UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCES AND ENGINEERING

Empowering Intelligent Classrooms with Attention Monitoring and Intervention Cycles

by

Maria Korozi

PhD Dissertation

Presented

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

Heraklion, November 2017

UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE

Empowering Intelligent Classrooms with Attention Monitoring and Intervention Cycles

PhD Dissertation Presented
by **Maria Korozi**
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

APPROVED BY:

_____

Author: Maria Korozi

_____

Supervisor: Constantine Stephanidis, Professor, University of Crete

_____

Committee Member: Antonis Argyros, Professor, University of Crete

_____

Committee Member: Anthony Savidis, Professor, University of Crete

_____

Committee Member: Panos Markopoulos, Professor, Technische Universiteit Eindhoven, Netherlands

_____

Committee Member: Xenophon Zabulis, Principal Researcher, FORTH-ICS

_____

Committee Member: Dimitris Grammenos, Principal Researcher, FORTH-ICS

_____

Committee Member: Margherita Antona, Principal Researcher, FORTH-ICS

_____

Department Chairman: Angelos Bilas, Professor, University of Crete

Heraklion, November 2017

This thesis is dedicated to the person who …

Inspires me

Motivates me

Makes me stronger

… my daughter Anastasia

Acknowledgments

I would like to thank my thesis supervisor, Professor Constantine Stephanidis for his support, assistance and continuous guidance throughout my Ph.D. studies. I am also grateful to professors Antonis Argyros and Anthony Savidis who helped me improve and evolve my work with their valuable feedback and critical thinking. I would also like to acknowledge Dr. Margherita Antona for her guidance and valuable advices which contributed the most to shaping this work into its current state.

Moreover, I owe my appreciation to the undergraduate students Evropi Stefanidi and Maria Doulgeraki for their contribution in the design and implementation of LECTORviewer, and to the M.Sc. student Anastasia Ntagianta for her contribution in the development life-cycle of CognitOS.

In addition, I would also like to thank Asterios Leonidis who as a colleague guided me throughout the design and development process of this work and as my spouse equipped me with all the strength and courage I needed. The biggest thank you though goes to my little daughter Anastasia who is the source of my inspiration.

I also want to thank the Department of Computer Science of the University of Crete and the Institute of Computer Science - FORTH-Hellas (Human Computer Interaction Laboratory) for equipping me with the means to achieve my research goals.

Finally, I need to thank my parents Costas and Dimitra who supported me and my family in every way possible, especially during the final months of my dissertation.

Abstract

The primary goal of an AmI environment is to help and support the people living in it; towards that objective it should be able to identify a user need and act accordingly. Many research approaches and commercial tools have focused on realizing this concept, which follows the paradigm of the trigger-action model; however, the majority of them poses several limitations (e.g., one trigger can be connected to a single action, artifact-oriented triggers mainly). The domain of education would particularly benefit from an AmI environment able to monitor students during their educational activities and intervene when deemed necessary to help, support or motivate them so as to promote the learning process. Nevertheless, despite the fact that the concept of the Intelligent Classroom has gained much attention from researchers over the past decade, none of the approaches proposed so far offers a generic, scalable, fast and easy way to connect triggers with actions.

Aiming to bridge this gap, this thesis presents a framework and an authoring tool that support both developers and educators in defining the **behaviors** (triggers) that lead to context-aware **interventions** (actions). Following an extensive literature review and an iterative elicitation process –based on multiple collection methods such as brainstorming, focus groups, observation and scenarios– the high-level functional and non-functional requirements that both the framework and authoring tool should satisfy were identified. Based on those findings, this work aims to equip the Intelligent Classroom with mechanisms that monitor the learners' attention levels and intervene, when necessary, to (i) provide motivating activities to distracted, unmotivated or tired individuals or (ii) suggest to educators alternative methodologies which would be beneficial for the entire classroom.

In more detail, the LECTOR framework offers a mechanism for identifying student **behaviors** that require remedial actions and intervening when the

students need help or support. This mechanism relies on "if-then" rules - created either by developers or by educators- to define the behavior of the classroom environment. In order to ensure scalability and simplify rules' management, a three (3) step process for connecting a behavior with an intervention was introduced. In more detail, the first step requires the user to define a **behavior**, next the conditions under which the behavior becomes a **trigger** have to be described, and during the last step connections between a trigger and appropriate **interventions** are created. This decomposition permits a behavior to be associated with multiple triggers, and a trigger with multiple interventions that alternate depending on the context of use. Furthermore, in contrast to the artifact-oriented recipes that are currently supported by the majority of such tools, LECTOR's rule structure supports the creation of **user-oriented** intervention scenarios.

Furthermore, LECTOR introduces a sophisticated authoring tool, named LECTORstudio, which aims to support both developers and educators in creating rules that describe behaviors, triggers, and interventions. However, since developers require further assistance in order to integrate the appropriate building blocks necessary for programming the AmI environment (e.g., actors, context, interventions), LECTORstudio also provides such functionality through intuitive user interfaces.

Lastly, in order to further support the targeted end-users of the Intelligent Classroom environment (i.e., students and educators), this work also features three (3) additional tools: LECTORviewer, NotifEye and CognitOS. The former two (namely LECTORviewer and NotifEye) aim to support educators in having an overview of the students' attention levels and providing their input regarding ambiguous behaviors or scheduled interventions that aim to re-engