an easy to learn and use language based on block coding. The students identified the need and proposed various language constructs that could be used to express concepts in this context.

4.9.2 Architecture

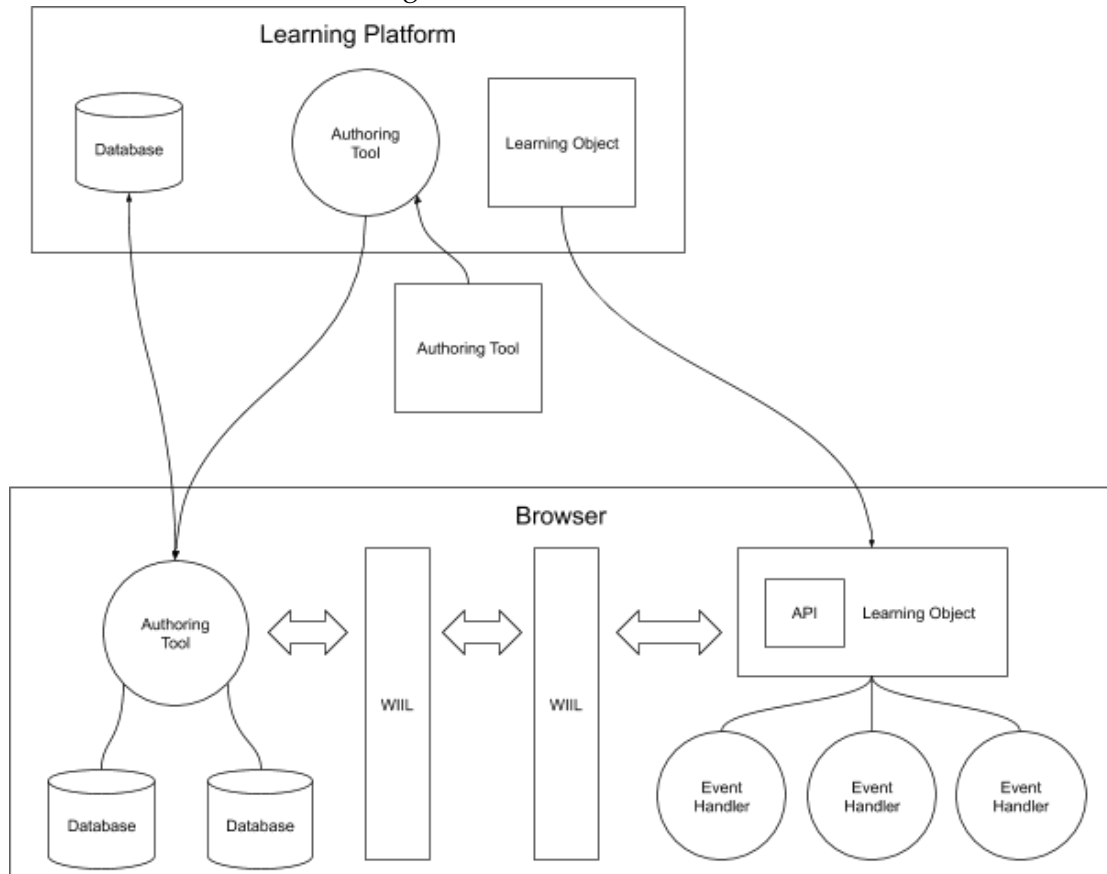
As mentioned previously, we think that the web browser is the de facto deployment platform for educational software nowadays. Therefore, the proposed architecture considers an authoring tool that is a native HTML5 application with no external dependencies and is physically and logically decoupled from learning platforms. The tool should be able to interoperate with any other web component (widget) that may be used as a learning environment as long as the latter is equipped with an API. That implies that it must be generic enough to accommodate interfacing with diverse architectures, APIs and data models. Implementing any platform-specific or proprietary functionality would make it deviate from that principle and therefore is not desirable. The design should follow the SOA paradigm so that the service can be utilised through standardised communication protocols and data formats.

As mentioned above, the tool may be physically and logically decoupled from the learning platform but after it has been "virtually" integrated the whole system should look unified and homogeneous. From the users' perspective we see integration as a very simple procedure that involves a single step. This is to provide a URL along with parameters that give information about how to instantiate the learning object (environment) to be configured and where to store the configuration data. This information is stored in the learning platform and is used whenever an author wants to configure logging and automated feedback for a learning object. The author initiates this process in the learning platform and implicitly gets redirected to the authoring tool (AT). The tool then creates an instance of the learning object (typically a web widget) that lives in its own private and secure space (sandbox). The two software components oper-

ate as independent applications in parallel (asynchronously) within the same browser instance. The interfacing between them is done by the Web Integration & Interoperability Layer introduced in (Karkalas, Bokhove, Charlton & Mavrikis 2015, Karkalas, Mavrikis & Charlton 2015). WIIL is a thin layer that can be used to integrate own or third party web components with a platform. It can also provide a simple but efficient communication mechanism so that the integrated components can interoperate with each other. Once the learning object is instantiated, it sends to AT its environment-specific metadata. This is information about the types of elements that can exist in the environment and the types of events that these elements can generate. This data is maintained in local in-memory databases at the AT side of the browser. The AT uses this information to construct dynamically a graphical user interface for the configuration of user activity logging. Something that needs to be noted here is the dynamic nature of this process. There are no presumptions about the information that is received from the widgets. Different widgets may provide different metadata and that, in turn, may result in the formation of different interfaces.

This GUI becomes immediately available to the author and the author can use it to set up data logging rules that have an immediate effect on how the instance behaves. The rules are stored in local databases in AT and they are also sent to the learning object so that the respective event handlers can be registered. After registration, the widget is able to generate data according to what the author prescribed in the configuration interface. This data becomes directly available to the author for inspection. The author uses this information to make decisions about what feedback needs to be provided to the students. This part of the configuration is also stored in local databases. The logging and feedback configuration is then passed by the tool to the learning platform, so that these settings can become available to the actual learning object instances that are going to be used by students. The learning platform sends these settings as part of the initialisation parameters during the widget launch process.

Figure 4.7: The Architecture



4.9.3 Integration at a Technical Level

The basic ingredient that makes authelo versatile and not tightly integrated with specific platforms and learning objects is WIIL. As seen in 4.4 WIIL can be used to provide a seamless integration between a platform (host) and a learning object (guest). This scheme can have many levels and result in a chain of components encapsulated one into another. In order to understand integration of authelo with interrelated components we have to consider two distinct cases.

Authoring phase

During authoring authelo is used like a learning platform that can host the learning en-

vironment of choice. In this scenario authelo is implemented as a web page equipped with WIIL and able to incorporate any WIIL enabled learning object. The learning object to be enhanced is referenced in an iframe element and once registration is done the two nodes start communicating with each other through HTML messaging. The learning object informs authelo about its environment-specific characteristics like the types of elements and the types of events available and authelo generates dynamically its authoring interface to accommodate these requirements. The author at that stage can start setting up logging rules and the learning activity to experiment with. All the information generated by those activities will then have to be stored somewhere. Authelo is not itself a learning platform, although it might appear as one. The information that derives from its operation should persist and made available for future reference by software running this activity in learner terminals. In order for this to happen authelo must communicate this information to a learning platform. Integrating authelo with a learning platform is equally easy as integrating it with the learning object. The difference in this case is that authelo is the guest and the learning platform its host. Once registration is done both systems can start operating in parallel and the author of automated support can save logging, support and evaluation rules to the platform. Saving in this case entails sending this information to the host platform in messages and letting the platform decide where and how to store it. This way authelo can stay as neutral and independent as possible from implementation details specific to its interoperating peers. It is just a component in the middle that handles information generated by the author and the learning object.

Deployment phase

Once the authoring is done and all the related information is stored in the platform database the learning activity along with the automated support created for it can be deployed in student terminals. A student terminal can be thought of as a learning platform implemented as a web page that can incorporate a learning object in an iframe.

Let us call this platform the activity player. A cut-down version of authelo implemented as a library must be referenced by this player. This gives the player all the necessary equipment to store locally information about logging, support and assessment rules as well as the infrastructure to execute those rules and provide assistance. Once the player gets instantiated and information about the learning activity becomes available the authelo infrastructure gets populated with the respective rules and automated support is ready to be used. Information about the learning activity entails the instantiation of the enclosed learning component in its iframe. Both the host (player) and the guest components are integrated through WIIL and once registration is done communication between them can begin. The student can start interacting with the learning object and activity indicators will start flowing from the widget to the host. The host will activate the logging rules to select the indicators of interest and store them in local in memory databases for authelo use. Once the learner hits a button to request support the corresponding rules will get evaluated and assistance will be displayed. The data generated either from the widget or from authelo itself like the support provided can then be sent off to the database of choice by the player. Authelo is completely unaware of this data and platform-specific operations. Again, as in the previous scenario authelo operates like the intermediary that intercepts data, activates given rules and displays assistance in a totally neutral and disconnected manner.

4.9.4 The PoC

This section presents a tool that has been designed according to the methodology and architecture presented in the two previous sections. The name of the tool is AuthELO and stands for feedback "Authoring for Exploratory Learning Objects". AuthELO can be used to configure user activity logging rules and rules that dictate how automated feedback for exploratory learning objects (ELOs) can be generated based on these logs. ELOs are web widgets that instantiate highly interactive exploratory learning environ-

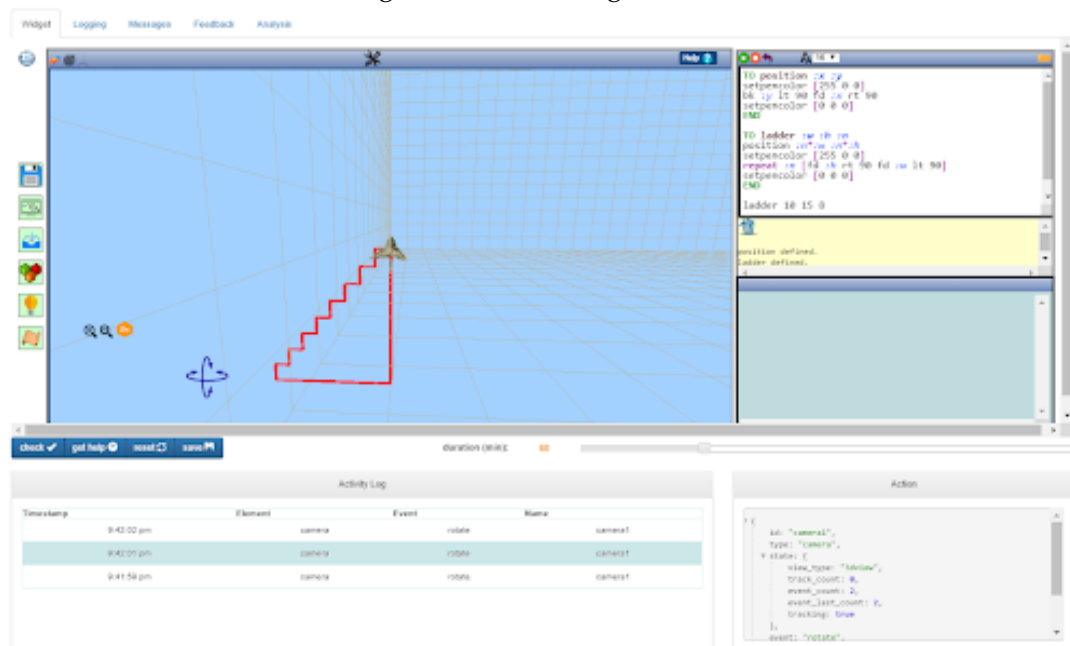
ments. ELOs are expected to be highly diverse and there is no assumption about architectures, APIs, communication protocols or data formats. AuthELO was designed to harness the heterogeneity of those ELOs and provide a simple, common and efficient authoring interface for automated support. A central design objective of AuthELO was to normalise the discrepancies between the ELOs and give the ability to non-experts to easily develop or modify feedback that is provided to students based on their interaction. The design is based on the following five main requirements as they were formulated after multiple re-resign iterations:

- Authors must be able to dynamically configure what data will be logged by a learning object during a session with a user. This can be data generated from interactions between the user and the widget and derivative data that gets generated by the widget itself as a result of some event.
- Authors must be able to specify rules about real-time feedback that should be provided to the students. These rules should be based on log data that is dynamically generated as the student engages with the activity.
- It should be possible for authors to configure all the available widgets through a common interface. This interface should be able to hide the diversity of potentially heterogeneous learning components that might be offered in the system.
- The tool must not impose barriers in terms of skills and technological expertise. Teachers with a certain degree of IT literacy should be able to use it for authoring interactive learning material.
- The tool must be able to offer opportunities for exploratory authoring of feedback reducing the cognitive load that is expected for non-structured tasks of exploratory activities.

The main screen of the tool presents the instantiated ELO along with a table view of the user activity logs. User activities that are being logged appear in that list and

are selectable by the author. Selecting a user action reveals the underlying data of the respective indicator in JSON format under the 'Action' section of the page. That provides the author with a very good view of the data and can be used for an initial data exploration or the configuration of data handling and feedback rules, figure 4.8.

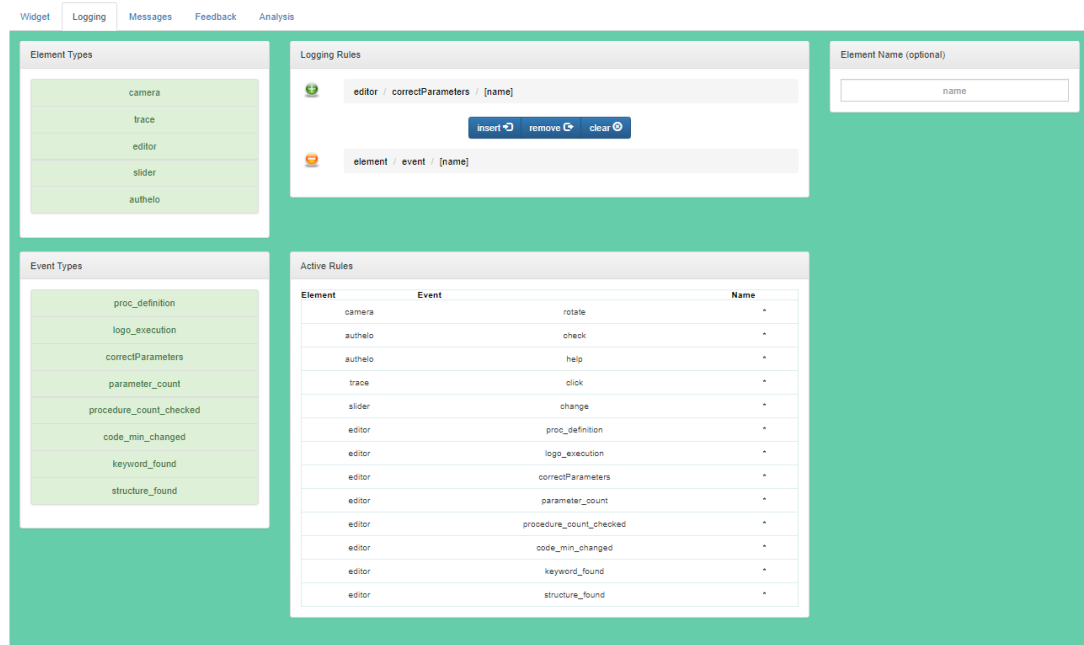
Figure 4.8: The Widget View



The author can use this instance of the ELO to do the activity like a student and check the coherence and logical flow of the expected interactions. This process generates data. The data that is being captured reflects the data logging rules that have been previously configured in the system. The configuration of logging rules is a part of the authoring process that is not depicted in the FRAME diagram because it is implied that the learning platform takes care of it in the background. The second tab shows the configuration options for this part, figure 4.9.

This part of the interface is generated by AuthELO dynamically based on the meta-data that is received from the enclosed ELO. Although this is ELO-dependent information this interface is able to dynamically accommodate any discrepancies in terms

Figure 4.9: The Logging View



of element types and events between diverse ELOs. The author can combine element types, events and potentially names of individual elements to configure logging rules. Once inserted, a rule becomes immediately effective and the ELO starts generating data. Rules are stored in a local database in AuthELO and they are also sent to the widget through WIIL so that the respective event handlers can be registered. After registration, the widget is able to generate data according to what the author prescribed. Data logging is essential because ideally we want to prevent data flooding from happening by logging every little unnecessary detail.

Evidence detection is the next part and that requires some data pre-processing so that we can transform the data logs to usable information that reflects the concepts to be supported. This processing can be done by using JavaScript in the analysis tab, figure 4.10. If the analyst is a competent programmer then fully-fledged JavaScript can be used to accommodate very fine and detailed processing. In addition to that, there is also a JavaScript library available that provides a high-level interface to the

most common data collection and analysis requirements of typical authors. The language constructs available reflect the requirements identified in 4.3.1. This library is the first step towards the design of a high-level language specialised for that part of the processing. The high-level language constructs that are available follow:

- actions.size()
- actions.noOfActionsBy([property],[element])
- actions.firstActionBy([property],[element])
- actions.lastActionBy([property],[element])
- actions.actionsBy([property],[element])
- actions.timeElapsed()

They all become available through an object with a global scope named 'actions'.

Figure 4.10: The Analysis View

The screenshot shows a web-based interface with a navigation bar at the top containing 'Widget', 'Logging', 'Messages', 'Feedback', and 'Analysis'. The main area is split into two panels. The left panel, titled 'Code Editor for Analysis Authoring', contains the following JavaScript code:

```

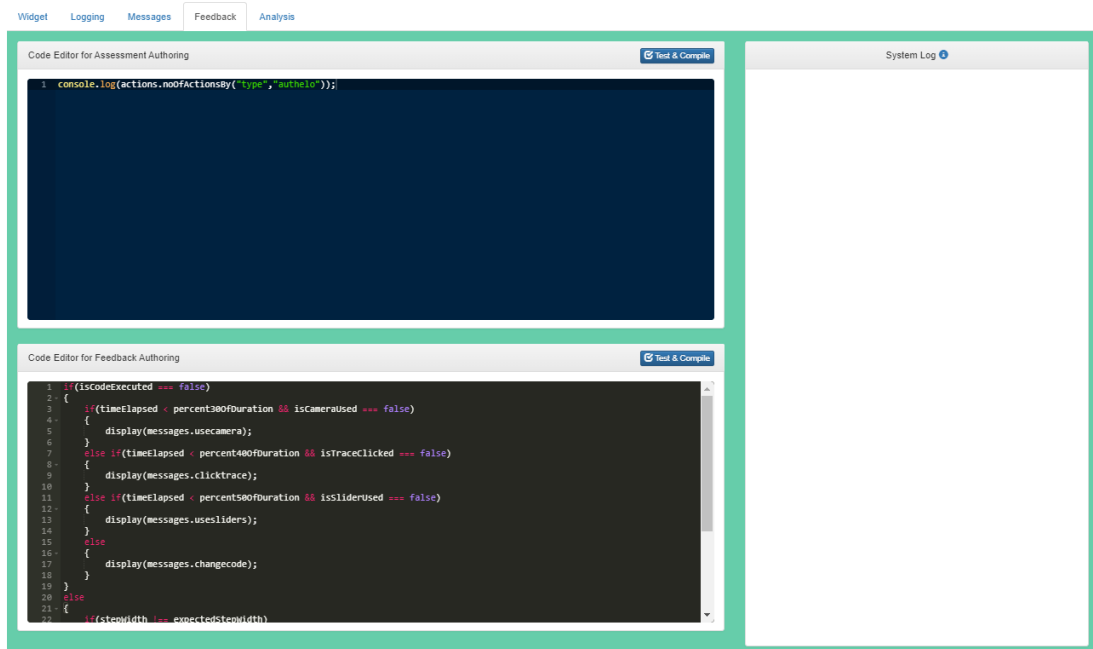
1 var timeElapsed = actions.timeElapsed();
2 var durationInMin = duration(duration);
3 var percentOfDuration = 0.3 * durationInMin;
4 var percentOfDuration = 0.4 * durationInMin;
5 var percentOfDuration = 0.5 * durationInMin;
6 var actionCount = actions.size();
7 var isCodeExecuted = actions.noOfActionsBy("type","editor") > 0;
8 var isCameraUsed = actions.noOfActionsBy("type","camera") > 0;
9 var isTraceClicked = actions.noOfActionsBy("type","trace") > 0;
10 var isSliderUsed = actions.noOfActionsBy("type","slider") > 0;
11 var stepWidth = actions.lastActionBy("event","log_execution").state.value.parameter_values[0] || 0;
12 var stepHeight = actions.lastActionBy("event","log_execution").state.value.parameter_values[1] || 0;
13 var expectedStepWidth = 10;
14 var expectedStepHeight = 10;
15 console.log("time elapsed " + timeElapsed);
16 console.log("duration in min " + durationInMin);
17 console.log("30% of estimated duration is min " + percentOfDuration);
18 console.log("40% of estimated duration is min " + percentOfDuration);
19 console.log("50% of estimated duration is min " + percentOfDuration);
20 console.log("no of actions done " + actionCount);
21 console.log("is code executed " + isCodeExecuted);
22 console.log("is camera used " + isCameraUsed);
23 console.log("is trace clicked " + isTraceClicked);
24 console.log("is slider used " + isSliderUsed);
25 console.log("step width given " + stepWidth);
26 console.log("step height given " + stepHeight);
27
28

```

The right panel, titled 'System Log', is currently empty.

The system log view provides a real-time view of errors or potential problems identified by the system as the author writes code in the editor. The author can also utilise this view to display their own logs and debug their code. Once the data needed is collected and properly formed they can be made available to the Feedback tab, figure 4.11. This is the place where decisions are made regarding when and how the feedback is going to appear. Feedback rules are split in two categories. The first category concerns the 'what' needs to be produced and the second concerns the 'how' implementation should be done. The editor titled 'Assessment Authoring' informs the learner about how much of the expected output has already been created. The 'Feedback Authoring' editor is used for rules that generate instructions that help the user overcome problems and move towards the designated target. These editors can refer to artifacts generated in the analysis tab.

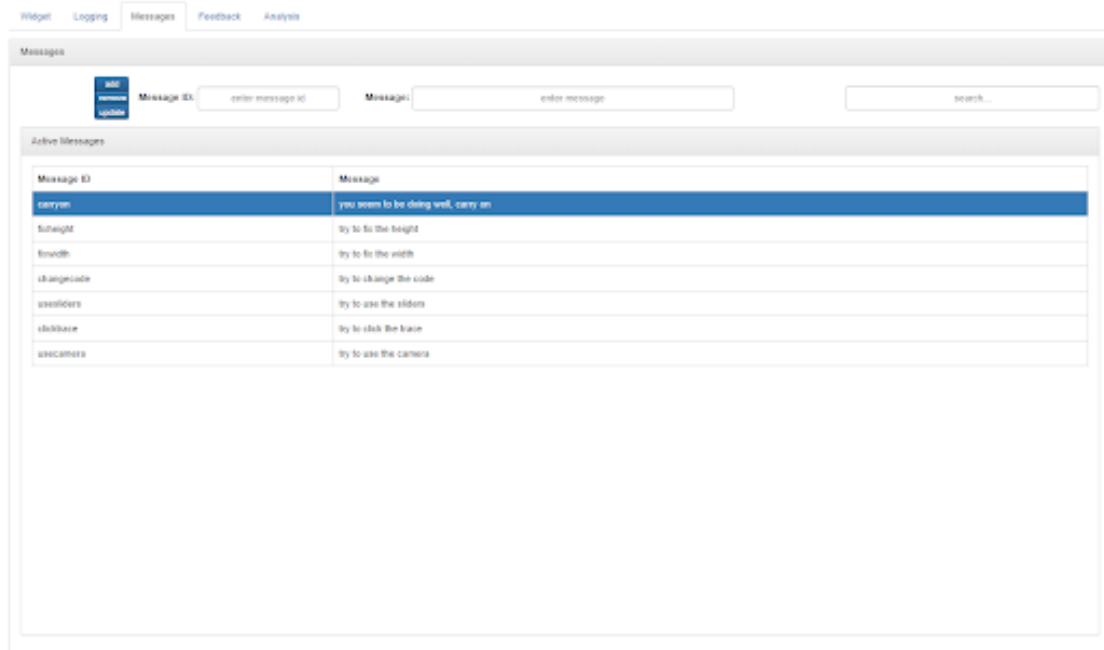
Figure 4.11: The Feedback View



The last tab is titled 'Messages' and can be used to create named variables that hold pieces of text that we want to display as messages to the learner - see figure 4.12. These

variables can be directly referenced in the authoring tab to generate messages.

Figure 4.12: The Messages View



Once all the above steps are completed and there is a feedback support scenario in place, the author can go back to the first tab to test the rules, figure 4.13. The rules have immediate effect in the authoring environment and can be tested without any administrative and redeployment overheads. The author can reset the logs and start the activity from the beginning to generate the data needed to simulate the problem. Feedback can be requested using the buttons under the ELO.

Figure 4.13: Feedback Testing

The screenshot displays the AuthELO software interface. The main window shows a 3D grid with a green ladder object. The ladder is composed of several steps, each with a width of 5 units and a height of 7 units. The ladder is positioned at a distance of 10 units from the origin and has 8 steps. The code editor on the right contains the following commands:

```

TO position :x :y
  setpencolor [255 0 0]
  bk :y lt 90 fd :x rt 90
  setpencolor [0 0 0]
  END

TO ladder :w :h :n
  position :n* :w :n* :h
  setpencolor [255 0 0]
  repeat :n [fd :h rt 90 fd :w lt 90]
  setpencolor [0 0 0]
  END

ladder 10 15 8

```

The console output shows the following messages:

```

position defined.
ladder defined.
oops: to position needs an end
position redefined.
ladder redefined.

```

The sliders for the ladder function are set as follows:

Name	From	To	Step
w	5	20	1
h	7	30	1
n	4	18	1

The bottom status bar shows a duration of 60 minutes. A feedback message on the right says: "Feedback you seem to be doing well carry on".

5

Evaluation of AuthELO

AuthELO is a central contribution of this work. It is designed based on data collected and findings derived during the thesis and it is also a component designed to address needs that emerged during this thesis. As a design it needs to be evaluated and potentially go through re-design iterations until the design objectives are satisfied. For this purpose there are two evaluation cycles that complement this process. In order to have a comprehensive coverage, for the first part of the evaluation we use learning technologists whereas for the second part we use industry experts.

5.1 First Evaluation

The first evaluation started with a workshop involving the three learning designers that contributed to the initial design of authELO. This study was targeted to JavaScript developers (and not teachers) as at this stage AuthELO did not support high level abstractions like block-based coding for the development of feedback.

The workshop was divided into three parts. In the first part, the participants were asked to develop automated support on their respective widgets for the same usage scenarios used in the design stage. The time required to develop the scenarios was recorded. In the second part the experts were interviewed in order to identify key phases in the authoring process. The common denominator of those phases was a series of seven steps that seem to give the basic workflow followed in the development process. This workflow was an important finding because it was used in subsequent steps to help with the qualitative evaluation of findings and ultimately to further inform subsequent re-design iterations of authELO. The final part involves evaluation of authELO via a usability testing. The aim of this test was to measure speed, efficiency and user satisfaction.

To facilitate the process and have a meaningful comparison, we followed three different approaches based on the particularities of the widget:

1. Geogebra: To our knowledge, there is no standard way of authoring feedback on dynamic geometry activities like GeoGebra. We followed our previous work (Karkalas, Bokhove, Charlton & Mavrikis 2015, Karkalas, Mavrikis & Charlton 2015) and provided an experimental platform that takes advantage of Geogebra's API and allows quick integration of widgets and event handling capabilities to ease the work of the programmers. A widget was preconfigured to generate every possible event for every element present in the construction and the data was collected in the central repository of the learning platform.
2. FractionsLab: FractionsLab was developed in C# in Unity (Hansen et al. 2015). For authoring feedback the developer wrote code for a particular task, after a framework for support had been developed in the Unity Development Environment.
3. MALT+: This is an HTML-based widget developed in JavaScript. For authoring feedback the corresponding developer first established an approach to handle key events from the environment and then wrote code in JavaScript.

We allowed this way the strength of each platform, programming language and developer expertise to manifest itself rather than setting up the three developers to fail. After the task, we interviewed the developers to identify key phases in the process, which we summarise below:

1. find the item(s) of interest in the construction
2. consult the documentation of the widget to see how they are represented in the data
3. go to the back-end database server or intercept data locally (e.g. consulting the browser console) to determine how events from the system can provide evidence for determining the feedback

4. write the code that uses this evidence to generate the message
5. reset (e.g. reload or in some cases re-compile) the system and start again the activity
6. perform all the actions needed to form the state that generates the feedback
7. check whether the feedback is correct and either move on or repeat the process

This cycle of actions requires a significant amount of time if the author needs to move back and forth between the server and the client part of the application multiple times. This is the case even if both the client and server components reside in the same physical tier. Configuration for the data logging may not be needed but the fact that the database may be filled with irrelevant data makes retrieval and processing more difficult. The fact that new code cannot have immediate effect and the author has to reset the system and go through the activity steps again is possibly the biggest obstacle in this process. Things become even more complicated if the author makes a mistake. This process can be an exhaustive experience for the author. It is pretty obvious that the process as a whole imposes a high cognitive load and reduces the effectiveness of the authoring task.

The third component of the workshop was a usability test. The aim of this test was to measure efficiency in terms of speed and ease of use. In particular the metrics (Seffah et al. 2006) used follow.

AuthELO is considered as a successful tool if:

- (speed) A non-expert using authELO can achieve at least 20% time gain in development of automated support comparing to an expert without authELO
- (ease of use) A non-expert using authELO can develop the same level of support as an expert without authELO

The experts were randomly allocated to a different widget. To ensure that the odds are the same for all participants a random generator app was used as follows:

1. The three participants as well as their respective widgets were numbered. Initially they all have their own widgets.
2. The facilitator (researcher) runs the app once to get a random number between 1 and 3
3. The participant selected runs the app as many times as necessary to get a number (between 1 and 3) different to their own number - allocation 1
4. The remaining participant that still has a widget is allocated the widget of the participant in step 3
5. The remaining participant is allocated the only widget available

Three representative learning activities, one for each widget, were selected and used for this part. There was a short familiarisation session with the particularities of widgets, the respective activities/usage scenarios and authELO. For the familiarisation session there was also a process to follow. The first phase was the familiarisation with the widgets. For this there were three 1:1 sessions that took place sequentially. The widget expert gave a short tutorial to the non-expert on the allocated widget. The second phase was the familiarisation with the activities and authelo. For this there were again three 1:1 sessions that took place sequentially between the facilitator and the non-experts.

This part was necessary because the users were not knowledgeable on the widgets. Then, the users were asked to develop support for the given activities through authELO. Again, the time required to develop the scenarios was recorded and the experts were asked to evaluate the level of support developed by the non-experts. Then a discussion on the findings based on the key phases of the authoring process took place in

order to have a more qualitative evaluation and justify the outputs. The quantitative results are given in table 5.1:

Table 5.1: Time Savings with AuthELO

	Native	AuthELO	%
GeoGebra	245	135	44.8
FractionsLab	330	250	24.2
MALT+	103	82	20.3

It is evident that there are substantial time saving possibilities with AuthELO. Especially if we take into account the fact that in the first experiment the authors did not have to configure data logging at all and they could start directly with authoring the feedback, we can see that in all cases the task was completed by a different developer in a shorter time than the original expert in this tool. Observing the developers using AuthELO we confirmed that, despite the short familiarisation session, developers were able to select the items of interest and check directly whether the widget generates the data required. The data gets displayed dynamically as the author interacts with the widget. There is no need to consult the widget documentation for anything or to switch context and query the back-end database. We think that this is a really important point, contributing significantly to reducing the overall time, particularly because otherwise one needs to spend a significant amount of time going through the events that generate data, especially in the context of exploratory learning objects. This is not something we can expect the average teacher to have the training or time to do. The feedback code has immediate effect and messages can be generated on the spot. There is no need to reset the system and repeat the activity. Testing the changes is a matter of pressing a button. Mistakes can easily be fixed. Syntax problems appear dynamically as the author is typing the code. Author messages that display values for debugging are presented in the front page when the logic for the feedback is checked. In conclu-

sion the whole cycle is performed at one place and the available utilities simplify and speed up the process.

After the workshop a re-design iteration of authELO followed to address important findings like the basic workflow of the development process. The new version of authELO was then evaluated again with the same criteria as before but this time in the context of a real project. Two of the participants in the evaluation workshop that specialise in Malt+ were selected to take part in this experiment. The purpose of this evaluation was to confirm previous findings after changes in the design.

This evaluation was a quasi experiment - single group study (Privitera & Delzell 2019) that took place in the context of a real project. The aim of the experiment was again to evaluate the tool with regard to the same variables as above: speed and ease of use. The conditions under which this part took place did not allow us to have full control over the process. Instead of having a control and a test group we used a single group of two designers completing the same task in the same context twice. The task was to integrate Malt+ with a learning platform and develop automated support for a number of activities. The only difference - and thus the independent variable - in the second iteration was the involvement of authELO in the authoring process.

After the experiment we interviewed the designers and asked them to provide development estimates for both iterations. The results clearly confirm the findings obtained from previous studies. The designers were able to develop the same level and quality of support in approximately half the time with authELO. They said that in the first iteration it took them two weeks of full-time work to develop the required level of support. The same task with authELO took them four days full-time. The designers also responded positively to questions about usability and satisfaction. The outcome is positive but we need to consider the issues raised regarding the internal validity of such an experiment. For example, the fact that the second time the designers perform the same task may have a positive correlation with the improved performance due to a learning effect (Zhang et al. 2014). Even if that is the case, the estimated difference is

substantial and it seems safe to assume that it exceeds estimates for the learning effect. Finally, the fact that the process took place in the context of a real project adds more ecological validity to the experiment.

5.2 Second Evaluation

The second evaluation was conducted utilising the qualitative method of focus group (O’leary 2017). The group comprised five top software engineers senior to lead level working as part of an elite team responsible for the R&D of a major consulting company in the software industry. These people typically work on research projects and develop prototypes. They are accustomed to dealing with diverse technologies and do rapid development of high quality.

The aim of this session was to obtain time estimations from industry experts about the development of the basic infrastructure for a system like authELO and the key phases in the authoring process identified in the previous step. We also wanted to have qualitative evaluations of existing and proposed features from people that have industry background and high expertise in areas related to the underlying technologies used. The benefit from that was to have a more complete evaluation that covers areas likely to be overlooked by regular mainstream users. The target group of users for authelo is learning technologists and skilled teachers. Top software engineers in the industry have very different profiles, experiences and technical abilities than these users. In that respect these are lead users and not mainstream ones (Urban & Von Hippel 1988). Lead users are the ones that have everyday experiences of similar problems, they may have already thought of possible solutions for those problems and they will bring those experiences in the discussion to uncover issues, complement solutions and verify accomplishments. Lead users are users that experience these problems years ahead of the majority of regular users (Ulrich & Eppinger 2011) and this is why their input is very valuable when evaluating new designs.

The participants were given a guiding worksheet and the session started with a presentation of the subject with the researcher as a facilitator. An example activity developed in Malt+ was given as a point of reference during the presentation. Then the developers were given access to authELO and a short familiarisation session with

the technology took place. After this session the participants were given some time to brainstorm, discuss their understanding of the system and try to identify similarities with problems they had previously experienced in their line of work. During this discussion there were many references to previous projects and similar challenges especially with regard to integration and interoperability. The final part was to get the participants to fill in the given worksheet with time estimates and their qualitative evaluation.

The worksheet was divided in three sections covering the basic infrastructure, AI authoring and the participant opinion on authoring automation features. The final part was a discussion about weaknesses and suggestions for improvements.

We can't really have a meaningful direct comparison between the two cases (with or without authELO) as far as the basic infrastructure is concerned because without authELO a developer typically has to go through all the steps needed to set up the infrastructure. The only comparable component between the two scenarios is integration and interoperability because this is the only part of the process that is not fully automated. The average time needed to integrate a new component with the platform is 180 minutes and that is about 40% of the time needed to do the same without authELO - see table 5.2. If we count all the rest of the components needed, then a developer using authELO would need only 7.18% of the time required without it.

The second part is about the AI authoring process. In this case the developers tried to overestimate and think of the worst case scenario - see table 5.3. There are certain steps in the process that don't have to be done in authELO, like logging rules. That alone seems to be giving a big advantage to authELO. The developers gave the same estimate even in code writing for feedback rules which may not be true in reality, given that authELO provides a high-level interface for that purpose. They also responded that they would need the same number of iterations in both cases. Nevertheless, in total, authELO seems to be able to perform the same task in half the time.

The third part was about rating authoring automation features. The developers

were given five Likert scale (1-10) and two open-ended questions. The first five questions were about distinctive features of authELO. As shown in table 5.4 they all gave top rating for all the features. The feature that was given the lowest rating was the high-level language specialised in feedback authoring. This was no surprise as these people were all competent programmers and possibly thought that using a lower level language would give them more control over the process.

The two open ended questions where:

1. Any suggestions for additional features?
2. Any obvious limitations?

In that part two suggestions were given. One is to provide more information about the components and instructions on how to use them. The other one was about the language. They said that it would be preferable to have a prompted, text-based parser instead of a block-based visual language. That would be faster to use. Again, this is said from the perspective of experienced programmers and is obviously not applicable to low skilled learning designers and teachers. The limitation identified was that there is no standard structure for components and therefore a developer is needed to add them to the system. That requires some technical Knowledge as there will be cases where the integrator will have to write WIIL implementations for the various component APIs. This is true, but inevitable if you have to deal with non-standard and diverse components.

In conclusion we could say that although the overall estimations confirmed the time gains achieved in the previous steps the engineers did not seem to significantly appreciate the contribution of authELO regarding the key steps of the authoring process with a notable exception of evidence preparation. The fact that they had a very high level of technical expertise played a role on that. In the qualitative part of the evaluation they gave top ratings to all the distinctive features of authELO. They also

expressed their preference on a high-level language specialised to authoring rather than a block-based visual programming environment.

Table 5.2: Results for Part 1 - AuthELO

	Manual	AuthELO	Gain
Part 1 - Infrastructure	time(min)	time(min)	%
Database	180	0	
Web server	180	0	
Server-side components	564	0	
Web page	516	0	
Client-side components	612	0	
Integration & Interoperability	456	180	60.53%
Total in min	2508	180	92.82%
Total in hours	41.8	3.0	92.82%

Table 5.3: Results for Part 2 - AuthELO

	without AuthELO	with AuthELO	Gain
Part 2 - Automated Feedback Authoring	time(min)	time(min)	%
find elements of interest in the construction (using UI or the docs)	300	300	0.00%
consult the API docs to see what events they respond to and what data they generate	180	180	0.00%
write code to intercept only the events of interest and exclude irrelevant data (logging rules)	240	0	100.00%
interact with the widget to generate some data (do the learning activity)	264	264	0.00%
check the data in the db and analyse, aggregate as appropriate (prepare data for the decision making part)	192	48	75.00%
write code to determine what help may be useful for the learner based on the data (ai support rules)	408	408	0.00%
recompile, reset system and do learning activity again to generate the same data	396	0	100.00%
check the messages generated by the ai support code (testing)	336	0	100.00%
number of iterations you think needed to get the desired outcome from the above process	9.6	9.6	0.00%
Total in min	2316	1200	48.19%
Total in hours	38.6	20	48.19%

Table 5.4: Results for Part 3 - AuthELO

AuthELO Features	Rating (1-10)
visual definition of logging rules and instant deployment	9.4
real-time visual inspection of data generated as you interact with environment	9.6
instant (local) deployment and testing of AI support without having to save, recompile, reset and repeat the activity	9.6
high level language specialised in authoring AI support (no JavaScript)	8.6
visual language (block-based) specialised in AI support	9



Addressing new Requirements

The material presented in this chapter is supported by the following paper: (Mavrikis et al. 2019).

6.1 Requirements Elicitation

The findings of the last evaluation of authELO revealed a new requirement. This is the need to have a more high-level language specialised in authoring feedback. An iteration to revisit the design was needed (Nieveen & Folmer 2013). In order to address this requirement we organised a requirements elicitation workshop (see 4.3.1) with two categories of participants: ICT teachers with some programming skills and EdTech students with no programming skills. The ICT teachers were asked to develop support for a number of activities in Malt+ using authELO. The teachers completed the assignment using support and under supervision. In a parallel session the group of EdTech students were asked to do the same at a much higher level. The students proposed language constructs that could be used to express concepts in that context.

Results: The outputs from the first session were analysed and a common pattern for the data acquisition part was identified. This gave the requirements for the development of the high-level language. The outputs from the second session were studied and confirmed that there is indeed a need for a block-based language for less skilled people in the sense that these people would be able to develop simple support scenarios if there was such a language available.

6.2 Making Authoring Simpler

The previous step revealed the requirements for the high-level language constructs needed to make the evidence acquisition part easier. In order to address this issue and improve authELO, a JavaScript library was developed to provide these constructs in the current system. This was an easy and cost-effective improvement to lower the entry threshold. The next step is to develop a proper language on top of JavaScript that specialises in authoring, which itself is a separate requirement.

6.3 Implementation of LFT

The third and final step of this research was an attempt to address the requirement of developing a high-level language specialised in authoring feedback. Instead of doing that, we designed and developed a tool that can be used for the specification of new languages that can be executed in a web browser. The tool is called *Lingua Franca Transformer (LFT)* and can be used to facilitate the authoring process of a new or an existing language and generates transpilers from that language to JavaScript (see 3.5). The transpilers allow real-time execution of any language in the browser with no server-side dependencies.

Result: The tool was used to develop several languages including LOGO, a version of Scheme and Java 7. These languages were used to experiment with and manipulate learning environments like the well-known LOGO turtle environment. The benefit is two-fold: the ability to speak any language in the browser context increases reusability of web components. People can use the language of preference to manipulate widgets. In particular, learning designers can teach any programming language with any learning environment. In addition we give the ability to skilled designers to develop a high-level specialised language of their preference for authoring feedback and thus increase the reusability of authELO.

The final form of the reasoning patterns is the following:

"ELE with automated support" + "authELO" → "make authoring of automated support for teaching programming easier"

AuthELO is the final outcome for the HOW component as it gives us the means to achieve the aspired value. Alongside the final product, a series of methods, processes, tools, techniques, that embrace the working principle are given as artifacts that provide added value. These artifacts can either be re-used as individual values to complement

other projects or help other researchers and practitioners as a roadmap that shows proven and effective ways to develop similar projects.

6.4 The Lingua Franca Transformer (LFT)

The programming language used to teach programming to beginners is a vehicle that carries all the concepts that need to be taught in a CS1 course. Languages may differ in how specialised they are, how close they are to the physical architecture of the machine (high or low level), what programming paradigms they support, how expressive they are and so on. All of this means that not all languages may be suitable to teach elementary programming. Depending on the approach and what the orientation of the course is, one language may be better or more suitable than the other and that is not static. As existing languages evolve and improve and new languages develop, there is always the question of which one is the most suitable one to play that crucial role in the learning process. This, obviously, is an ongoing and endless debate that has been addressed and discussed numerous times in the academic literature (Levy 1995, Kaplan 2010, Van Roy et al. 2003, Van Roy & Haridi 2003, Kruglyk & Lvov 2012, Reges 2002). We are not going to attempt to give an answer to this question here simply because there isn't one. It all depends on the criteria used by the people designing and teaching the course. Choosing between languages is not really the problem. Taking advantage of what we need from them is. What we should be able to do is to easily switch from one language to the other to address different needs and learning objectives in the process. This need becomes more apparent in microworlds where you have the ability to use constructions to learn programming. Typically, these environments are tied to a specific language. The language used in Turtle Graphics (Solomon & Papert 1976, Papert 1980) and Malt+ (Kynigos & Latsi 2007) is LOGO and the one used in Greenfoot (Henriksen & Kölling 2004) is Java. It would be really nice to have the ability to use the same environments with other languages like C# and Python. That would multiply the usefulness of those environments as it would offer the opportunity to educators to take advantage of the strengths of different languages to convey different concepts using well tested and familiar to students interfaces. Therefore, an

interesting question is: would it be possible to reuse existing learning environments with languages other than the ones they natively support?

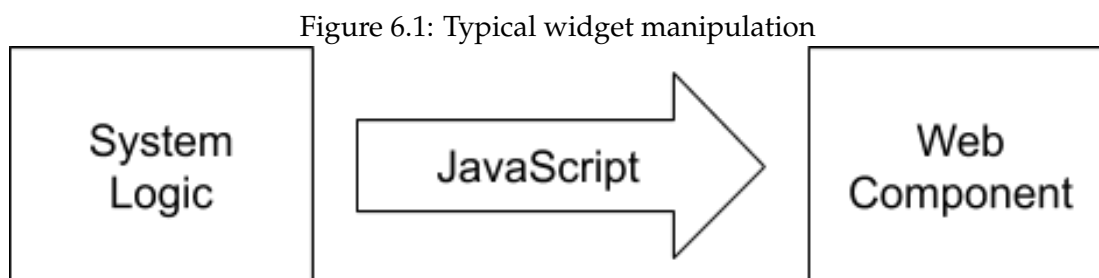
The de facto platform for deployment of educational software nowadays is the web browser. This offers a lot of flexibility in terms of development, administration and deployment but at the same time it introduces barriers in terms of what can be executed in such a platform. The most obvious concerns are the limitations of memory and processing resources in this context. The not so obvious concern but crucial in this discussion is that the machine language of the browser is JavaScript. That reveals an additional problem which is the fact that no matter how we express ourselves in terms of language the common denominator is always JavaScript and execution takes place in the underlying JavaScript engine supported by the browser. This is both good and bad. It is good because it is guaranteed that standard Javascript will always be the same in any platform and that means that no recompilation is needed for different machine and platform architectures. This guarantee of uniformity gives independence, flexibility and reduces the cost of developing solutions. On the flip side though, it is also bad because, to an extent, the limitations and particularities of JavaScript will always be a limiting factor to what can be expressed and executed in the context of a web browser.

This section presents a system called the "Lingua Franca Transformer" (LFT). As its name suggests this is a tool intended to be used for transformations from any language to JavaScript. JavaScript is the lingua franca in the browser world. If another language needs to be used, then the only way to use it is to produce the equivalent code in JavaScript. LFT is a tool that can be used to develop the definition of any language, new or existing one, and automatically generate the respective transpiler to JavaScript. The outcome of this process is a JavaScript library that can be injected into any system and make it agnostic of the language it represents. That opens the door to using existing learning environments like Malt+ with different languages like Java. It also opens the door to transforming or enhancing systems like web-based Geogebra so

that they can be used for teaching programming without any constraints in terms of language. New programming languages may be defined with the same ease as existing ones. That means that the tool can be used as a vehicle for experimentation for new languages designed to be used particularly for educational purposes. It also may be used to develop high-level specialised languages sitting above JavaScript to facilitate easier coding by people less skilled in programming. This can be especially useful in authoring environments like the one proposed in 4.3.

6.4.1 Architecture

As explained above LFT is primarily intended to be used for design of new languages and transformations between those languages and JavaScript. This is based on the premise that the web component we need to programmatically manipulate exposes an interface in JavaScript. The typical use of such a widget would require instructions expressed in JavaScript to be sent directly to the component through its API, image 6.1.



If system logic needs to be expressed in an alternative language, we need a mechanism to transform this code dynamically into JavaScript before we send it off to the component, image 6.2.

This transformation mechanism needs to know the formal specification of the language that is to be used as input. That means that the syntax rules need to be available so that a parser that understands the language can be developed and used to process the text,

Figure 6.2: Widget manipulation with other languages

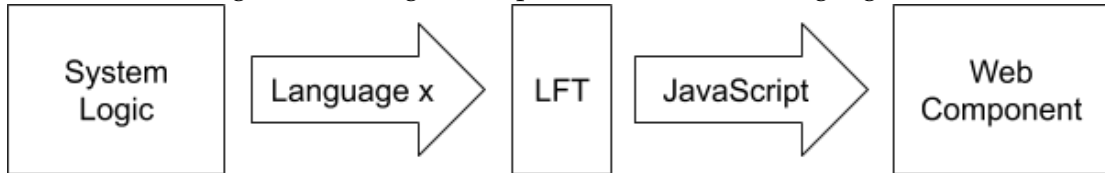
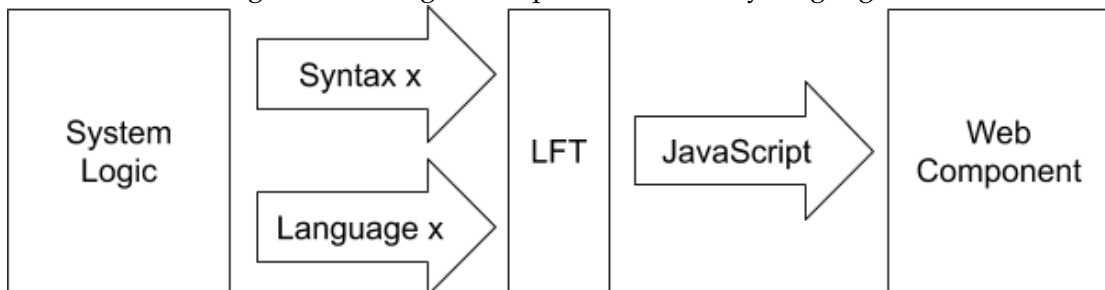


image 6.3.

Figure 6.3: Widget manipulation with any language

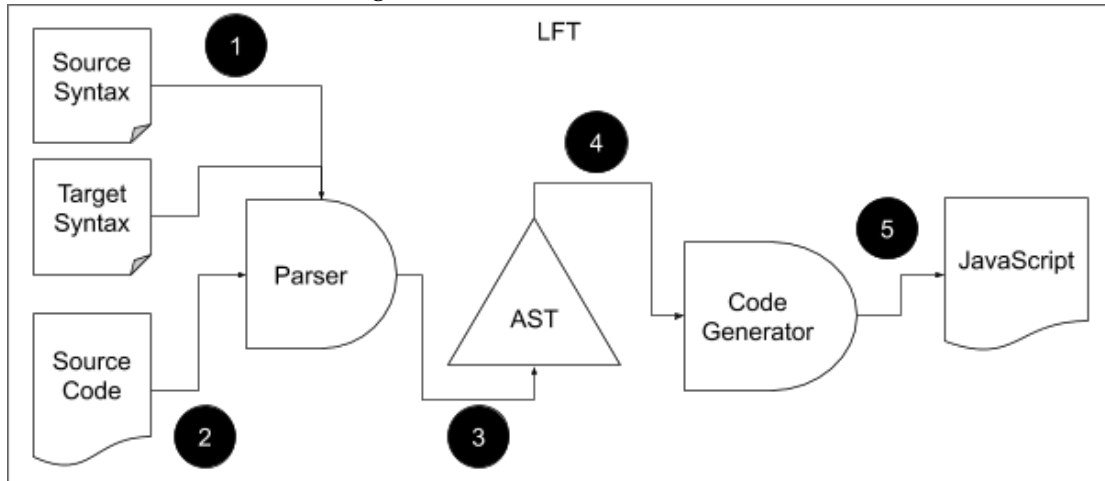


That changes dramatically the way we programmatically interact with components. Instead of just using a component by sending instructions in its native language, we now send instructions along with the syntax rules of the language that these instructions are written in. If this process is fully dynamic and there are no assumptions about the language being used as input, then that gives us a great deal of flexibility in the sense that the language may be changed even in the middle of a session without that affecting anything in the process. Even this extreme use case is perfectly possible with LFT.

LFT, image 6.4, accepts the formal specification of a language in the form of syntax rules and generates a parser for that language. It also accepts the syntax rules for the output language so that it can generate the new code in the appropriate format. All of this information is embedded in the parser. The parser can then be used to process the given code and produce an equivalent representation in the intended form. Typically, in LFT this form is an Abstract Syntax Tree (AST) that conforms to the ESTree ¹

¹<https://github.com/estree/estree>

Figure 6.4: The LFT Architecture



specification, formerly known as SpiderMonkey AST. This is not mandatory but rather a convention used for convenience. AST is a tree-like representation of the syntactic structure and the language constructs found in the source code. It is abstract in the sense that it does not depict every little detail found in the syntax or the semantics of the code but only the structure. The ESTree specification is one of the well-known, standard formats for JavaScript ASTs. In LFT the ESTree AST serves as the common language denominator because its specification is much simpler than that of JavaScript itself. It also allows easier manipulation of the code and automated conversion into JavaScript via code generators. The final step is to dynamically evaluate the JavaScript code and execute the instruction in the component. LFT is not restrictive to using a particular tree representation as an intermediary form. An alternative scenario that converts one language directly into another is possible. If, for example, the native API exposed by a component is in LOGO and there is no code generator based on ASTs for LOGO, it is perfectly possible to generate a parser that transforms any language directly into LOGO. The difference is that this process will be more difficult because the specification of the target language will have to be given in full and inserted into LFT.

6.4.2 The Language Specification Syntax

In formal language theory the syntax rules for the language specification is called formal grammar. That is a set of production rules that dictate how valid strings from the language alphabet can be formed. There are different formalisms that can be used to express formal grammar. LFT utilises the formalism called Parsing Expression Grammar (PEG). PEG formalises a language in terms of an analytic grammar and that means that syntax rules correspond more directly to the structure and the semantics of the parser that is generated for that language. PEG is a generalisation of the Top-Down Parsing Language (TDPL) which is a highly minimalist analytic grammar formalism developed for top-down parsers. PEG is designed specifically to accommodate the needs of programming language and compiler writing and is considered more powerful than traditional LL and LR formalisms. LFT uses a parser generator implemented in JavaScript and called PEG.js. This implementation accepts syntax rules in PEG and allows inline statements expressed in JavaScript. That means that we can embed syntax rules for the output language in the same document. The benefit from that is that the parser derived from this process is able to reshape the resulting AST and/or transform it to the syntax of the output language. Without that, we would need another software component to process the resulting tree and reshape it so that it conforms with the ESTree specification. The following is a very simple example of a language specification given in PEG:

```
text = words:(w:word space? {return w;})* {return words;}
word = letter+
letter = [a-zA-Z0-9]
space = " "
```

Parsing starts with the rule given first. The initial rule is called **text** and references two other rules named **word** and **space** respectively. These rules need to be defined as well. The rule **word** is defined as one or more letters. **Letter** is defined as any alphabetical or

numerical character (English alphabet). **Space** is defined as the whitespace character. Going upwards, **text** is defined as any sequence of **word** tokens optionally followed by a **space**. The JavaScript snippets give instructions to the parser to return only the matched word tokens, not the spaces. If this specification is given to LFT along with the text "hello world" as input, the output will be the following tree:

```
[
  [
    "h",
    "e",
    "e",
    "l",
    "o"
  ],
  [
    "t",
    "h",
    "e",
    "r",
    "e"
  ]
]
```

6.4.3 The Tool

The tool is designed to facilitate the authoring process of syntax rules for new and existing languages in PEG. The assumed target language is the ESTree specification. The interface is split in panels organised as tabs. The first tab shows the editor that can be used to author the syntax rules for the input language - see figure 6.5. On the right

there is another editor that can be used to test these rules with some text expressed in the language being specified. If the rules are well formed and the input is valid the resulting AST is displayed in another frame. This part of the interface can be used for the specification of any output language and ESTree output is not assumed. If the target language is ESTree then a more careful inspection of the output is needed. For that there is an additional tab that allows the author to perform a comparative analysis of the output between the statements given in the new language and JavaScript. The comparisons tab gives two editors, one for each language - see figures 6.6, 6.7. The author is supposed to provide equivalent statements in both languages to express the same operation and examine carefully the generated ASTs.

Figure 6.5: The LFT Editor

The screenshot shows the Lingua Franca Transformer (LFT) Editor interface. At the top, there's a title bar 'Lingua Franca Transformer (LFT)'. Below it are four tabs: 'Editor', 'Comparisons', 'Testbed', and 'AST'. The 'Editor' tab is active, displaying a code editor with language rules. The rules are defined in a JSON-like structure, including binary expressions, update expressions, and function declarations. A green status bar at the bottom of the editor says 'rules are defined without problems'. To the right, the 'Language Text' field contains the following code:

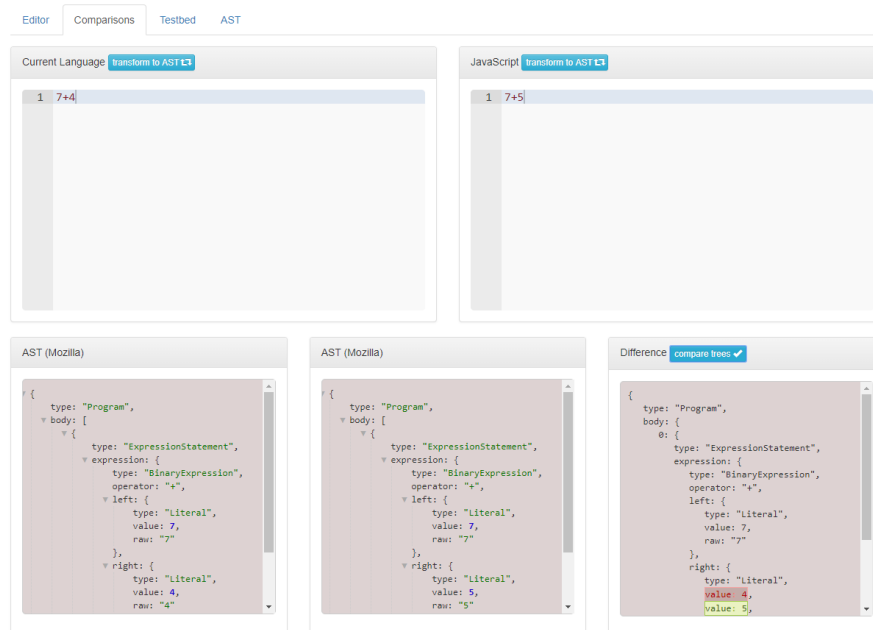
```
1 to shape :sides :length
2 localmake "turn 360 / :sides
3 repeat :sides [forward :length right :turn]
4 end
5
6 to complexshape :angle :sides :length
7 localmake "turns 360 / :angle
8 repeat :turns [shape :sides :length right :angle]
9 end
10
11 complexshape 20 6 100
12
```

Below the 'Language Text' field, another green status bar indicates 'code evaluates without problems'. The 'AST' tab is also visible, showing the resulting Abstract Syntax Tree (AST) for the input code. The AST is a JSON object representing the structure of the code, including the program, function declarations, and the function body.

If the ASTs don't match up then there is a mistake in the PEG syntax. An automatic comparison is performed and differences are highlighted in a third frame.

The next tab is called Testbed and is designed to give an idea of how the newly defined

Figure 6.6: The LFT Comparisons Section



language could be used to manipulate a turtle in a Microworld. Both turtle graphics and text I/O interfaces are available and directly usable by the new language. The screenshot in figure 6.8 shows a complex shape drawn by LOGO code dynamically transpiled to JavaScript in LFT. The LOGO language specification was developed from scratch in LFT as a proof of concept.

As the transformation takes place, this part of the interface provides real time information about the well-formedness of the syntax rules, the validity of the source code and the state of the operations that take place, figure 6.9.

In addition to all the above an interactive AST for the code given in the testbed is given in the last tab, figure 6.10:

6.4.4 Implementation Details

LFT operates as an authoring tool. Its primary task is to allow an author to build an effective transpiler from one language to another. This task entails the development of a correct specification for an input language along with correct transformation rules

Figure 6.7: The LFT Comparisons Section

Lingua Franca Transformer (LFT)

Editor
Comparisons
Testbed
AST

Current Language transform to AST ↗

```

1 to shape :sides :length
2 localmake "turn 360 / :sides
3 repeat :sides [forward :length right :turn]
4 end
5
```

JavaScript transform to AST ↗

```

1 function shape(sides, length)
2 {
3   var turn = 360 / sides;
4   for(var i = 0; i < sides; i++)
5   {
6     forward(length);
7     right(turn);
8   }
9 }
```

AST (Mozilla)

```

{
  type: "Program",
  body: [
    {
      type: "FunctionDeclaration",
      id: {
        type: "Identifier",
        name: "shape"
      },
      params: [
        {
          type: "Identifier",
          name: "sides"
        },
        {
          type: "Identifier",
          name: "length"
        }
      ]
    }
  ]
}
```

AST (Mozilla)

```

{
  type: "Program",
  body: [
    {
      type: "FunctionDeclaration",
      id: {
        type: "Identifier",
        name: "shape"
      },
      params: [
        {
          type: "Identifier",
          name: "sides"
        },
        {
          type: "Identifier",
          name: "length"
        }
      ]
    }
  ]
}
```

Difference compare trees ↕

```

},
kind: "var"
},
1: {
  type: "ExpressionStatement",
  expression: {
    type: "ForStatement",
    expression: {
      type: "CallExpression",
      callee: {
        type: "FunctionExpression",
        id: null,
        params: [],
        defaults: [],
        body: {
          type: "BlockStatement"
        }
      }
    }
  }
}
```

Figure 6.8: The LFT Testbed Section

Editor
Comparisons
Testbed
AST

run ✓ clear screen clear text home

```

1 to shape :sides :length
2 localmake "turn 360 / :sides
3 repeat :sides [forward :length right :turn]
4 end
5
6 to complexshape :angle :sides :length
7 localmake "turns 360 / :angle
8 repeat :turns [shape :sides :length right :angle]
9 end
10
11 complexshape 20 6 100
12
```

Language Editor

Text I/O

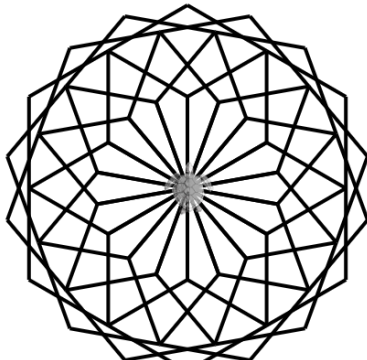
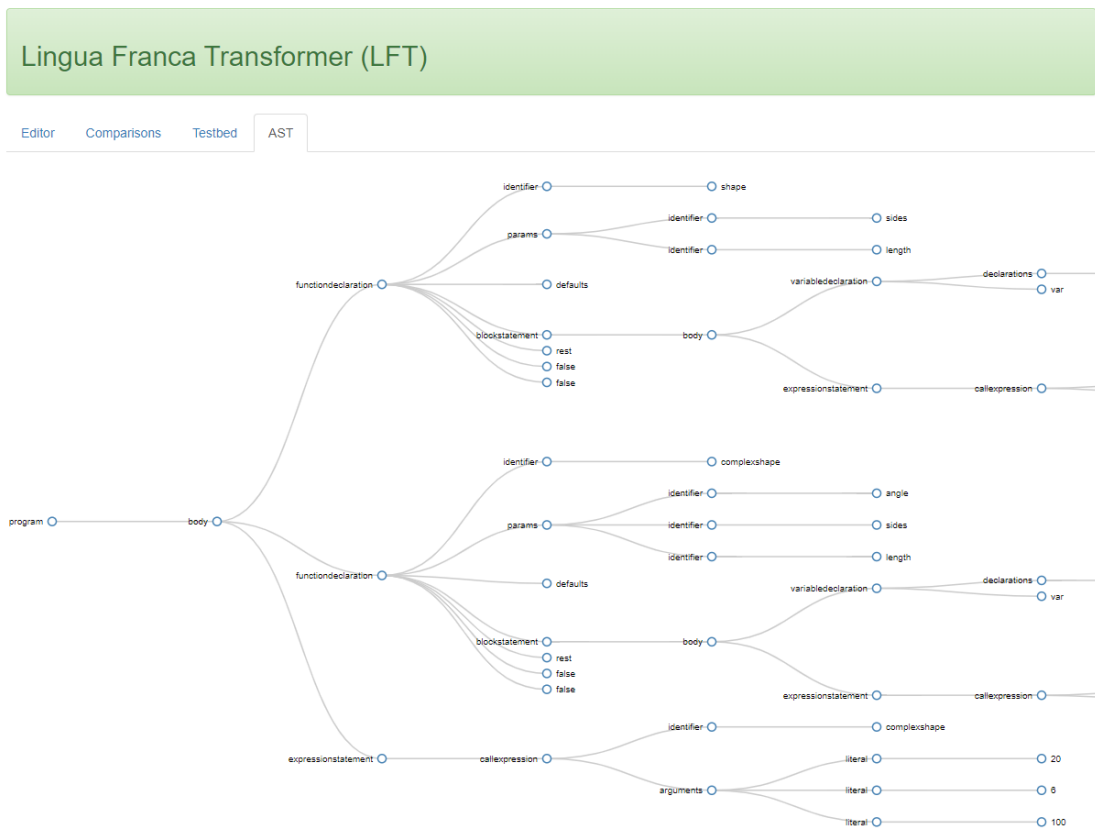


Figure 6.9: LFT Real-time Updates

Syntax Rules		Source Code	
Time	Message	Time	Message
23:09:30.457	language editor change	12:53:47.264	ast created without problems
23:09:30.457	language editor change		[{"start": {"row": 0, "column": 0}, "end":
JS Conversion		Execution	
Time	Message	Time	Message
12:53:47.270	JS code generated without problems	12:53:47.318	js code evaluated without problems

Figure 6.10: LFT Interactive AST Visualisation



for an output language. If the syntax rules given are valid and the process is successful the output is a parser. In the context of LFT the parser is a JavaScript object that gets generated dynamically by a third party component found in the PEG.js² li-

²<https://pegjs.org/>

brary. This component is a parser generator that conforms to the PEG formalism as presented in 6.4.2. Once the parser is created it can be used to parse code expressed in the input language and generate code expressed in the output language dynamically. LFT was designed to allow transformations between any two languages but its primary goal is to allow the creation of transpilers to JavaScript. As stated previously the purpose for this is to enable the use of different languages in the context of web browsers. Therefore, the expected target language is typically an Abstract Syntax Tree (AST) that conforms to the ESTree³ specification. This specification is the most common and standard format for JavaScript AST. Once the parser is created the next step is to enter a sample text in the input language and let the parser generate the respective AST for inspection. The AST is formatted appropriately and placed in a graphical control to enable visual inspection. A third-party library called JSON Viewer⁴ is used for the formatting. Further tests are done in the 'Comparisons' tab and that entails the automatic comparison and visual inspection of ASTs that are supposed to be identical. The test at that stage is to give two equivalent statements, one in the input language and the other in JavaScript and check if the resulting ASTs are exactly the same. Parsing the JavaScript code is done by a tool named Esprima⁵. Automatic comparison is performed by a third party component called objectDiff⁶. This checks both ASTs, detects discrepancies in the structures and displays visual indicators of the differences. This allows for a much more succinct testing of the parser conformity with the ESTree specification. The final and ultimate test is to evaluate the parser in the challenging context of a microworld. This entails writing code in the input language to control and manipulate a turtle in a turtleworld. This environment is an own component built specifically for this project. The environment was developed in HTML5, JavaScript and Raphael⁷. The API exposed by this microworld expects to be used in JavaScript.

³<https://github.com/estree/estree>

⁴<https://github.com/abodelot/jquery.json-viewer>

⁵<https://esprima.org/>

⁶<https://github.com/NV/objectDiff.js>

⁷<http://raphaeljs.com/>

Therefore, the ESTree AST generated by the parser must be transformed to Javascript before it gets executed. For this a third-party component named *Escodegen*⁸ is used. The visualisation that shows the interactive AST for the code given in the testbed is done with the third-party library named *D3*⁹. Finally, all the text editors used in this project are implemented using the third-party library called *Ace*¹⁰.

⁸<https://github.com/estools/escodegen>

⁹<https://d3js.org/>

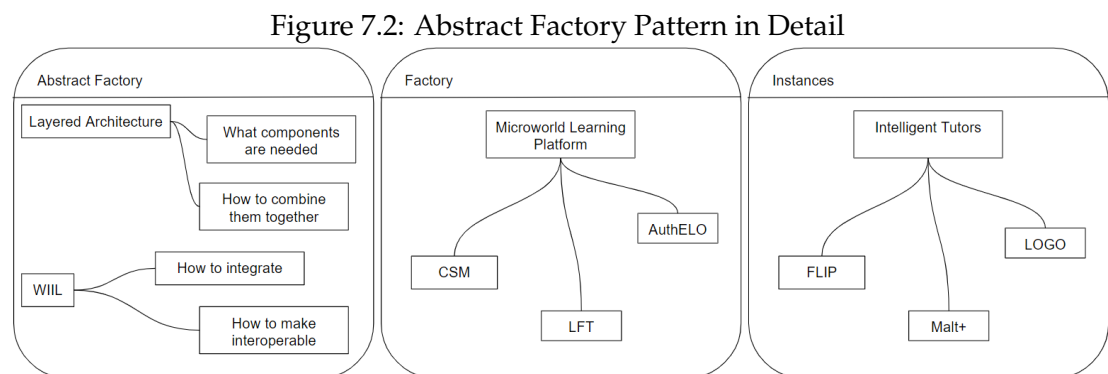
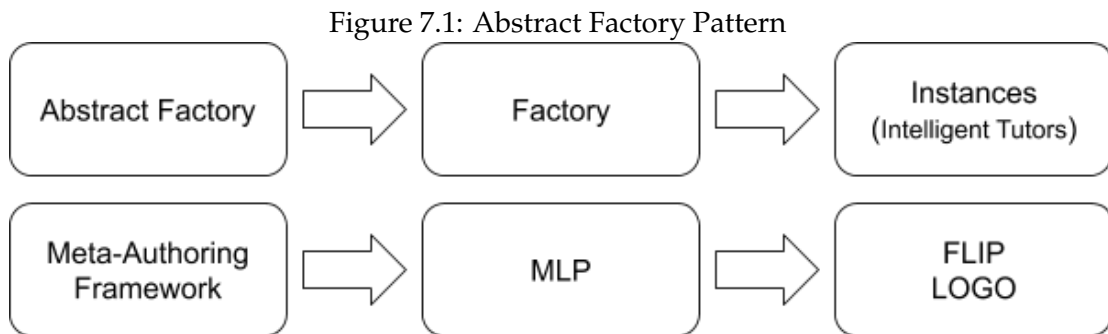
¹⁰<https://ace.c9.io/>



The Microworld Learning Platform

7.1 A Conceptual Overview

Overall, the artifacts produced in this thesis can be combined in a way that follows the abstract factory software design pattern - see figure 7.1.



As depicted in figures 7.1 and 7.2, we start from an abstract factory. This is a combination of the layered architecture and WIIL. This part acts as a model that shows the conceptualisation of a system that represents an intelligent tutor. It describes the logical components expected in such a system and shows how these components could be combined together to achieve the required functionality. Apart from the conceptual part, there is also the technical specification of how these components can integrate with a platform and interoperate with each other. This part is collectively called Meta-Authoring Framework (MAF). This framework can be used to create systems that act as factories of instances that represent intelligent learning environments. A factory is

a software application that utilises LFT to create language-neutral programming tutors, or existing learning environments like Malt+ along with AuthELO to generate enhanced tutors equipped with intelligence and adaptability. Concrete instances of WIIL are used to integrate these components together and make them interoperable so that authors can join them together to synthesise new environments with minimal effort and cognitive load. The factory presented in this thesis is Microworld Learning Platform. This factory can be used both as a development and as a deployment platform for enhanced programming tutors. The outcome of the authoring process is typically an instance of an intelligent tutor equipped with functionality that corresponds to some or all of the layers prescribed in the architecture. The instances presented in this thesis are FLIP, Malt+, GeoGebra Coding and the LOGO tutors.

7.2 The Platform

This chapter presents a platform that has been developed to demonstrate how the knowledge discovered and the methods, techniques and technologies proposed in this thesis can be combined together to allow learning designers, technologists and teachers to synthesise intelligent, automated tutors for programming with ease using non-standard and heterogeneous web components. As explained in the previous section, this platform conceptually represents a factory of programming tutors. It encompasses all the ingredients prescribed in the Meta-Authoring Framework (MAF) which is the conceptual aspect of it. All the knowledge required to define appropriate combinations and join the components together is embedded into this platform. Therefore it inherently supports the integration and interoperation of diverse web components in ways that satisfy the layered architecture in order to develop automated tutors enhanced with extra knowledge and functionality. Existing knowledge about common misconceptions, or newly developed knowledge for certain tasks may be integrated with learning environments to provide adaptability and intelligence. All of this is done

with minimal development and administrative overhead. Once web components are integrated with the platform, the author can perform all of these operations seamlessly with minimal cognitive load and effort. This factory can be used both as a development and as a deployment platform for enhanced programming tutors. The outcome of the authoring process is typically an instance of an intelligent tutor equipped with functionality that corresponds to some or all of the layers prescribed in the architecture. Currently, the platform supports the development of instances based on FLIP, Malt+, GeoGebra Coding and LOGO tutors.

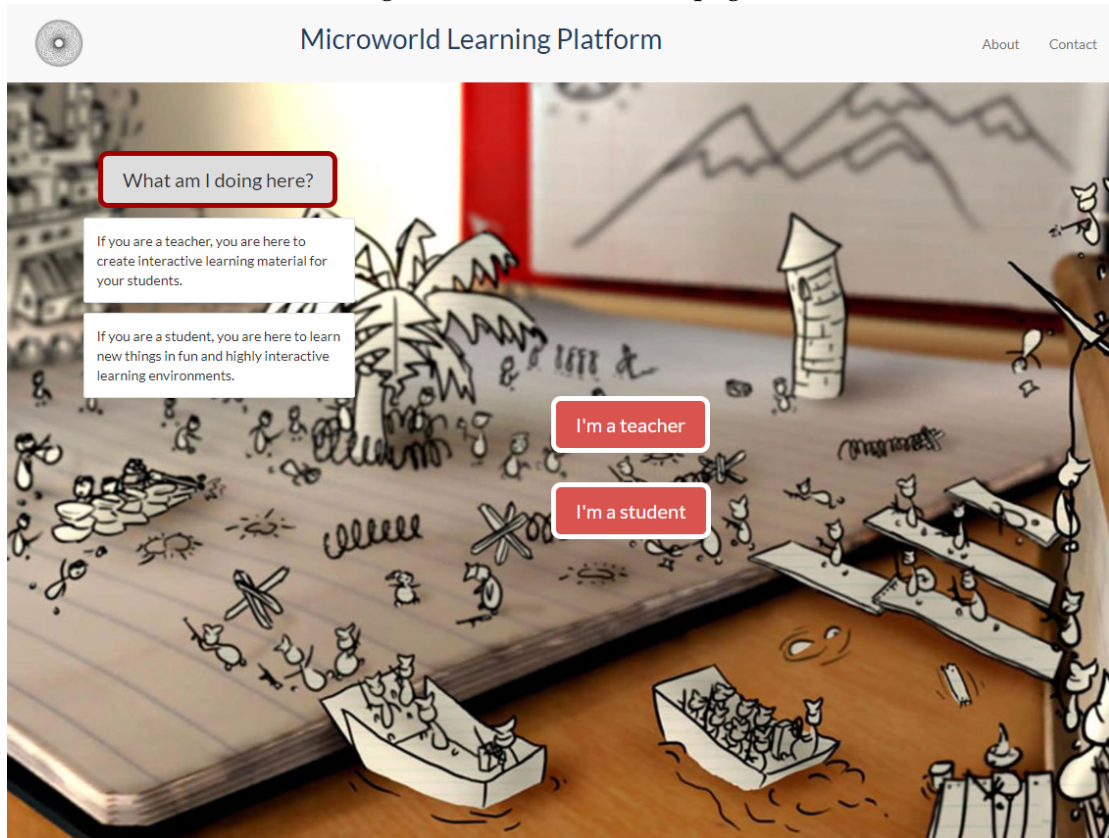
7.3 The Basic Workflow

Since this is both a development (authoring) and a deployment platform, the front-page is the starting point for both teachers and students - see figure 7.3. There is a rudimentary security infrastructure in place to allow learning designers authenticate themselves and authorise operations in the platform.

Authors are assumed to be teachers and therefore initially they are presented with their profile screen - see figure 7.4. The main parts in that screen are learning activities, classes and students. The platform allows the teacher to create classes, individual students and allocate students to classes. It also allows them to create instances of learning activities based on learning environments supported by the system. These instances, once created, may be customised so that they can present some initial construction that facilitates best the learning scenario planned for the session and the intended learning outcomes. Learning environments and their instances are integrated with the system through WIIL.

Once a learning activity is ready for deployment, the author can simply select a class of students and drag and drop both entities to the bottom of the page under the section 'Current Session'. That automatically shows the students associated with the selected class, figure 7.5.

Figure 7.3: The MLP Frontpage



In the meantime students may be instructed to use the same interface to access the activity player, figure 7.6.

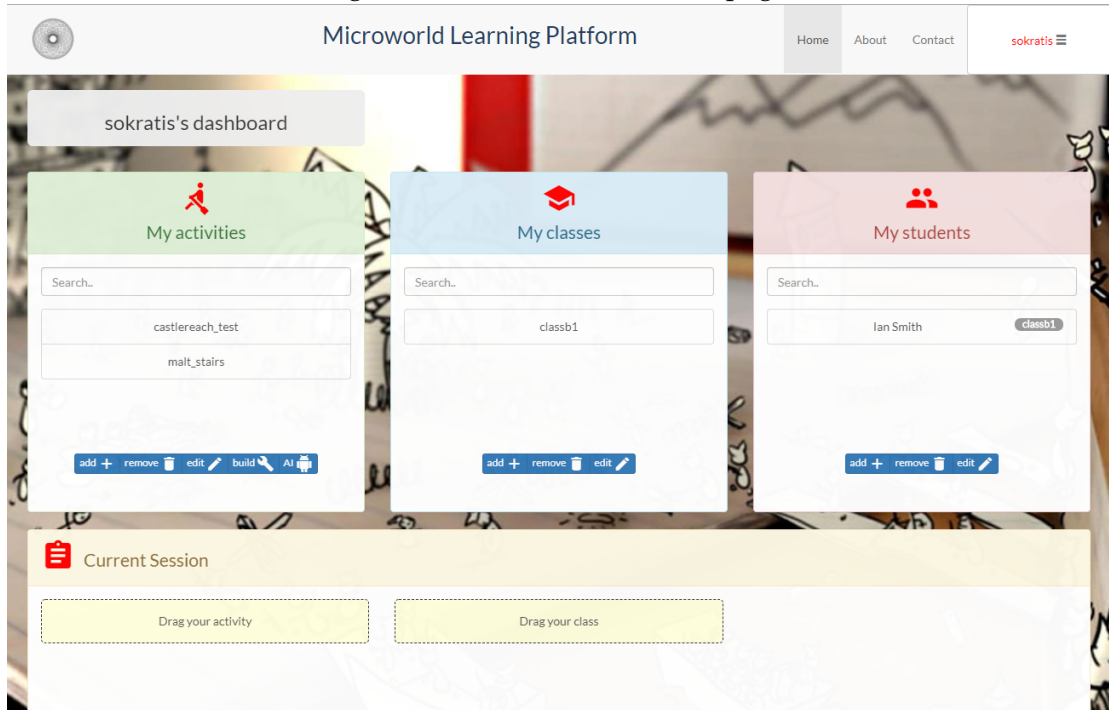
Through this interface they select their teacher, the class they belong to and finally their name in that class. There is no security here. The intention is to provide hassle-free access to the activity. They then press a button to connect their player to the platform, figure 7.7.

Once connected the interface gets updated immediately at both ends (student and teacher), figure 7.8.

Once all the students are connected the teacher deploys the activity with a push of a button, figure 7.9.

This action presents the activity to the student terminals and the session commences.

Figure 7.4: The Author's Homepage



The students work with the task given and the system processes automatically their interactions to adapt the activity and provide support if needed. The platform provides two buttons at the top of the page for that - see figure 7.10. The button **check** may be used by the student to ask for an evaluation of the work done so far and an estimate of how much is completed. The button **help** may be used to ask the system for assistance on the process and/or the concepts or skills involved.

Help is provided through an intelligent assistant that presents itself as an owl. The assistant is initially hidden and appears only after the user requests help. It responds to these requests with cues, instructions, suggestions etc. This assistant is not part of the original Microworld, in this case Malt+. It is dynamically inserted by the platform after the learning environment is integrated with authELO. AuthELO is used to develop task-specific support for the activity and the means to communicate that support to the students. The two components operate seamlessly as one and the mechanics of the

Figure 7.5: Activity Deployment

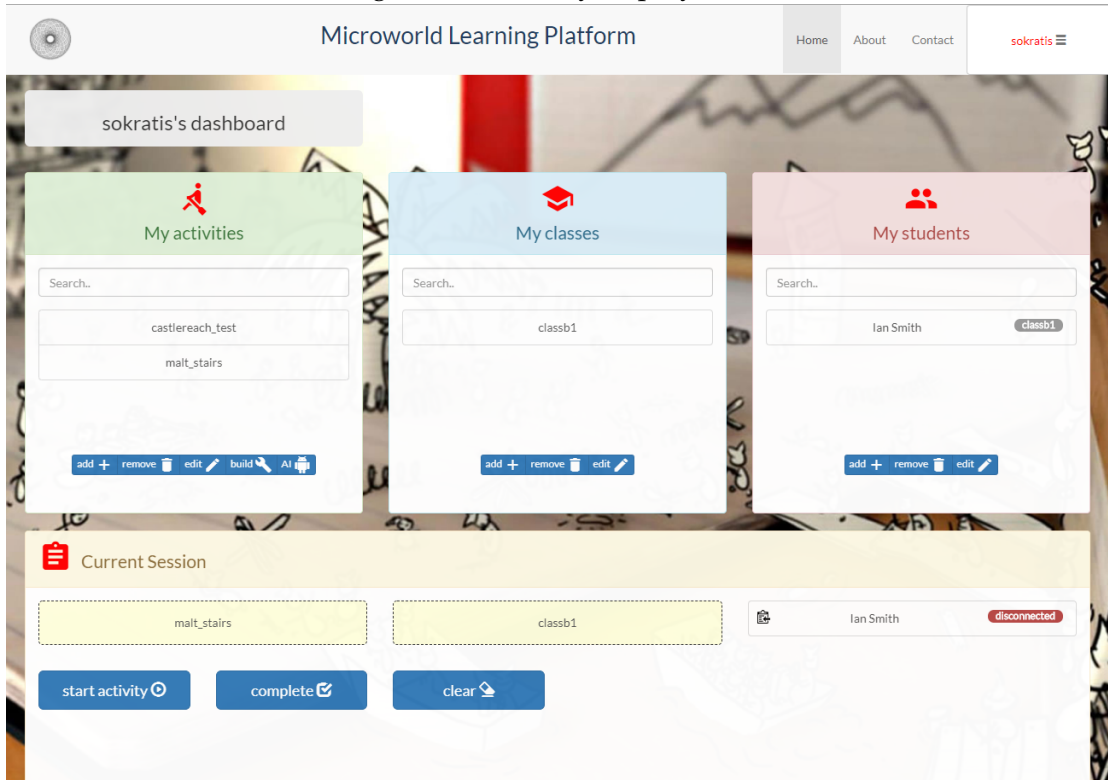
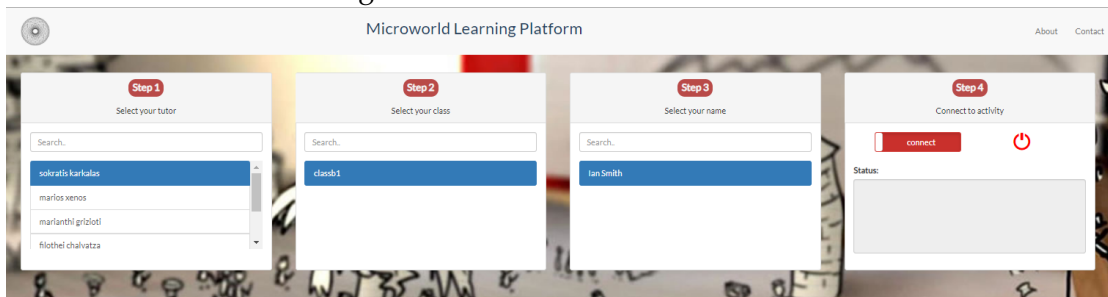


Figure 7.6: The Students' Terminal



whole process are completely transparent to the end users, students and authors. All the user interactions get intercepted by the learning environment and passed through WIIL to authELO for processing. If help is requested, authELO, behind the scenes, evaluates the facts along with the available rules and activates the part that generates the answer. The answer is then communicated back to the user through the assistant. This process is completely transparent and hidden from the author or even the inte-

Figure 7.7: Connecting to the System

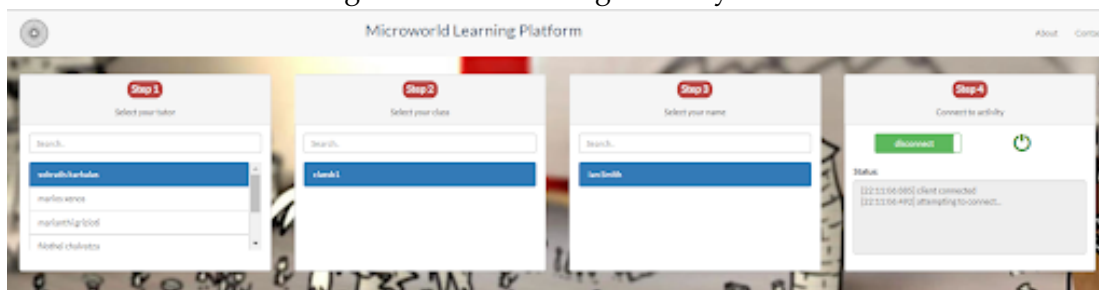
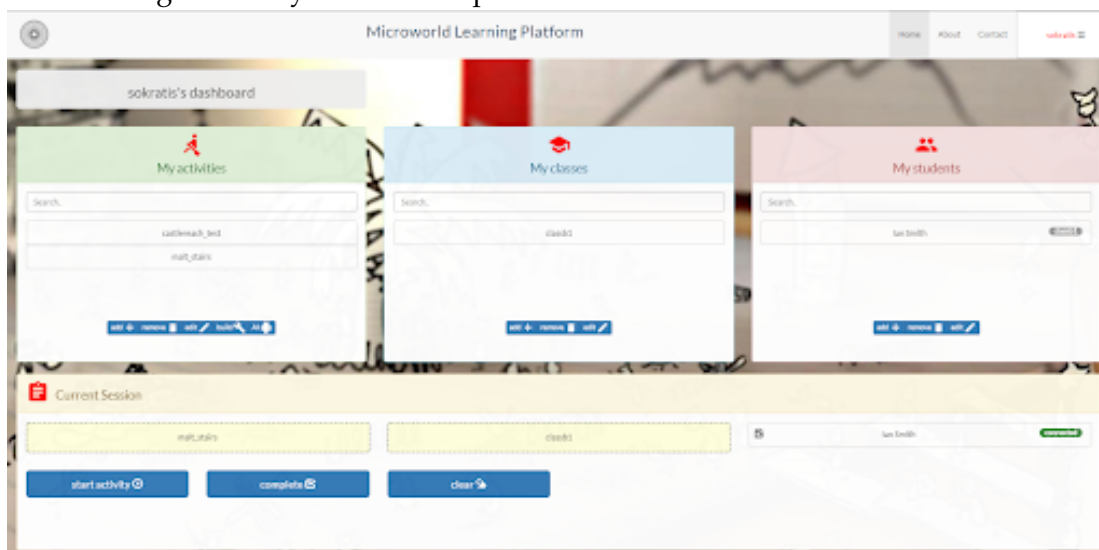


Figure 7.8: Synchronous Operation between Teacher and Students



grator of the components. Integration is merely a matter of registering the component with the platform and setting up its interface. After that all the rest takes place automatically by the platform itself. That means that the whole process is performed in exactly the same way regardless of which learning environment is used and which component is used to enhance it. Once the time for the activity elapses the teacher terminates the session by pressing the 'Complete' button. That removes the activity from the student terminals and the display switches back to the initial screen. The teacher can then go back to the activity card and download the data related to the session for analysis. This includes all the activity indicators generated by the learning environment during the session.

Figure 7.9: Activity in Student Terminal

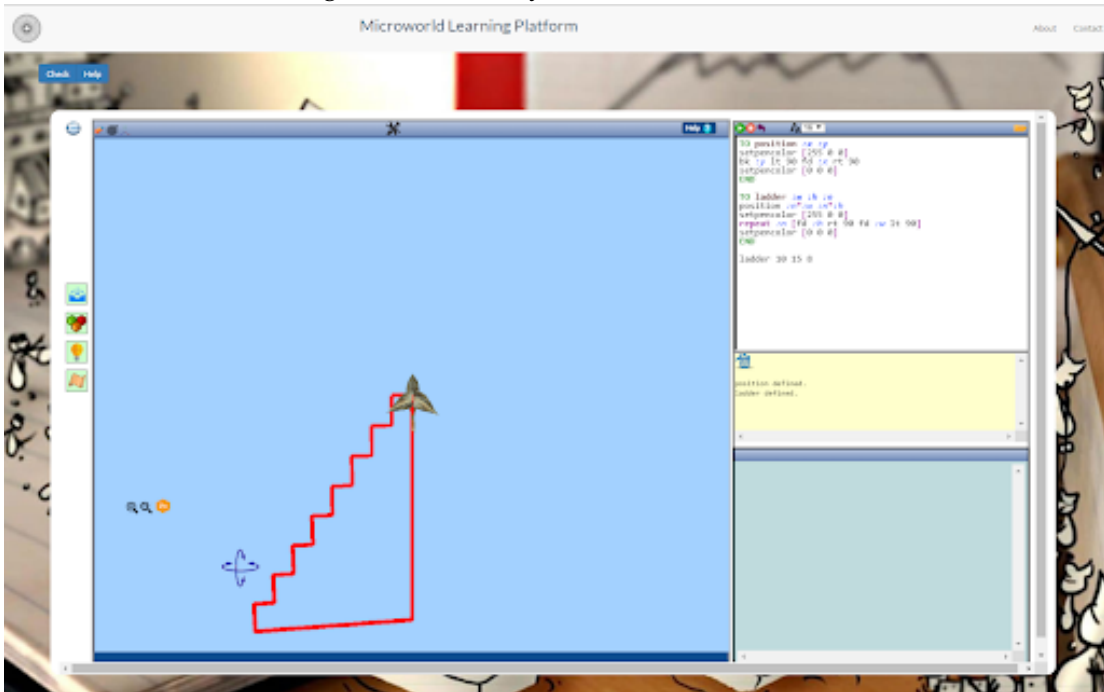
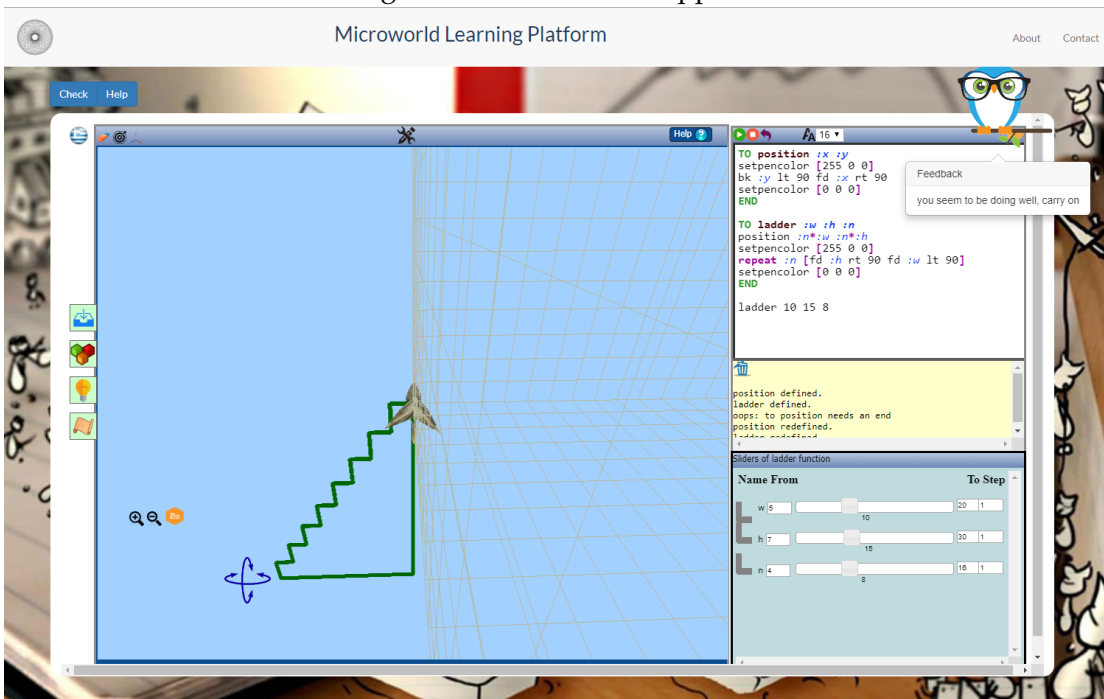


Figure 7.10: Real-time Support



7.4 Creating and Enhancing Instances of Learning Components

Learning environments that have already been integrated with the platform serve as factories for instances of learning objects. Under the 'My activities' section the button **add** can be used to create an instance, figure 7.11. This instance can then be further developed so that an initial construction can be added or the environment can be customised appropriately for the activity. For this the button **build** must be used. During the customisation phase there is a lot of information generated. All of this data is generated in the native environment of the instance itself and is then passed back to the platform through WIIL. The hosting platform stores this data in an internal database and associates it with the instance. If and when this instance ever gets deployed, this information is retrieved from the database and sent through WIIL to the student terminal so that the learning component can be properly initialised for the activity that follows. The same process takes place even when the component gets loaded in the authoring environment for editing.

Enhancing the learning component with intelligent support and adaptivity can be initiated with the button **AI**. This action loads the component and initialises it in the same way that it would instantiate it in the student's terminal. The difference is that in this case the component is loaded from within the authoring environment of authELO, figure 7.12. AuthELO communicates with the component and asks for component-specific details like the type of elements available in the environment and the respective events associated with them. All of this information is then used to dynamically create the interface for the 'Logging' screen where the author can specify with rules what elements are of interest and which events need to be intercepted, figure 7.13. Once that is done, the user can go over to the widget tab in authELO and check whether the user activity for the intended elements is properly logged. Interacting with the environment should display new entries under the **Activity Log** section. The next step is to go over to the **Analysis** tab to prepare the data needed for authoring the feedback

and assessment rules, figure 7.14. Once all the required data is processed and given the expected form, we go over to the Feedback tab to insert the rules for feedback and assessment, figure 7.15. These rules refer to the outcome of the analysis to evaluate the current state of the learner and generate support. If long messages are needed to communicate the consequent part of the rules, then the Messages tab may be used to create named variables to facilitate easy management and referencing of those messages, figure 7.16. This part is expected to be heavily used by teachers with very limited IT skills that want to refine enhancements prepared by learning technologists. The last step is to check the rules for validity, figure 7.17. This is done in the Widget screen. The author does the activity like a student and tries to generate feedback related to the rules they just added to the system. If everything seems to function properly the author can save the rules from the authELO environment and embed them into the activity.

Figure 7.11: Creating a new Activity

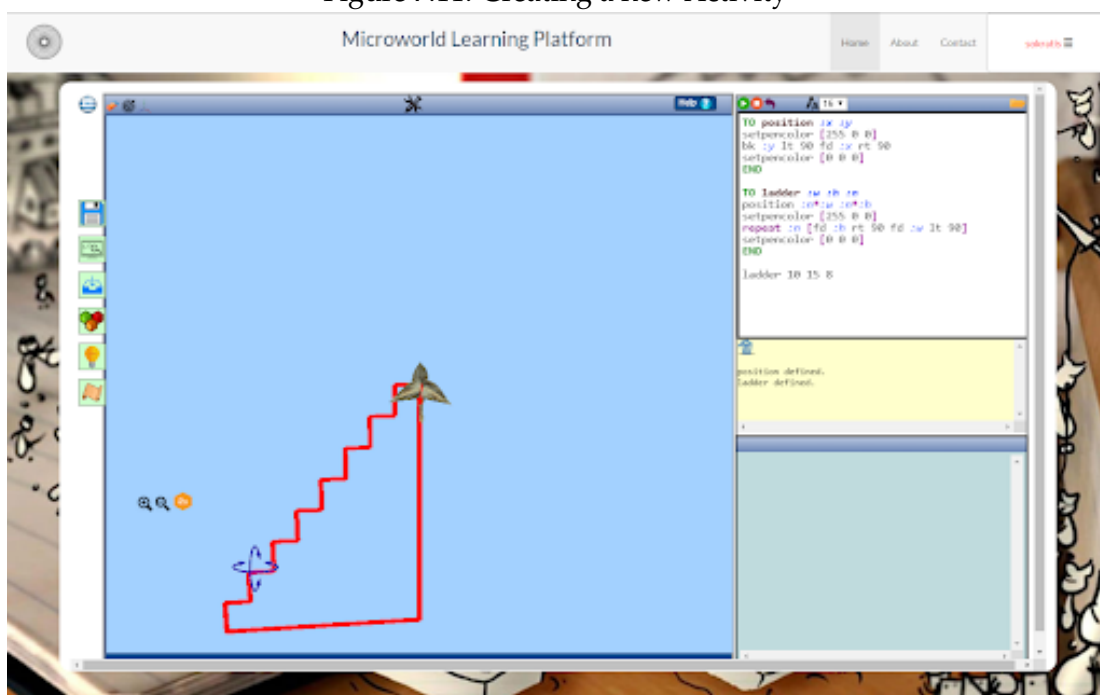


Figure 7.12: Enhancing an Activity with Automated Support

The screenshot displays the Microworld Learning Platform interface. The main window shows a 3D environment with a blue grid floor and a red ladder. A small character is standing on the ladder. The interface includes a top navigation bar with 'Home', 'About', and 'Contact' links, and a 'sokratis' logo. Below the main window, there is an 'Activity Log' table and an 'Action' panel.

Activity Log

Timestamp	Element	Event	Name
0:42:02 pm	camera	rotate	camera1
0:42:01 pm	camera	rotate	camera1
0:41:59 pm	camera	rotate	camera1

Action

```
{
  id: "camera1",
  type: "camera",
  v: state {
    view_type: "3dview",
    track_count: 0,
    event_count: 2,
    event_last_count: 2,
    tracking: true
  },
  event: "rotate"
}
```

Figure 7.13: Configuring Logging Rules

The screenshot displays the Microworld Learning Platform interface for configuring logging rules. The interface is divided into several sections: 'Element Types', 'Event Types', 'Logging Rules', and 'Active Rules'.

Element Types

- camera
- trace
- editor
- slider
- authelo

Event Types

- proc_definition
- logo_execution
- correctParameters
- parameter_count
- procedure_count_checked
- code_min_changed
- keyword_found
- structure_found

Logging Rules

editor / correctParameters [name]

insert remove clear

element / event [name]

Active Rules

Element	Event	Name
camera	rotate	-
authelo	check	-
authelo	help	-
trace	click	-
slider	change	-
editor	proc_definition	-
editor	logo_execution	-
editor	correctParameters	-
editor	parameter_count	-
editor	procedure_count_checked	-
editor	code_min_changed	-
editor	keyword_found	-
editor	structure_found	-

Figure 7.16: Editing Feedback Messages

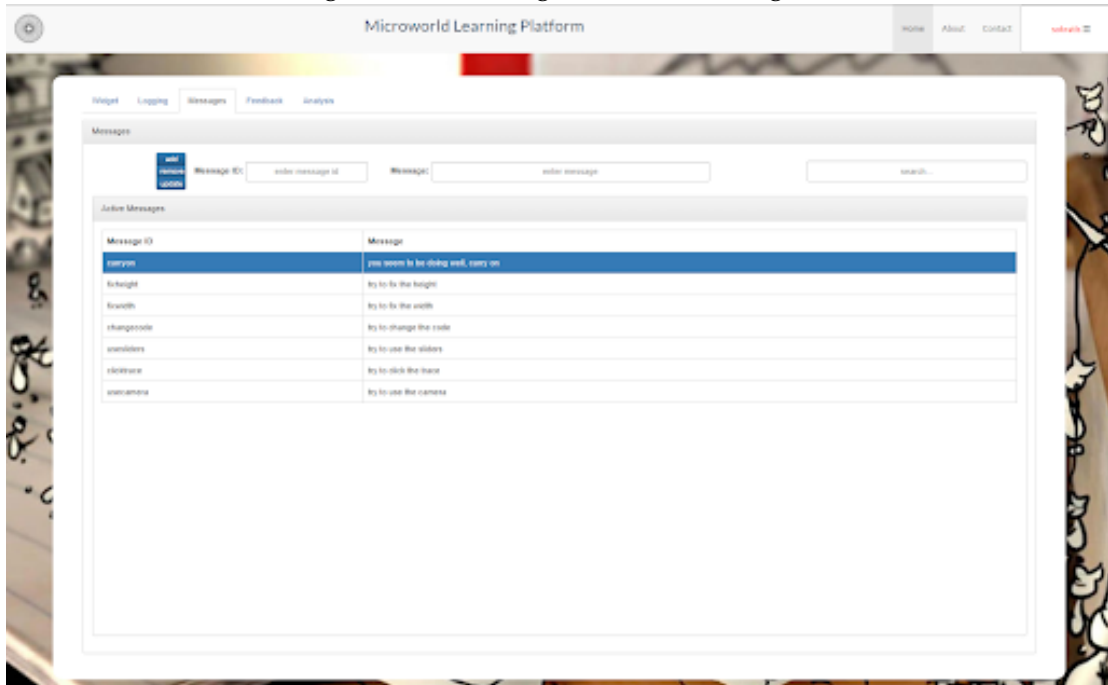
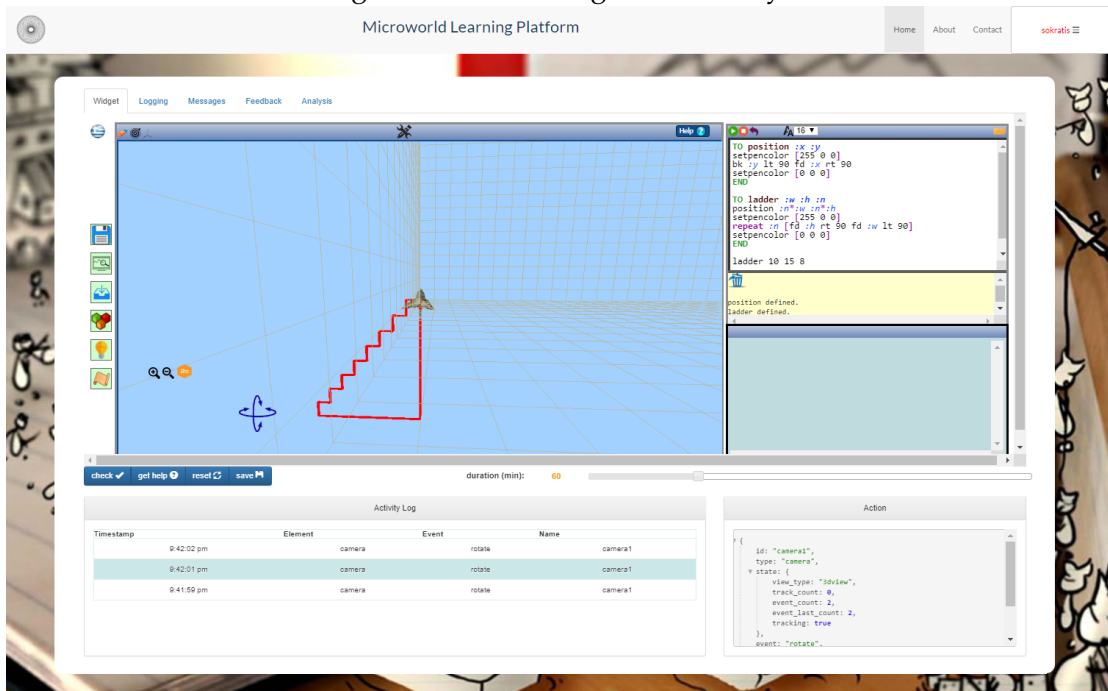
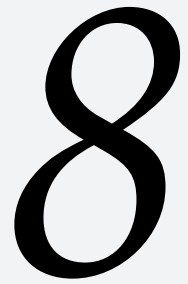


Figure 7.17: Checking Rule Validity





Contributions

The contributions achieved in this work are presented as individual components in sections 8.1., 8.2 and 8.3. The first two sections correspond to the research objectives stated in section 1.2 and the last one covers non-categorised contributions. In section 8.4 there is an overview of the project that shows how all these components are combined together in a coherent whole that gives more added value than its parts.

8.1 Part 1 - Facilitate Reuse

In this part there are two contributions:

WIIL The first one is a technique that enables seamless integration and interoperability of web components with learning platforms with minimal administrative and development overhead. The name of the technique is Web Integration & Interoperability Layer (WIIL). The technique overcomes diversity and heterogeneity of components and provides a cost-effective and efficient way to reuse existing and new technologies that do not comply with integration and interoperability standards.

LFT The second one is an authoring tool that can be used to define languages other than JavaScript that may be executed on a web browser. The premise here is that the web browser is the defacto platform for the deployment of educational software. The tool enables the development of language-specific compilers that do the transcoding from any language into JavaScript which is the machine language of web browsers. The name of the tool is Lingua Franca Translator (LFT) and it offers a fully fledged IDE that can be used to simplify the development of the syntax rules needed to specify any language. This language can then be exported and used in any web-based environment to take advantage of all the functionality that may be exposed by this environment. In other words if there is a component that does 3D dynamic geometry in the browser, then this component, without any changes, may be manipulated by any

of the languages developed with LFT. That maximises the usefulness of existing web components and language neutrality gives us the ability to utilise them for teaching programming with any language.

Definition of programming languages can also be helpful with authoring automated support. High-level specialised languages may be defined to assist authors express themselves in a more succinct, efficient and meaningful way when authoring intelligent support for learning activities. This can be a significant enhancement in authELO and satisfy the relevant requirement that emerged during the evaluation process.

8.2 Part 2 - Simplify Authoring

In this part there are three contributions:

Common Student Misconceptions This part presents the research that has been conducted to derive common student misconceptions for elementary programming. It also presents the research approach used for knowledge elicitation, the concept inventory compiled with the findings, the techniques and tools for representing those concepts in a knowledge base along with rules that determine how they can be utilised to support learners and the reasoning mechanisms that can process those rules to materialise intelligent support in a real learning environment.

Intelligent Tutor Layered Architecture This part defines the architectural design for systems that foster intelligent tutoring for programming. It presents the different aspects of such a system and the way the respective components relate to each other. The architecture presented uses a layered approach to gradually enhance the level of support provided in an incremental manner. Its use is not expected to be exhaustive as not all components are mandatory. The intention of this design is to be used as a roadmap

for the construction of intelligent tutoring systems specialised in programming.

AuthELO This part presents both a process for designing authoring tools for the development of automated support and a real tool that instantiates these learnings. The tool is able to seamlessly integrate with any web component, dynamically configure it and enhance it with automated support. It is not dependent on specific technologies and platforms. It can operate autonomously or in conjunction with existing systems in a loosely coupled manner. It produces solutions for any domain and is particularly suited to open and exploratory learning environments. It simplifies the authoring process and makes it more accessible to low skilled designers. It is fast, efficient and simplifies both development and deployment of support in a secure and non-disruptive way.

8.3 Part 3 - Miscellaneous

In this part there are two contributions:

FLIP This part presents a PoC designed for teaching introductory programming through inquiry-based learning scenarios in an open and exploratory manner. Its main objective is to enable students work continuously and minimise interruptions in the learning process. It provides automated support on demand on common student misconceptions and adaptability based on student activity.

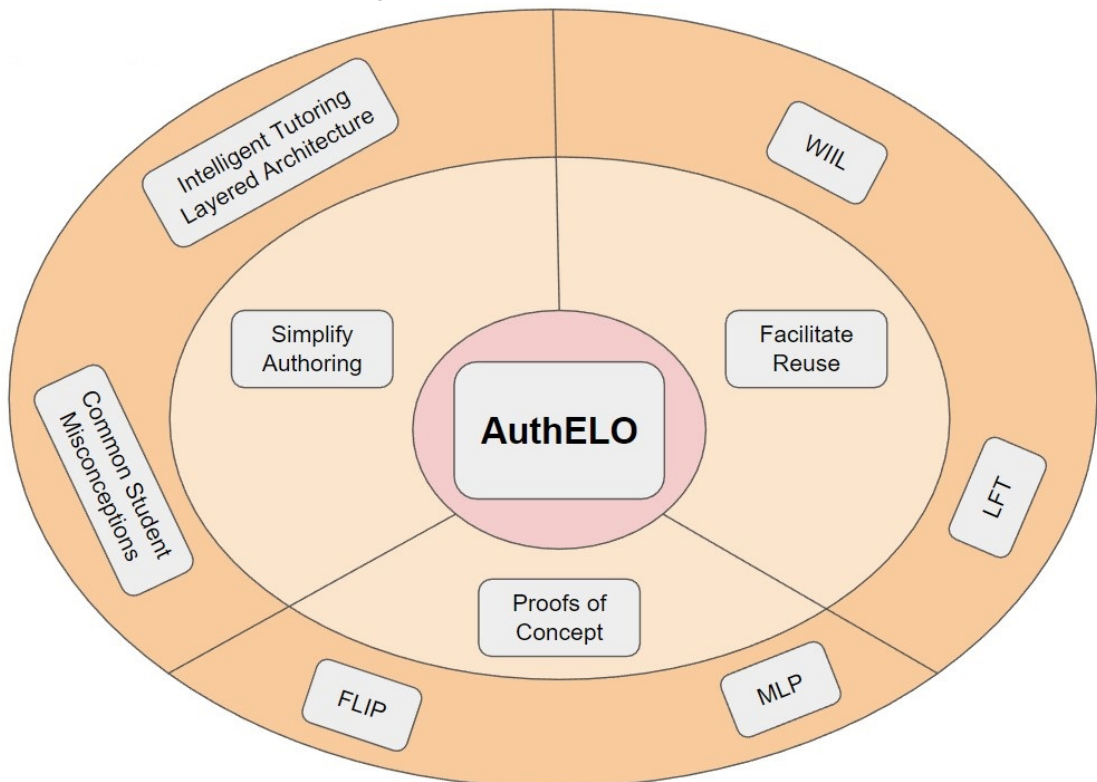
MLP This part presents a PoC that shows how all the above contributions can be combined together and real, usable systems be realised. The name of this PoC is Microworld Learning Platform.

8.4 How it all fits together

This section shows how the outcome of this work as it is described in more detail in previous sections can be seen as a coherent whole that translates the challenges in 1.2.1 into actual contributions. The following figure depicts all the contributions in relation to the categories and the challenges described in 1.2.1.

The main contribution that lies in the centre of the graph is AuthELO. AuthELO is both a process for designing authoring tools for the development of automated support and a real materialised tool that can transfer this value directly to educational practice. What makes AuthELO unique and innovative is its ability to cope with diverse components that operate in exploratory settings.

Figure 8.1: Contributions Chart



Facilitate Reuse

AuthELO is able to communicate and interoperate with learning components through a generic and lightweight integration and interoperability technique named WIIL. WIIL does not presuppose compliance with interoperability standards. Its purpose is to enable seamless integration and interoperability between diverse components with minimal overhead. This makes AuthELO reusable and able to unify in the same process and the same interface the authoring support for diverse learning components. LFT is another authoring environment that potentially increases reusability of learning components and reduces even further the difficulty of the authoring process in AuthELO. It does that through the definition of languages that can be directly usable in the browser to facilitate both goals.

Simplify Authoring

The Intelligent Tutoring Layered Architecture is a framework intended to be used as a roadmap that shows how these components can be combined together in meaningful ways in order for the authors to synthesise useful and compelling learning environments. The Common Student Misconceptions is a concept inventory developed as a byproduct of this research that may be used to simplify the development of automated support for environments that support learning programming through free exploration.

PoCs

Finally, FLIP and MLP is software that has been developed throughout the research project and can be used to demonstrate how the previously mentioned components can be translated into real and usable systems.



Future Work

In various parts of this text there are mentions of areas where there is potentially room for further investigation and improvement. The reasons to leave those parts for future work were primarily time constraints and in some cases the danger to move to an area that is outside the scope of this thesis. In this section we are giving a more detailed presentation of these cases.

9.1 Visual Integration Editor for WIIL

WIIL is a thin software layer that operates in the same context as the component it represents and functions as an enhancement that selectively exposes its public API and semantically enhances it where needed. All of this, currently, requires human intervention in terms of examining the component and its internal implementation and coding the missing parts in the WIIL library, like pushing public methods. Coding WIIL is fairly straightforward since it is done in a very controllable manner through WIIL's API but the argument here is that it could be done using visual aids from a specialised IDE. The component to be integrated could be preloaded along with WIIL and queried dynamically to provide a visual view of what functionality is available and in what form. This would take advantage of the fact that functions in JavaScript are first-class citizens and therefore they are represented in memory as values. That makes them transferable, processable and in general available for examination and manipulation. This feature would eliminate the need to manually discover the parts to be exported as public API which was identified as a serious limitation of the method by reviewers.

Ideally, the integrator should be able to utilise a visual programming environment that dynamically queries the component and provides all the necessary information to generate the code needed to complement WIIL.

9.2 Use AuthELO to handle Common Student Misconceptions

TI and TD support are treated in this text as different types of support because they are and as a result of that the way support is designed and implemented for these cases is fundamentally different. A rule-based representation technique is used to represent concepts related to common student misconceptions. Knowledge is inserted manually through an editor that allows coding in a visual language and processing is done through a specialised reasoner. The question is whether we could use authELO as a single interface through which a designer would be able to handle both types of support in order to simplify the process. If, for example, an editor enhanced with a parser and a code analyser could be integrated with authELO as a component so that we could intercept code written by the student and represent it as a vector of characteristics (variable/predicate pairs), then we should be able as designers to identify the current student state and write rules in authELO to provide support for it. AuthELO is a much easier to use interface and it would be really valuable to utilise it as a single and common interface for both cases.

9.3 Enhance AuthELO with high-level Authoring Languages

The primary design goal of authELO is to minimise the cognitive load required by the designer to develop automated support for learning objects. This goal is partially fulfilled with the current implementation that offers a high-level programming interface based on JavaScript. Although, this is a set of language constructs tailored for authoring support the fact that designers must express themselves using a language makes the technology inaccessible by certain categories of people like low skilled designers and teachers. The use of LFT may be used to derive a much simpler language, not necessarily C-like in syntax, to focus on support and further help skilled designers to express analysis and reasoning statements. On top of that, it would be really benefi-

cial if we could build a block-based visual programming environment like scratch, to enable people that possess algorithmic thinking skills but lack the ability to code in an artificial language to develop simple feedback scenarios without support. The latter was a clear user recommendation derived from the evaluation process.

9.4 Enhance AuthELO with Machine Learning Techniques

As mentioned in the previous section the primary design goal of authELO is to minimise the cognitive load required by the designer. If the system could be fed with activity indicators coming from student sessions and identify the patterns of interest in the student behaviour automatically, then the work required by the designers would be very much simplified. We believe that this is the natural next step in this process and we envisage to continue investigating towards this direction after the completion of this work.

Bibliography

- Adam, A. & Laurent, J.-P. (1980), 'Laura, a system to debug student programs', *artificial intelligence* **15**(1-2), 75–122.
- Aleven, V., McLaren, B. M., Sewall, J. & Koedinger, K. R. (2006), The cognitive tutor authoring tools (ctat): preliminary evaluation of efficiency gains, in 'International Conference on Intelligent Tutoring Systems', Springer, pp. 61–70.
- Aleven, V., McLaren, B. M., Sewall, J. & Koedinger, K. R. (2009), 'A new paradigm for intelligent tutoring systems: Example-tracing tutors', *International Journal of Artificial Intelligence in Education* **19**(2), 105–154.
- Aleven, V., McLaren, B. M., Sewall, J., Van Velsen, M., Popescu, O., Demi, S., Ringenberg, M. & Koedinger, K. R. (2016), 'Example-tracing tutors: Intelligent tutor development for non-programmers', *International Journal of Artificial Intelligence in Education* **26**(1), 224–269.
- Aleven, V., Sewall, J., McLaren, B. M. & Koedinger, K. R. (2006), Rapid authoring of intelligent tutors for real-world and experimental use, in 'Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06)', IEEE, pp. 847–851.
- Amershi, S. & Conati, C. (2006), Automatic recognition of learner groups in exploratory learning environments, in 'International Conference on Intelligent Tutoring Systems', Springer, pp. 463–472.

- Amir, O. & Gal, Y. (2013), 'Plan recognition and visualization in exploratory learning environments', *ACM Transactions on Interactive Intelligent Systems (TiiS)* 3(3), 1–23.
- Angros Jr, R., Johnson, W. L., Rickel, J. & Scholer, A. (2002), Learning domain knowledge for teaching procedural skills, in 'Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3', pp. 1372–1378.
- Barker, P. (2005), 'What is iee learning object metadata/ims learning resource metadata', *CETIS Standards Briefing Series, JISC (Joint Information Systems Committee of the Universities' Funding Councils)* .
- Barnum, C. M. (2020), *Usability testing essentials: ready, set... test!*, Morgan Kaufmann.
- Beard, M. & Barr, A. (1976), 'The basic instructional program student manual.'
- Becker, B. W. (2001), Teaching cs1 with karel the robot in java, in 'ACM SIGCSE Bulletin', Vol. 33, ACM, pp. 50–54.
- Bellaby, G., McDonald, C. & Patterson, A. (2003), Why lecture, in 'Proceedings of the 4th annual Conference of the LTSN', Citeseer, pp. 228–231.
- Bergin, J., Stehlik, M., Roberts, J. & Pattis, R. (1997a), *Karel+: A Gentle Introduction to the Art of Object-oriented Programming*, Wiley.
- Bergin, J., Stehlik, M., Roberts, J. & Pattis, R. (1997b), *Karel++: A gentle introduction to the art of object-oriented programming*, Vol. 1, Wiley New York.
- Bergin, J., Stehlik, M., Roberts, J. & Pattis, R. (2005), *Karel J Robot: A gentle introduction to the art of object-oriented programming in Java*, Dream Songs Press.
- Bernardini, A. & Conati, C. (2010), Discovering and recognizing student interaction patterns in exploratory learning environments, in 'International Conference on Intelligent Tutoring Systems', Springer, pp. 125–134.

- Blessing, S. B. (1997), 'A programming by demonstration authoring tool for model-tracing tutors'.
- Blessing, S., Gilbert, S., Ourada, S. & Ritter, S. (2007), 'Lowering the bar for creating model-tracing intelligent tutoring systems', *Frontiers in Artificial Intelligence and Applications* **158**, 443.
- Blikstein, P. (2013), Gears of our childhood: constructionist toolkits, robotics, and physical computing, past and future, in 'Proceedings of the 12th international conference on interaction design and children', pp. 173–182.
- Blikstein, P. & Krannich, D. (2013), The makers' movement and fablabs in education: experiences, technologies, and research, in 'Proceedings of the 12th international conference on interaction design and children', pp. 613–616.
- Bødker, S. & Kyng, M. (2018), 'Participatory design that matters—facing the big issues', *ACM Transactions on Computer-Human Interaction (TOCHI)* **25**(1), 1–31.
- Bohl, O., Scheuhase, J., Sengler, R. & Winand, U. (2002), The sharable content object reference model (scorm)-a critical review, in 'Computers in education, 2002. proceedings. international conference on', IEEE, pp. 950–951.
- Bonar, J. & Cunningham, R. (1988), 'Bridge: An intelligent tutor for thinking about programming', *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction. J. Self. London, Chapman and Hall* **432**.
- Bradford, P., Porciello, M., Balkon, N. & Backus, D. (2007), 'The blackboard learning system: The be all and end all in educational instruction?', *Journal of Educational Technology Systems* **35**(3), 301–314.
- Brown, J. S. & Burton, R. R. (1978), 'Diagnostic models for procedural bugs in basic mathematical skills*', *Cognitive science* **2**(2), 155–192.

- Brusilovsky, P. (1992), 'Intelligent tutor, environment and manual for introductory programming', *Educational & Training Technology International* **29**(1), 26–34.
- Brusilovsky, P., Schwarz, E. & Weber, G. (1996), Elm-art: An intelligent tutoring system on world wide web, *in* 'Intelligent tutoring systems', Springer, pp. 261–269.
- Buchanan, B. G. & Duda, R. O. (1983), Principles of rule-based expert systems, *in* 'Advances in computers', Vol. 22, Elsevier, pp. 163–216.
- Bunt, A., Conati, C., Huggett, M. & Muldner, K. (2001), On improving the effectiveness of open learning environments through tailored support for exploration, *in* 'Proceedings of AIED 2001, 10th World Conference of Artificial Intelligence and Education', Citeseer, pp. 365–376.
- Bunt, A., Conati, C. & Muldner, K. (2004), Scaffolding self-explanation to improve learning in exploratory learning environments., *in* 'International Conference on Intelligent Tutoring Systems', Springer, pp. 656–667.
- Calloni, B. A. & Bagert, D. J. (1994), Iconic programming in baccii vs. textual programming: which is a better learning environment?, *in* 'Proceedings of the twenty-fifth SIGCSE symposium on Computer science education', pp. 188–192.
- Carlier, F. & Renault, V. (2010), Educational webportals augmented by mobile devices with ifrimousse architecture, *in* '2010 10th IEEE International Conference on Advanced Learning Technologies', IEEE, pp. 236–240.
- Carroll, J. (1987), 'M., rosson, mb (1987). paradox of the active user', *Interfacing thought: cognitive aspects of human-computer interaction*, John M. Carroll (Ed.). MIT Press, Cambridge, MA, USA pp. 80–111.
- Carroll, J. M. (1982), 'The adventure of getting to know a computer', *Computer* (11), 49–58.

- Carroll, J. M. (1990), 'The nurnberg funnel: designing minimalist instruction for practical computer skill'.
- Carroll, J. M., Mack, R. L., Lewis, C. H., Grischkowsky, N. L. & Robertson, S. R. (1985), 'Exploring exploring a word processor', *Human-computer interaction* **1**(3), 283–307.
- Carroll, J. & Mazur, S. (1986), 'Learning lisa', *IEEE Computer* **91**, 35–49.
- Chakrabarti, A. (2010), 'A course for teaching design research methodology', *AI EDAM* **24**(3), 317–334.
- Chappell, D. (2011), 'Introducing odata', *Data Access for the Web, The Cloud, Mobile Devices, and More* pp. 1–24.
- Chen, M. (1995), 'A methodology for characterizing computer-based learning environments', *Instructional Science* **23**(1-3), 183–220.
- Chung, E., Huang, Y., Yajnik, S., Liang, D., Shih, J. C., Wang, C. & Wang, Y. (1997), 'Dcom and corba side by side', *Step by Step, and Layer by Layer* <http://www.cs.wustl.edu/~schmidt/submit/Paper.html> .
- Cocca, M., Gutierrez-Santos, S. & Magoulas, G. (2008a), Challenges for intelligent support in exploratory learning: the case of shapebuilder, in '1st International Workshop on Intelligent Support for Exploratory Environments', CEUR Workshop Proceedings.
- Cocca, M., Gutierrez-santos, S. & Magoulas, G. D. (2008b), S.: The challenge of intelligent support in exploratory learning environments: A study of the scenarios, in 'In: Proceedings of the 1st International Workshop in Intelligent Support for Exploratory Environments on European Conference on Technology Enhanced Learning', Cite-seer.

- Cocea, M. & Magoulas, G. (2009a), Context-dependent personalised feedback prioritisation in exploratory learning for mathematical generalisation, *in* 'International Conference on User Modeling, Adaptation, and Personalization', Springer, pp. 271–282.
- Cocea, M. & Magoulas, G. D. (2009b), 'Hybrid model for learner modelling and feedback prioritisation in exploratory learning', *International Journal of Hybrid Intelligent Systems* 6(4), 211–230.
- Cocea, M. & Magoulas, G. D. (2010), Group formation for collaboration in exploratory learning using group technology techniques, *in* 'International Conference on Knowledge-Based and Intelligent Information and Engineering Systems', Springer, pp. 103–113.
- Cooper, S., Dann, W. & Pausch, R. (2000), 'Alice: a 3-d tool for introductory programming concepts', *Journal of computing sciences in colleges* 15(5), 107–116.
- Corbett, A. T. & Anderson, J. R. (1993), Student modeling in an intelligent programming tutor, *in* 'Cognitive models and intelligent environments for learning programming', Springer, pp. 135–144.
- Cronin, P., Ryan, F. & Coughlan, M. (2008), 'Undertaking a literature review: a step-by-step approach', *British journal of nursing* 17(1), 38–43.
- Dann, W., Cooper, S. & Pausch, R. (2000), Making the connection: programming with animated small world, *in* 'ACM SIGCSE Bulletin', Vol. 32, ACM, pp. 41–44.
- Deek, F. P. & McHugh, J. A. (1998), 'A survey and critical analysis of tools for learning programming', *Computer Science Education* 8(2), 130–178.
- Dorst, K. (2011), 'The core of 'design thinking' and its application', *Design studies* 32(6), 521–532.
- Duda, R. O. & Shortliffe, E. H. (1983), 'Expert systems research', *Science* 220(4594), 261–268.

- Fällman, D. (2004), Design oriented-research versus research-oriented design, in 'Workshop Paper, CHI 2004 Workshop on Design and HCI, Conference on Human Factors in Computing Systems, CHI', Citeseer, pp. 24–29.
- Ferré, X. & Vegas, S. (1999), An evaluation of domain analysis methods, in 'Proceedings 4th CAiSE Workshop on Exploring Modelling Methods for Systems Analysis and Design', Citeseer, pp. 1–13.
- Forcheri, P. & Molfino, M. T. (1994), 'Software tools for the learning of programming: A proposal', *Computers & Education* **23**(4), 269–276.
- Gaffney, C., Dagger, D. & Wade, V. (2010), 'Authoring and delivering personalised simulations-an innovative approach to adaptive elearning for soft skills.', *J. UCS* **16**(19), 2780–2800.
- Gal, Y., Yamangil, E., Shieber, S. M., Rubin, A. & Grosz, B. J. (2008), Towards collaborative intelligent tutors: Automated recognition of users' strategies, in 'International Conference on Intelligent Tutoring Systems', Springer, pp. 162–172.
- Gentner, D., Landers, R. et al. (1985), Analogical reminding: A good match is hard to find, in 'Proceedings of the international conference on systems, man and cybernetics', pp. 607–613.
- Ginon, B., Jean-Daubias, S., Lefevre, M., Champin, P.-A. et al. (2014), Adding epiphytic assistance systems in learning applications using the sepia system, in 'European Conference on Technology Enhanced Learning', Springer, pp. 138–151.
- Glinert, E. P. & Tanimoto, S. L. (1984), 'Pict: An interactive graphical programming environment', *Computer* (11), 7–25.
- Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C. & Zilles, C. (2008), Identifying important and difficult concepts in introductory computing

- courses using a delphi process, *in* 'Proceedings of the 39th SIGCSE technical symposium on Computer science education', pp. 256–260.
- González, M. A. C., Penalvo, F. J. G., Guerrero, M. J. C. & Forment, M. A. (2009), Adapting lms architecture to the soa: an architectural approach, *in* 'Internet and Web Applications and Services, 2009. ICIW'09. Fourth International Conference on', IEEE, pp. 322–327.
- Gravemeijer, K. & Cobb, P. (2006), 'Educational design research', *J. Van den Akker, K. Gravemeijer, S. McKenney, & N. Nieveen (Eds.)* pp. 17–51.
- Grudin, J. (2009), 'Ai and hci: Two fields divided by a common focus', *Ai Magazine* 30(4), 48–48.
- Guest, G., MacQueen, K. M. & Namey, E. E. (2011), *Applied thematic analysis*, sage publications.
- Guimarães, M. A. M., de Lucena, C. J. P. & Cavalcanti, M. R. (1994), 'Experience using the asa algorithm teaching system', *ACM SIGCSE Bulletin* 26(4), 45–50.
- Gurram, R., Mo, B. & Gueldemeister, R. (2008), A web based mashup platform for enterprise 2.0, *in* 'Web Information Systems Engineering–WISE 2008 Workshops', Springer, pp. 144–151.
- Gutierrez-Santos, S., Cocea, M. & Magoulas, G. (2010), A case-based reasoning approach to provide adaptive feedback in microworlds, *in* 'International Conference on Intelligent Tutoring Systems', Springer, pp. 330–333.
- Gutierrez-Santos, S., Geraniou, E., Pearce-Lazard, D. & Poulouvasilis, A. (2012), 'Design of teacher assistance tools in an exploratory learning environment for algebraic generalization', *IEEE Transactions on Learning Technologies* 5(4), 366–376.

- Gutierrez-Santos, S., Mavrikis, M. & Magoulas, G. (2010), Layered development and evaluation for intelligent support in exploratory environments: the case of microworlds, in 'International Conference on Intelligent Tutoring Systems', Springer, pp. 105–114.
- Gutierrez-Santos, S., Mavrikis, M., Magoulas, G. D. et al. (2012), 'A separation of concerns for engineering intelligent support for exploratory learning environments', *Journal of Research and Practice in Information Technology* **44**(3), 347.
- Hannafin, M., Land, S. & Oliver, K. (1999), 'Open learning environments: Foundations, methods, and models', *Instructional-design theories and models: A new paradigm of instructional theory* **2**, 115–140.
- Hansen, A., Mavrikis, M., Holmes, W. & Geraniou, E. (2015), Designing interactive representations for learning fraction equivalence, in 'Proceedings of the 12th International Conference on Technology in Mathematics Teaching, Faro, Portugal'.
- Harris, J. (2002), 'An introduction to authoring tools', *ASTD's Learning Circuits online magazine*.
- Hartig, O. & Pérez, J. (2017), 'An initial analysis of facebook's graphql language'.
- Henriksen, P. & Kölling, M. (2004), Greenfoot: combining object visualisation with interaction, in 'Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications', pp. 73–82.
- Henry, R. R., Whaley, K. M. & Forstall, B. (1990), 'The university of washington illustrating compiler', *ACM SIGPLAN Notices* **25**(6), 223–233.
- Henson, K. L. & Knezek, G. A. (1991), 'The use of prototyping for educational software development', *Journal of Research on Computing in Education* **24**(2), 230–239.

- Hohmann, L., Guzdial, M. & Soloway, E. (1992), Soda: A computer-aided design environment for the doing and learning of software design, *in* 'International Conference on Computer Assisted Learning', Springer, pp. 307–319.
- Holland, J., Mitrovic, A. & Martin, B. (2009), 'J-latte: a constraint-based tutor for java'.
- Huitt, W. (2003), 'Constructivism', *Educational psychology interactive* .
- Isoda, S., Shimomura, T. & Ono, Y. (1987), 'Vips: A visual debugger', *IEEE Software* (3), 8–19.
- Jackson, C. & Wang, H. J. (2007), Subspace: secure cross-domain communication for web mashups, *in* 'Proceedings of the 16th international conference on World Wide Web', pp. 611–620.
- Jalali, S. & Wohlin, C. (2012), Systematic literature studies: database searches vs. backward snowballing, *in* 'Proceedings of the 2012 ACM-IEEE international symposium on empirical software engineering and measurement', IEEE, pp. 29–38.
- Järvinen, H. (2011), Html5 web workers, *in* 'T-111.5502 Seminar on Media Technology BP, Final Report', p. 27.
- Jarvis, M. P., Nuzzo-Jones, G. & Heffernan, N. T. (2004), Applying machine learning techniques to rule generation in intelligent tutoring systems, *in* 'International Conference on Intelligent Tutoring Systems', Springer, pp. 541–553.
- Jenkins, C. W. (2012), 'Microworlds: Building powerful ideas in the secondary school.', *Online Submission* .
- Jenkins, T. (2002), On the difficulty of learning to program, *in* 'Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences', Vol. 4, pp. 53–58.

- Johnson, W. L. & Soloway, E. (1985), 'Proust: Knowledge-based program understanding', *Software Engineering, IEEE Transactions on* (3), 267–275.
- Kaczmarczyk, L. C., Petrick, E. R., East, J. P. & Herman, G. L. (2010), Identifying student misconceptions of programming, in 'Proceedings of the 41st ACM technical symposium on Computer science education', pp. 107–111.
- Kafai, Y. B. & Burke, Q. (2013), 'Computer programming goes back to school', *Phi Delta Kappan* 95(1), 61–65.
- Kahn, K. (1996), 'Toontalktm—an animated programming environment for children', *Journal of Visual Languages & Computing* 7(2), 197–217.
- Kaplan, R. M. (2010), Choosing a first programming language, in 'Proceedings of the 2010 ACM conference on Information technology education', pp. 163–164.
- Karkalas, S., Bokhove, C., Charlton, P. & Mavrikis, M. (2015), Towards configurable learning analytics for constructionist mathematical e-books., in 'AIED Workshops'.
- Karkalas, S. & Gutierrez-Santos, S. (2014a), Eclipse student (in) activity detection tool, in 'European Conference on Technology Enhanced Learning', Springer, pp. 572–573.
- Karkalas, S. & Gutiérrez-Santos, S. (2014b), Enhance teaching and learning of computer programming in exploratory learning environments using intelligent support, in '2014 IEEE 14th International Conference on Advanced Learning Technologies', IEEE, pp. 765–767.
- Karkalas, S. & Gutierrez-Santos, S. (2014c), Enhanced javascript learning using code quality tools and a rule-based system in the flip exploratory learning environment, in '2014 IEEE 14th International Conference on Advanced Learning Technologies', IEEE, pp. 84–88.

- Karkalas, S. & Gutierrez-Santos, S. (2015), Intelligent and adaptive student support in flip-early computer programming, *in* 'Doctoral Consortium on Computer Supported Education', Vol. 2, SCITEPRESS, pp. 23–27.
- Karkalas, S. & Mavrikis, M. (2016), Feedback authoring for exploratory learning objects: Authelo, *in* 'CSEDU 2016-Proceedings of the 8th International Conference on Computer Supported Education', Vol. 1, Science and Technology Publications, Lda, pp. 144–153.
- Karkalas, S., Mavrikis, M. & Charlton, P. (2015), The web integration & interoperability layer (wiil)-turning web content into learning content using a lightweight integration and interoperability technique, *in* 'International Conference on Knowledge Engineering and Ontology Development', Vol. 2, SCITEPRESS, pp. 139–146.
- Karkalas, S., Mavrikis, M., Xenos, M. & Kynigos, C. (2016), Feedback authoring for exploratory activities: The case of a logo-based 3d microworld, *in* 'International Conference on Computer Supported Education', Springer, pp. 259–278.
- Karkalas, S. & Santos, S. G. (2014), Intelligent student support in the flip learning system based on student initial misconceptions and student modelling., *in* 'KEOD', pp. 353–360.
- Karoui, A., Marfisi-Schottman, I. & George, S. (2016), Mobile learning game authoring tools: assessment, synthesis and proposals, *in* 'International Conference on Games and Learning Alliance', Springer, pp. 281–291.
- Kirschner, P. A., Sweller, J. & Clark, R. E. (2006), 'Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching', *Educational psychologist* 41(2), 75–86.
- Klahr, D. & Nigam, M. (2004), 'The equivalence of learning paths in early science in-

- struction: Effects of direct instruction and discovery learning', *Psychological science* **15**(10), 661–667.
- Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B. & Hockenberry, M. (2004), Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration, in 'International Conference on Intelligent Tutoring Systems', Springer, pp. 162–174.
- Köhne, A. & Weber, G. (1987), Struedi: A lisp-structure editor for novice programmers, in 'Human–Computer Interaction–INTERACT'87', Elsevier, pp. 125–129.
- Kolb, D. A. et al. (1984), *Experiential learning: Experience as the source of learning and development*, Vol. 1, Prentice-Hall Englewood Cliffs, NJ.
- Kölling, M. (2010), 'The greenfoot programming environment', *ACM Transactions on Computing Education (TOCE)* **10**(4), 14.
- Kölling, M. & Henriksen, P. (2005), Game programming in introductory courses with direct state manipulation, in 'Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education', pp. 59–63.
- Kölling, M., Quig, B., Patterson, A. & Rosenberg, J. (2003), 'The bluej system and its pedagogy', *Computer Science Education* **13**(4), 249–268.
- Kölling, M. & Rosenberg, J. (1996), 'An object-oriented program development environment for the first programming course', *ACM SIGCSE Bulletin* **28**(1), 83–87.
- Konak, A., Clark, T. K. & Nasereddin, M. (2014), 'Using kolb's experiential learning cycle to improve student learning in virtual computer laboratories', *Computers & Education* **72**, 11–22.
- URL:** <http://linkinghub.elsevier.com/retrieve/pii/S0360131513002984>
- Kraemer, E. & Stasko, J. T. (1998), 'Creating an accurate portrayal of concurrent executions', *IEEE concurrency* **6**(1), 36–46.

- Kruglyk, V. & Lvov, M. (2012), Choosing the first educational programming language, *in* 'ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer: Proceedings of the 8th International Conference ICTERI 2012', Kherson, pp. 188–189.
- Kynigos, C. (1992), Insights into pupils' and teachers' activities in pupil-controlled problem-solving situations: A longitudinally developing use for programming by all in a primary school, *in* 'Mathematical Problem Solving and New Information Technologies', Springer, pp. 219–238.
- Kynigos, C. (2015), Constructionism: Theory of learning or theory of design?, *in* 'Selected regular lectures from the 12th International Congress on Mathematical Education', Springer, pp. 417–438.
- Kynigos, C. & Latsi, M. (2007), 'Turtle's navigation and manipulation of geometrical figures constructed by variable processes in a 3d simulated space.', *Informatics in education* 6(2), 359–372.
- Lahtinen, E. & Ahoniemi, T. (2005), Visualizations to support programming on different levels of cognitive development, *in* 'Proceedings of The Fifth Koli Calling Conference on Computer Science Education', pp. 87–94.
- Laubsch, J. H. & Eisenstadt, M. (1981), Domain specific debugging aids for novice programmers., *in* 'IJCAI', pp. 964–969.
- Leo, D. H., Pérez, J. I. A. & Dimitriadis, Y. A. (2004), Ims learning design support for the formalization of collaborative learning patterns, *in* 'IEEE International Conference on Advanced Learning Technologies, 2004. Proceedings.', IEEE, pp. 350–354.
- Levy, S. P. (1995), 'Computer language usage in cs1: Survey results', *ACM SIGCSE Bulletin* 27(3), 21–26.

- Lor, R. (2017), Design thinking in education: a critical review of literature, *in* 'IACSSM; ACEP, Conference Proceedings', pp. 24–26.
- Lynch, C. F., Ashley, K. D., Alevan, V. & Pinkwart, N. (2006), Defining ill-defined domains; a literature survey, *in* 'Intelligent Tutoring Systems (ITS 2006): Workshop on Intelligent Tutoring Systems for Ill-Defined Domains'.
- Malone, T. W. (1980), What makes things fun to learn? heuristics for designing instructional computer games, *in* 'Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems', pp. 162–169.
- Malone, T. W. (1982), Heuristics for designing enjoyable user interfaces: Lessons from computer games, *in* 'Proceedings of the 1982 conference on Human factors in computing systems', pp. 63–68.
- Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M. & Rusk, N. (2008), 'Programming by choice: urban youth learning programming with scratch', *ACM SIGCSE Bulletin* **40**(1), 367–371.
- Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G. & Koedinger, K. R. (2007), 'Predicting students' performance with simstudent: Learning cognitive skills from observation', *Frontiers in Artificial Intelligence and Applications* **158**, 467.
- Mavrikis, M., Gutierrez-Santos, S., Geraniou, E. & Noss, R. (2013), 'Design requirements, student perception indicators and validation metrics for intelligent exploratory learning environments', *Personal and ubiquitous computing* **17**(8), 1605–1620.
- Mavrikis, M., Karkalas, S., Cukurova, M. & Papapesiou, E. (2019), Participatory design to lower the threshold for intelligent support authoring, *in* 'International Conference on Artificial Intelligence in Education', Springer, pp. 185–189.
- Mayer, R. E. (2004), 'Should there be a three-strikes rule against pure discovery learning?', *American Psychologist* **59**(1), 14.

- McCalla, G. & Murtagh, K. (1985), *GENIUS: An experiment in ignorance-based automated program advising*, University of Saskatchewan, Department of Computational Science.
- McKenney, S. & Reeves, T. C. (2014), Educational design research, in 'Handbook of research on educational communications and technology', Springer, pp. 131–140.
- McKenney, S. & Reeves, T. C. (2018), *Conducting educational design research*, Routledge.
- Melles, G., Anderson, N., Barrett, T. & Thompson-Whiteside, S. (2015), Problem finding through design thinking in education, in 'Inquiry-based learning for multidisciplinary programs: A conceptual and practical resource for educators', Emerald Group Publishing Limited.
- Millard, N., Lynch, P. & Tracey, K. (1998), Child's play: using techniques developed to elicit requirements from children with adults, in 'Proceedings of IEEE International Symposium on Requirements Engineering: RE'98', IEEE, pp. 66–73.
- Mills, D. & Morton, M. (2013), *Ethnography in education*, Sage.
- Mitrovic, A. (1997), 'Sql-tutor: a preliminary report'.
- Mitrovic, A. (2003), 'An intelligent sql tutor on the web', *International Journal of Artificial Intelligence in Education* **13**(2), 173–197.
- Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J. & McGuigan, N. (2009), 'Aspire: an authoring system and deployment environment for constraint-based tutors', *International Journal of Artificial Intelligence in Education* **19**(2), 155–188.
- Montori, V. M., Wilczynski, N. L., Morgan, D. & Haynes, R. B. (2003), 'Systematic reviews: a cross-sectional study of location and citation counts', *BMC medicine* **1**(1), 1–7.

- Morales Gamboa, R. & Gamboa, R. M. (2000), 'Exploring participative learner modelling and its effects on learner behaviour'.
- Morgado, L. & Kahn, K. (2008), 'Towards a specification of the toontalk language', *Journal of Visual Languages & Computing* **19**(5), 574–597.
- Mott, J. (2010), 'Envisioning the post-lms era: The open learning network', *Educause Quarterly* **33**(1), 1–9.
- Mukherjea, S. & Stasko, J. T. (1994), 'Toward visual debugging: integrating algorithm animation capabilities within a source-level debugger', *ACM Transactions on Computer-Human Interaction (TOCHI)* **1**(3), 215–244.
- Murray, T. (1997), 'Expanding the knowledge acquisition bottleneck for intelligent tutoring systems', *International Journal of Artificial Intelligence in Education* **8**(3-4), 222–232.
- Murray, T. (1999), 'Authoring intelligent tutoring systems: An analysis of the state of the art'.
- Murray, T. (2016), 'Coordinating the complexity of tools, tasks, and users: On theory-based approaches to authoring tool usability', *International Journal of Artificial Intelligence in Education* **26**(1), 37–71.
- Myers, B. A., Chandhok, R. & Sareen, A. (1988), Automatic data visualization for novice pascal programmers., in 'VL', pp. 192–198.
- Neal, L. R. (1986), Cognition-sensitive design and user modeling for syntax-directed editors, in 'Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface', pp. 99–102.
- Neal, L. R. (1989), 'A system for example-based programming', *ACM SIGCHI Bulletin* **20**(SI), 63–68.

- Newell, A. (1962), Some problems of basic organization in problem-solving programs, Technical report, Rand corp santa monica ca.
- Nieveen, N. & Folmer, E. (2013), 'Formative evaluation in educational design research', *Design Research* **153**, 152–169.
- Noblit, G. W. (2003), Reinscribing critique in educational ethnography: Critical and postcritical ethnography, in 'Foundations for Research', Routledge, pp. 197–218.
- Noss, R. & Hoyles, C. (1996), *Windows on mathematical meanings: Learning cultures and computers*, Vol. 17, Springer Science & Business Media.
- O'leary, Z. (2017), *The essential guide to doing your research project*, Sage.
- Olsen, K. A., Harnes, P., Pedersen, B. & Tosse, O.-J. (1988), The dsp system-a visual system to support teaching of programming, in '[Proceedings] 1988 IEEE Workshop on Visual Languages', IEEE, pp. 199–206.
- Papert, S. (1980), 'Mindstons', *New York: Basic Books* **607**.
- Papert, S. (1993), 'The children's machine', *TECHNOLOGY REVIEW-MANCHESTER NH- 96*, 28–28.
- Paquette, G., Pachet, F., Giroux, S. & Girard, J. (1996), 'Epitalk: Generating advisory agents for existing information systems', *Journal of Artificial Intelligence in Education* **7**, 349–379.
- Paquette, G. & Tchounikine, P. (1999), Towards a knowledge engineering method for the construction of advisor systems, in 'AI-Ed'99', number 50, IOS Press, pp. 753–755.
- Patsopoulos, N. A., Analatos, A. A. & Ioannidis, J. P. (2005), 'Relative citation impact of various study designs in the health sciences', *Jama* **293**(19), 2362–2366.

- Pattis, R. E. (1981), *Karel the robot: a gentle introduction to the art of programming*, John Wiley & Sons, Inc.
- Pawar, U. S., Pal, J. & Toyama, K. (2006), Multiple mice for computers in education in developing countries, in '2006 International Conference on Information and Communication Technologies and Development', IEEE, pp. 64–71.
- Pearce, D., Mavrikis, M., Geraniou, E., Gutiérrez, S. & Kahn, K. (2008), An environment for expressing mathematical generalisation using pattern construction and analysis, in 'Workshop for HCI for Technology Enhanced Learning, at HCI2008 Culture, Creativity, Interaction'.
- Pearce, D. & Poulouvasilis, A. (2009), The conceptual and architectural design of a system supporting exploratory learning of mathematics generalisation, in 'European Conference on Technology Enhanced Learning', Springer, pp. 22–36.
- Pearce-Lazard, D., Poulouvasilis, A. & Geraniou, E. (2010), The design of teacher assistance tools in an exploratory learning environment for mathematics generalisation, in 'European Conference on Technology Enhanced Learning', Springer, pp. 260–275.
- Peylo, C., Teiken, W., Rollinger, C.-R. & Gust, H. (2000), An ontology as domain model in a web-based educational system for prolog., in 'FLAIRS Conference', pp. 55–59.
- Plomp, T. (2013), 'Educational design research: An introduction', *Educational design research* pp. 11–50.
- Pole, C. & Morrison, M. (2003), *Ethnography for education*, McGraw-Hill Education (UK).
- Prieto-Diaz, R. (1990), 'Domain analysis: An introduction', *ACM SIGSOFT Software Engineering Notes* **15**(2), 47–54.
- Privitera, G. & Delzell, L. (2019), 'Quasy-experimental and single-case experimental designs', *Research methods for education* pp. 333–370.

- Provenzo Jr, E. F. (1991), *Video kids: Making sense of Nintendo.*, Harvard University Press.
- Ramadhan, H. (1992), An intelligent discovery programming system, in 'Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing: technological challenges of the 1990's', pp. 149–159.
- Ramadhan, H. & du Boulay, B. (1993), Programming environments for novices, in 'Cognitive models and intelligent environments for learning programming', Springer, pp. 125–134.
- Reges, S. (2002), 'Can c# replace java in cs1 and cs2?', *ACM SIGCSE Bulletin* 34(3), 4–8.
- Reiser, B. J., Anderson, J. R. & Farrell, R. G. (1985), Dynamic student modelling in an intelligent tutor for lisp programming., in 'IJCAI', pp. 8–14.
- Reiser, B., Ranney, M., Lovett, M. C. & Kimberg, D. Y. (1989), Facilitating students' reasoning with causal explanations and visual representations, Technical report, PRINCETON UNIV NJ COGNITIVE SCIENCE LAB.
- Reiss, S. P. (1985), 'Pecan: Program development systems that support multiple views', *IEEE Transactions on Software engineering* (3), 276–285.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. et al. (2009), 'Scratch: programming for all', *Communications of the ACM* 52(11), 60–67.
- Richard, B. & Tchounikine, P. (2004), 'Enhancing the adaptivity of an existing web-site with an epiphyte recommender system', *New review of hypermedia and multimedia* 10(1), 31–52.
- Richard, B., Tchounikine, P. & Jacoboni, P. (2003), An architecture to support navigation and propose tips within a dedicated web site, in 'Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)', IEEE, pp. 278–284.

-
- Rieman, J. (1996), 'A field study of exploratory learning strategies', *ACM Transactions on Computer-Human Interaction (TOCHI)* **3**(3), 189–218.
- Robillard, P. N. (1986), 'Schematic pseudocode for program constructs and its computer automation by schemacode', *Communications of the ACM* **29**(11), 1072–1089.
- Robins, A., Rountree, J. & Rountree, N. (2003), 'Learning and teaching programming: A review and discussion', *Computer Science Education* **13**(2), 137–172.
- Ruipérez-Valiente, J. A., Muñoz-Merino, P. J., Leony, D. & Kloos, C. D. (2015), 'Alas-ka: A learning analytics extension for better understanding the learning process in the khan academy platform', *Computers in Human Behavior* **47**, 139–148.
- Saaty, T. L. (1990), 'How to make a decision: the analytic hierarchy process', *European journal of operational research* **48**(1), 9–26.
- Savery, J. R. (2006), 'Overview of problem-based learning: Definitions and distinctions', *Interdisciplinary Journal of Problem-based Learning* **1**(1), 3.
- Schlosser, R. W., Wendt, O., Bhavnani, S. & Nail-Chiwetalu, B. (2006), 'Use of information-seeking strategies for developing systematic reviews and engaging in evidence-based practice: the application of traditional and comprehensive pearl growing. a review', *International Journal of Language & Communication Disorders* **41**(5), 567–582.
- Schmalhofer, F., Kühn, O., Charron, R. & Messamer, P. (1990), 'An implementation and empirical evaluation of an exploration environment with different tutoring strategies', *Behavior Research Methods, Instruments, & Computers* **22**(2), 179–183.
- Schmalhofer, F., Kühn, O., Messamer, P. & Charron, R. (1990), 'An experimental evaluation of different amounts of receptive and exploratory learning in a tutoring system', *Computers in human behavior* **6**(1), 51–68.

- Seffah, A., Donyaee, M., Kline, R. B. & Padda, H. K. (2006), 'Usability measurement and metrics: A consolidated model', *Software quality journal* **14**(2), 159–178.
- Severance, C., Hanss, T. & Hardin, J. (2010), 'Ims learning tools interoperability: Enabling a mash-up approach to teaching and learning tools', *Technology, Instruction, Cognition and Learning* **7**(3-4), 245–262.
- Severance, C., Hardin, J. & Whyte, A. (2008), 'The coming functionality mash-up in personal learning environments', *Interactive Learning Environments* **16**(1), 47–62.
- Shimomura, T. & Isoda, S. (1991), 'Linked-list visualization for debugging', *IEEE Software* **8**(3), 44–51.
- Shneiderman, B. (1993), '1.1 direct manipulation: a step beyond programming languages', *Sparks of innovation in human-computer interaction* **17**, 1993.
- Solomon, C. J. & Papert, S. (1976), A case study of a young child doing turtle graphics in logo, in 'Proceedings of the June 7-10, 1976, national computer conference and exposition', pp. 1049–1056.
- Soloway, E. (1986), 'Learning to program= learning to construct mechanisms and explanations', *Communications of the ACM* **29**(9), 850–858.
- Sottolare, R. A., Brawner, K. W., Goldberg, B. S. & Holden, H. K. (2012), 'The generalized intelligent framework for tutoring (gift)', *Orlando, FL: US Army Research Laboratory–Human Research & Engineering Directorate (ARL-HRED)* .
- Strauss, A. & Corbin, J. M. (1997), *Grounded theory in practice*, Sage.
- Sykes, E. R. & Franek, F. (2003), A prototype for an intelligent tutoring system for students learning to program in java (tm), in 'Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education, June 30-July 2, 2003, Rhodes, Greece', pp. 78–83.

- Sylvester, A., Tate, M. & Johnstone, D. (2013), 'Beyond synthesis: Re-presenting heterogeneous research literature', *Behaviour & Information Technology* 32(12), 1199–1215.
- Tecuci, G. & Keeling, H. (1998), Developing intelligent educational agents with the disciple learning agent shell, in 'International Conference on Intelligent Tutoring Systems', Springer, pp. 454–463.
- Tecuci, G. & Keeling, H. (1999), 'Developing an intelligent educational agent with disciple', *International Journal of Artificial Intelligence in Education* 10(3-4), 221–237.
- Tecuci, G., Wright, K., Lee, S., Boicu, M., Bowman, M. & Webster, D. (1998), A learning agent shell and methodology for developing intelligent agents, in 'AAAI-98 Workshop: Software Tools for Developing Agents', pp. 37–46.
- Thomas, J. M. & Young, R. M. (2011), Dynamic guidance for task-based exploratory learning, in 'International Conference on Artificial Intelligence in Education', Springer, pp. 369–376.
- Ueno, H. (1994), 'Integrated intelligent programming environment for learning programming', *IEICE Transactions on information and systems* 77(1), 68–79.
- Ulrich, K. & Eppinger, S. (2011), *EBOOK: Product Design and Development*, McGraw Hill.
- Urban, G. L. & Von Hippel, E. (1988), 'Lead user analyses for the development of new industrial products', *Management science* 34(5), 569–582.
- Van Haaster, K. & Hagan, D. (2004), 'Teaching and learning with bluej: an evaluation of a pedagogical tool.', *Issues in Informing Science & Information Technology* 1.
- van Joolingen, W. R. & Zacharia, Z. C. (2009), Developments in inquiry learning, in 'Technology-enhanced learning', Springer, pp. 21–37.
- Van Roy, P., Armstrong, J., Flatt, M. & Magnusson, B. (2003), 'The role of language paradigms in teaching programming', *ACM SIGCSE Bulletin* 35(1), 269–270.

- Van Roy, P. & Haridi, S. (2003), Teaching programming broadly and deeply: the kernel language approach, *in* 'Informatics Curricula and Teaching Methods', Springer, pp. 53–62.
- Verdejo, M. F., Fernández, I. & Urretavizcaya, M. T. (1993), Methodology and design issues in capra, an environment for learning program construction, *in* 'Cognitive Models and Intelligent Environments for Learning Programming', Springer, pp. 156–171.
- Verschuren, P. & Hartog, R. (2005), 'Evaluation in design-oriented research', *Quality and Quantity* **39**(6), 733–762.
- Vihavainen, A., Paksula, M. & Luukkainen, M. (2011), Extreme apprenticeship method in teaching programming for beginners, *in* 'Proceedings of the 42nd ACM technical symposium on Computer science education', ACM, pp. 93–98.
- Vines, J., Clarke, R., Wright, P., McCarthy, J. & Olivier, P. (2013), Configuring participation: on how we involve people in design, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', pp. 429–438.
- Vinoski, S. (1997), 'Corba: integrating diverse applications within distributed heterogeneous environments', *IEEE Communications magazine* **35**(2), 46–55.
- Virtanen, A. T., Lahtinen, E. & Järvinen, H.-M. (2005), Vip, a visual interpreter for learning introductory programming with c++, *in* 'Proceedings of The Fifth Koli Calling Conference on Computer Science Education', pp. 125–130.
- Vygotskiï, L. S., Cole, M. & John-Steiner, V. (1978), 'Mind in society'.
- Weber, G. & Mollenberg, A. (1994), 'Elm-pe: A knowledge-based programming environment for learning lisp.'
- Weibel, S., Kunze, J., Lagoze, C. & Wolf, M. (1998), 'Dublin core metadata for resource discovery', *Internet Engineering Task Force RFC* **2413**(222), 132.

- Wilson, S. & Currier, S. (2002), 'What is ims content packaging', *CETIS Standards Briefings Series, JISC* .
- Wilson, S., Sharples, P. & Griffiths, D. (2007), 'Extending ims learning design services using widgets: Initial findings and proposed architecture', *Current Research on IMS Learning Design and Lifelong Competence Development Infrastructures* p. 3.
- Wilson, S., Sharples, P. & Griffiths, D. (2008), Distributing education services to personal and institutional systems using widgets, in 'Proc. Mash-Up Personal Learning Environments-1st Workshop MUPPLE', Vol. 8, pp. 25–33.
- Wong, S. C. (1993), 'Quick prototyping of educational software: an object-oriented approach', *Journal of educational technology systems* **22**(2), 155–172.
- Woolf, B. & Cunningham, P. A. (1987), 'Multiple knowledge sources in intelligent teaching systems', *IEEE Expert* **2**(2), 41–54.
- Zelkowitz, M. V., Kowalchack, B., Itkin, D. & Herman, L. (1989), A support tool for teaching computer programming, in 'Issues in software engineering education', Springer, pp. 139–167.
- Zhang, H., Almeroth, K., Knight, A., Bulger, M. & Mayer, R. (2007), Moodog: Tracking students' online learning activities, in 'EdMedia+ Innovate Learning', Association for the Advancement of Computing in Education (AACE), pp. 4415–4422.
- Zhang, L., Zou, X. & Kan, Z. (2014), 'Improved strategy for resource allocation in repetitive projects considering the learning effect', *Journal of Construction Engineering and Management* **140**(11), 04014053.
- Zopounidis, C. & Doumpos, M. (2002), 'Multicriteria classification and sorting methods: A literature review', *European Journal of Operational Research* **138**(2), 229–246.

-
- Zschaler, S., Demuth, B. & Schmitz, L. (2014), 'Salespoint: A java framework for teaching object-oriented software development', *Science of Computer Programming* **79**, 189–203.



Sample Rules

A.1 Understanding the Role of Variable Declaration

This is a very simple rule that illustrates the misconception related to the meaning of variable declaration. This is a common problem with novice developers that redeclare an existing variable instead of simply referencing it. Some languages are too lenient with this as there is no mechanism to detect the problem and subsequent declarations are silently interpreted as simple references to the variable. If there is no compiler to detect the error early enough this may end up being a recurring problem that hinders

the development of other concepts.

A code example that illustrates the problem:

```
var x = new Array(5);  
var x = 5;  
alert(x);
```

The description of the rule follows:

- variable v1 is a var (var declaration in the code)
- v1 is not distinct (there are multiple declarations of the same identifier)
- if v1 is false stop the process
- if v1 is true activate the rule

The subsequent of this rule could be a suggestion to remove the word from all references to the variable that follow the initial declaration.

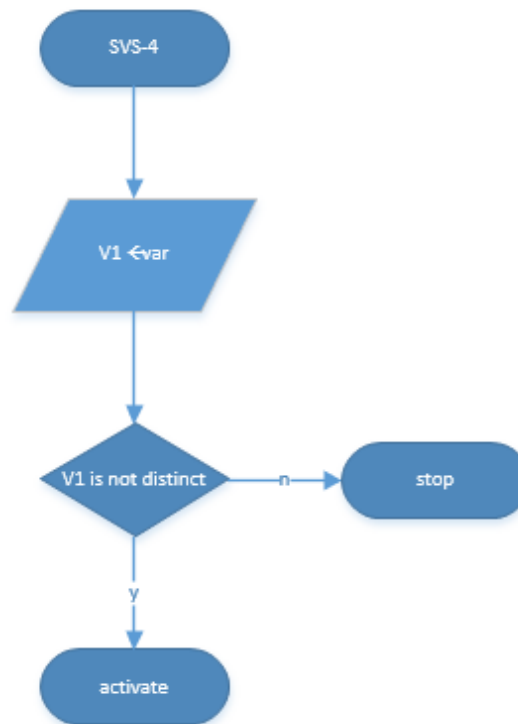
A.2 Understanding the Difference Between Variable Values and Literal Values

This is another common problem that reveals itself quite frequently. Students declare a variable and assign a value to it but they don't use it in subsequent operations. Instead they refer to the literal value directly.

A code example that illustrates the problem:

```
var x = 5;  
var y = 3;
```

Figure A.1: Understanding the Role of Variable Declaration



```
var z = 5 - 3;
```

```
if (5>3)
```

```
{
```

```
    z = 14;
```

```
}
```

In the code snippet above there are two instances of that problem (lines 3 and 5).

The description of the rule follows:

- variable v1 is a var (var declaration in the code)
- v1 is not null (if there is no v1 we stop the process)
- variable v2 is a literal (a literal value in the code)

- v2 is not null (if there is no v2 we stop the process)
- variable v3 is the value of variable v1 (let's say x in the code)
- v2 can be found in v3 (the literal value used in declaration of x)
- variable v4 is a literal (a literal value in the code)
- v4 is not null (if there is no v4 we stop the process)
- v4 is not equal to v2 (these are two distinct instances of literals - if not stop)
- variable v6 takes the value of the literal in V2 (in this case 5 - declaration of x)
- variable v7 takes the value of the literal in V4 (in this case 5 - either line 3 or 5)
- v6 is equal to v7 (if the two literal values are the same activate the rule, otherwise stop)

The subsequent of this rule could be a suggestion to replace the literal values with the variable identifier they correspond to.

A.3 Understanding the Necessity of Variables/Constants

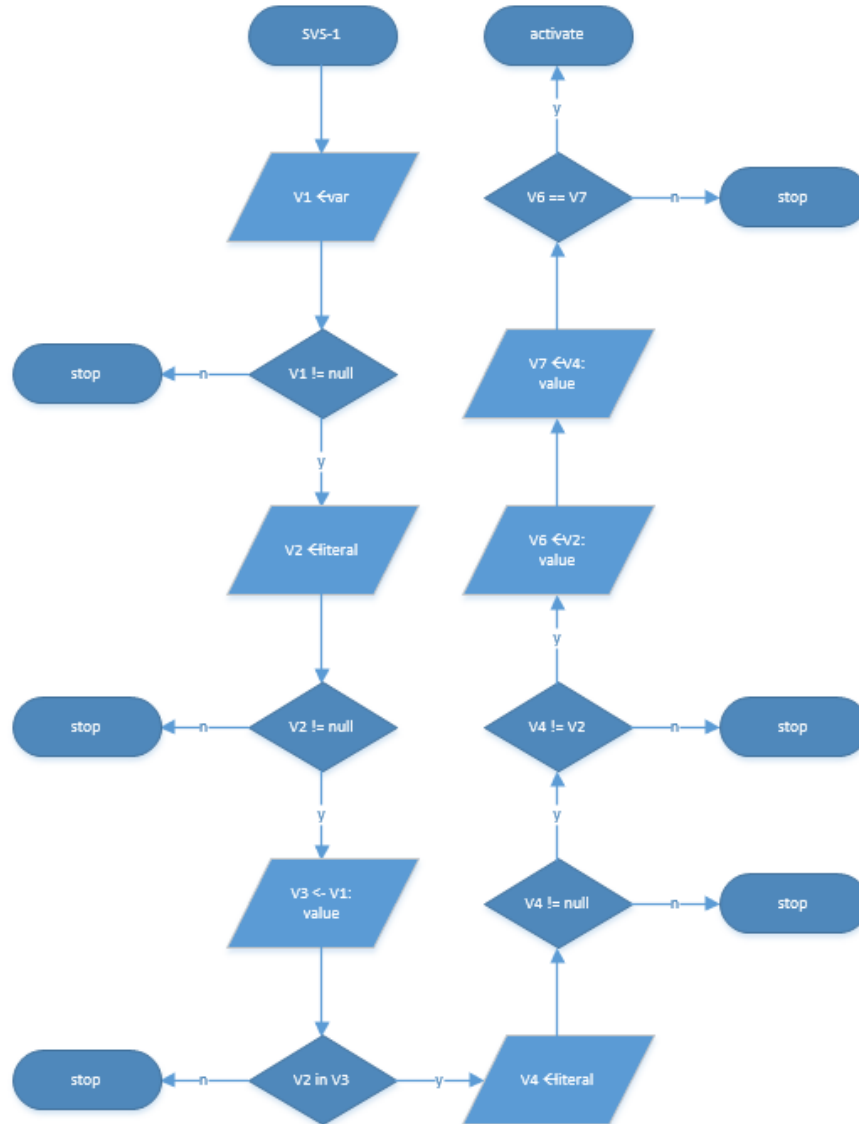
This is a situation where students use the same literal value in various places in the code instead of having a variable or constant.

A code example that illustrates the problem:

```
var x = new Array(5);    //5
var i = 0;

while(i<5)    //5
{
```

Figure A.2: Understanding the Difference Between Variable Values and Literal Values



```

if(x[i]=== 'stop')
{
    break;
}
i++;
}
  
```

```
if(i===5)    //5
{
    alert('there is no stop');
}
```

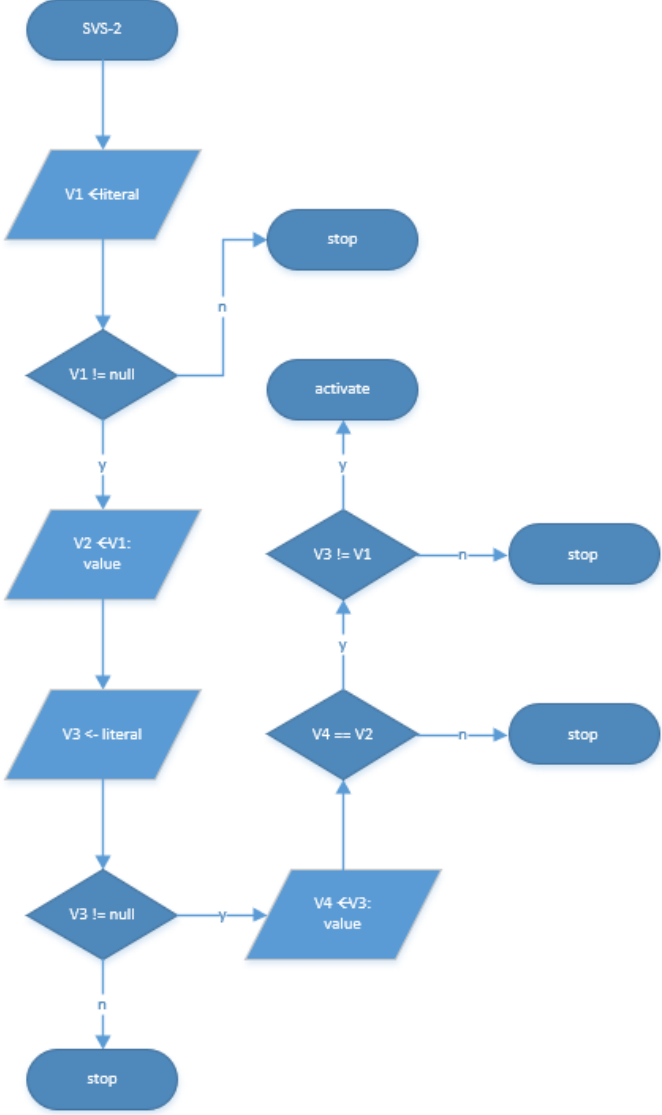
In the code snippet above there are three instances of that problem as indicated by the comments.

The description of the rule follows:

- variable v1 is a literal (a literal value in the code)
- v1 is not null (if there is no v1 we stop the process)
- variable v2 takes the value of the literal in V1 (in this case 5, line 1)
- variable v3 is a literal (a literal value in the code)
- v3 is not null (if there is no v3 we stop the process)
- variable v4 takes the value of the literal in V3 (in this case 5, line 4)
- v4 is equal to v2 (if the two literal values are the same carry on, otherwise stop)
- v3 is not equal to v1 (if these are two distinct instances of literals activate the rule otherwise stop)

The subsequent of this rule would be a suggestion to replace the literal values with references to a variable.

Figure A.3: Understanding the Necessity of Variables/Constants



A.4 Understanding the Necessity of Variables when Referring to Array Length

There are situations where students declare arrays and refer to their length using a literal value. This is most likely the case when a traversal is performed with a loop. Regardless of whether the array size remains the same or not, in general, it is good

practice to refer to the array itself and ask for its length, given that the language supports it, or measure the length and store the information in a variable. In any case it is the variable that should be used to refer to this value in the code and not a literal.

A code example that illustrates the problem:

```
var x = [2,5,1,8,9];
var sum = 0;

for(var i = 0; i < 5; i++)
{
    sum += x[i];
}

alert(sum);
```

The description of the rule follows:

- variable v1 is an array (an array value in the code)
- v1 is not null (if there is no v1 we stop the process)
- variable v2 is a for (a for control structure in the code)
- v2 is not null (if there is no v2 we stop the process)
- variable v3 takes the condition test of the for structure referenced by V2
- variable v4 takes the length of the array referenced by V1
- variable v5 is a literal (a literal value in the code)
- v5 is not null (if there is no v5 we stop the process)

- variable v6 takes the value of the literal location in code referenced by v5
- v7 takes the value of the literal in V5 (in this case 5)
- v7 is equal to v4 (if the literal value is equal to the array length move on otherwise stop)
- v8 takes the value of the for location in code referenced by v2
- v8 contains v6 (if the literal is located within the for loop move on, otherwise stop)
- v5 can be found in v6 (if the literal value is used in the condition test of the for loop activate the rule, otherwise stop)

The subsequent of this rule would be a suggestion to replace the literal values with a reference to the array length property.

A.5 Understanding off-by-one Errors with Arrays in Loops

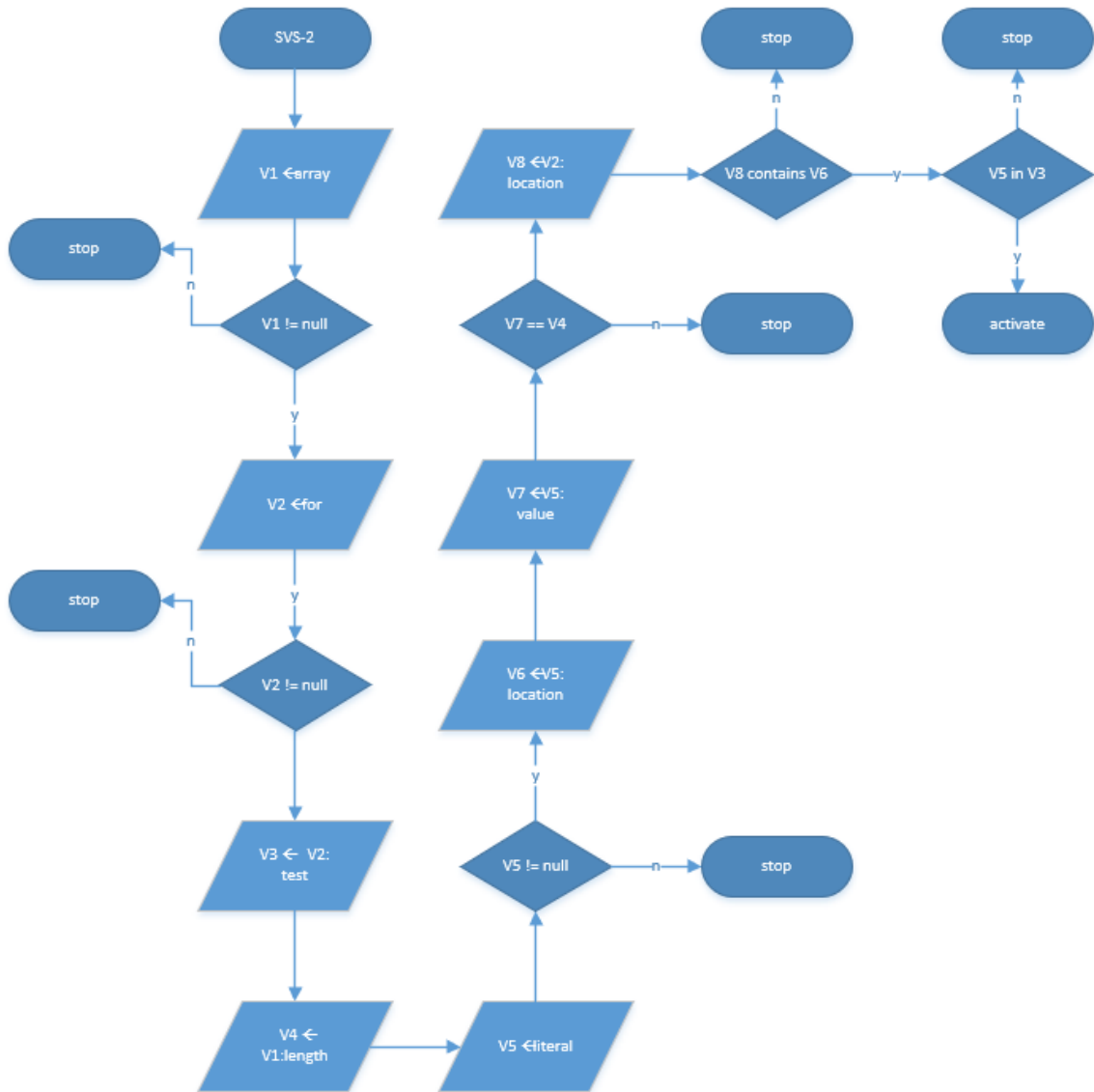
This is a very common mistake that even experienced programmers make sometimes especially when they switch between languages that support different array indexing schemes. The problem is that novice programmers fail to understand that when doing an array traversal in a zero-based indexed array they have to stop referencing elements just before the iterator becomes equal to the length of the array.

A code example that illustrates the problem:

```
var x = [2,5,1,8,9];
var sum = 0;

for(var i = 0; i <= 5; i++)
```

Figure A.4: Understanding the Necessity of Variables when Referring to Array Length



```

{
    sum += x[i];
}

```

```

alert(sum);

```

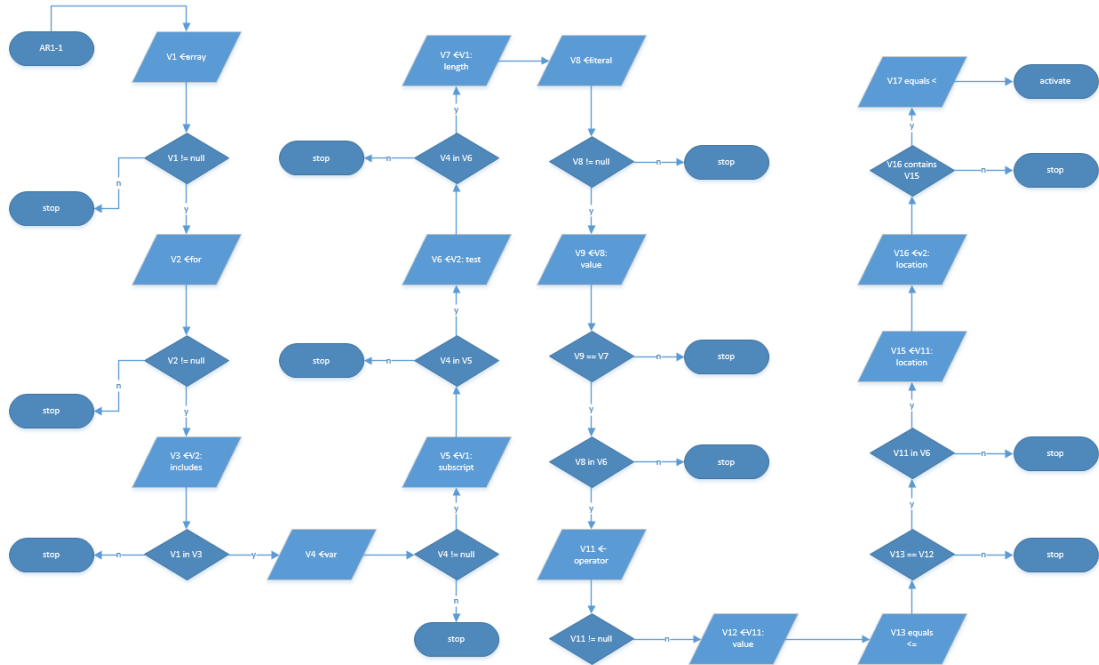
The description of the rule follows:

- variable v1 is an array (an array value in the code)
- v1 is not null (if there is no v1 we stop the process)
- variable v2 is a for (a for control structure in the code)
- v2 is not null (if there is no v2 we stop the process)
- variable v3 takes the body of the for structure referenced by V2
- v1 can be found in v3 (if the array is in the body of the loop move on, otherwise stop)
- variable v4 is a var (var declaration in the code - variable i in this case)
- v4 is not null (if there is no v4 we stop the process)
- variable v5 takes the subscript of the array referenced by V1 ([i] in the code example)
- v4 can be found in v5 (if variable i can be found in the array's subscript move on, otherwise stop)
- variable v6 takes the condition test of the for structure referenced by V2
- v4 can be found in v6 (if variable i can be found in for condition test move on, otherwise stop)
- variable v7 takes the length of the array referenced by V1 (in this case 5)
- variable v8 is a literal (a literal value in the code)
- v8 is not null (if there is no v8 we stop the process)
- variable v9 takes the value of the literal referenced by V8 (literal 5 in this example)


-
- v9 is equal to v7 (if the literal value is equal to the array length move on, otherwise stop)
 - v8 can be found in v6 (if the literal value can be found in the condition test move on, otherwise stop)
 - variable v11 is an operator (an operator in the code)
 - v11 is not null (if there is no v11 we stop the process)
 - variable v12 takes the value of the operator referenced by V11 (j= in this example)
 - v13 takes the value of the operator j=
 - v13 is equal to v12 (if the operator found is j= move on, otherwise stop)
 - v11 can be found in v6 (if the operator can be found in the condition test move on, otherwise stop)
 - variable v15 takes the location of the operator referenced by V11 (j= in this example)
 - variable v16 takes the location of the for loop referenced by V2
 - v16 takes the location of the for loop referenced by V2
 - v16 contains v15 (if the operator is located within the for loop move on, otherwise stop)
 - v17 takes the value of the operator j (give the value that is needed for refactoring the operator, activate the rule)

In this case the assumption is that the language supports only zero-based indexes for arrays. Therefore, we know that traversal cannot go beyond $n-1$. The very last statement in the rule definition encodes within the rule a value that is supposed to be used for refactoring the code, if the rule gets executed.

Figure A.5: Understanding off-by-one Errors with Arrays in Loops



The subsequent of this rule would be a suggestion to replace the operator used in the test with another one.

A large, bold, black serif letter 'B' is centered within a light gray square background.

Literature Review Strategy Used

In every research project it is essential to study and understand what has already been achieved in the area under investigation. Typically this is done through finding, analysing, evaluating, and synthesising the contents of related empirical and conceptual publications. Academic publications comprise a large corpus of documents that continuously grows at an exponential rate and that makes the selection of relevant sources a challenging process. The strategy to identify and select relevant sources may have a significant impact in the timeframe and the overall effectiveness of the process. The general strategy used in this work to approach and select existing knowledge was

a combination of the Pearl Growing technique (Schlosser et al. 2006) and the Snowballing technique (Jalali & Wohlin 2012). Initially iterative searches were performed on information sources and databases based on criteria to identify the key review papers in the area. Keywords of important papers found were gradually introduced into the searches to refine queries for relevant sources. Review papers offer a solid starting point for research projects as they can give a first clear outline of the literature in the area without too much detail (Sylvester et al. 2013). Their overarching purpose is to synthesise the literature in a field and provide a clear overview of the area in question. They also offer references to the most useful primary sources (Cronin et al. 2008) and they are typically cited more often than any other type of published article (Cronin et al. 2008, Montori et al. 2003, Patsopoulos et al. 2005). This characteristic significantly increases the chances to find relevant work both prior and after the publication date of the article. After analysis of the review papers the next step was to identify and select the references of interest and process them individually. Citations of these papers offer new references and the process carries on until there is no more relevant work to consider. This technique is called backward snowballing. In some cases the forward snowballing technique was also used to identify more recent papers than the sources. The Google scholar engine was used for both cases.



Observation Sheet

Section 1: General Info		
Date:	Course Level:	
Course Title:		
No of Participants	Students:	Tutors:

Section 2: Learners and the Learning Process
2.1 Understand how learners approach learning
2.2 Understand how learners solve their problems

Section 3: Teachers and the Teaching Process
3.1 Understand processes
3.2 Identify Flaws and Weaknesses
3.3 Recognise Good Practices

Section 4: Verify Findings in the Literature	
Learning programming is difficult	
Teaching programming is expensive	
Supporting learning programming is limited due to...	
Learning programming in exploratory settings offers more opportunities	
Learning programming in exploratory settings is more natural	
The need for support in exploratory learning is higher than in guided learning	

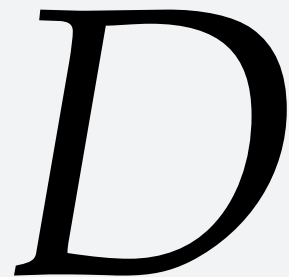
Section 5: Further Understanding of the Area of Interest
Notes that reveal hidden aspects or shed light to particular areas of interest

Section 6: Student Code

Area for code that requires attention

Section 7: Identify Potential Opportunities for Improvement and Innovation

Area for spontaneous brainstorming based on the recent personal experience
--



Usability Test Material

D.1 Day 1

1 Logical Propositions

Consider the following statements:

- Two trees gave 2 apples of variety A each.
- 4 more apples are available in a food refrigerator storage but only half of them

are of variety A.

- All apples of variety A undergo 2 levels of food quality testing.
 - Test 1 result = 2
 - Test 2 result = 3
- Apples of variety A can move to the next production phase only if
 - the apples available are at least 2 and
 - the tests give different results

Create variables for the items in the above description and give appropriate values. Create logical propositions for each statement using those variables. Evaluate the last proposition and consider the result (display the result in the console). Is this what you expected?

2 BMI

The body mass index (BMI) is a measure that uses your height and weight to work out if your weight is healthy. The formula is $BMI = kg/m^2$ where kg is a person's weight in kilograms and m^2 is their height in metres squared. Write a program that calculates your BMI. Use variables for height and weight. Values must be given in cm and kg respectively. Compare the result with an online BMI calculator.

3 BMI+

Extend the solution you gave for exercise No 2 and make your code accept input from the user. Values must be given dynamically so that your program can be reusable by any number of users without modifications. [Note: check the type of input values be-

fore you use them in calculations.. is this what you expect?]

4 BMI++

Extend the solution you gave for exercise No 3 so that the code generates a classification in textual form along with the index value. For adults the classifications are the following:

- underweight (under 18.5)
- normal weight (18.5 to 24.9)
- overweight (25 to 29.9)
- obese (30 or more)

[Note: check if you can reduce the number of conditions in your control structure]

5 BMI+++

Extend the solution you gave for exercise No 4 so that the code can perform input validation. The accepted ranges follow:

Height: shortest (0.5m) - tallest (2.5m)

Weight: lightest (2kg) - heaviest (600kg)

The program should not permit invalid input. If a value is out of range the code prompts the user to input it again. The process stops only when valid input is received.

[Note: which type of loop is more appropriate for this type of control?]

6 BMI Statistics

Write a program that accepts any number of BMI values in a row and displays the min, max and average of those values in the console. The values must be given dynamically by the user at runtime and input stops when the value -1 is given. Input validation is not necessary.

[Notes: which type of loop is more appropriate for this type of control?, is the result what you expect? check the type of input values]

7 BMI Statistics+

Write a program that accepts 10 BMI values in a row and displays the average of those values in the console as well as the values above the average. The values must be given dynamically by the user at runtime. Input validation is not necessary.

[Notes: which type of loop is more appropriate for this type of control?]

8 BMI Statistics++

Write a program that accepts any number of BMI values in a row and calculates a frequency distribution table for the classifications given in exercise 4. Input validation is not necessary. The distribution table should be displayed on the screen in the following form:

```
underweight [3]: ***
normal weight [1]: *
overweight [2]: **
obese [4]: ****
```

D.2 Day 2

1 Integer Division

Numeric values are represented in JavaScript floating point signed numbers. There is no concept of integer value in the language. Therefore, integer division is not natively supported. The division operator always returns a floating point number.

Write the function `quotient(n,d)` that takes two values as arguments (numerator and denominator) and computes the quotient of the division operation. The value should be returned as an integer. The function should be able to deal with negative numbers as well.

[Note: you may find the method `toString()` and the function `parseInt()` useful in your implementation]

2 Binary to Decimal Conversion

Write the function `binToDec(bin)` that takes a number in binary and returns the same number in decimal. The input should be given as an array of integers (zeroes and ones). You should not assume a specific length for the array. You can compare your results with the function `parseInt(bin.join(""), 2)` that performs the same operation.

[Note: you may find the method `Math.pow()` useful in your implementation]

3 Russian Peasant Multiplication

Write the function `peasantMult(a,b)` that takes two integer values and performs multiplication using the Russian peasant algorithm.

[Note: you may find the quotient function useful in your implementation]

4 Russian Peasant Multiplication+

Amend the previous function so that it performs the same operation without using the multiplication operator and the quotient function.

[Note: you may find the shift operators useful in your implementation]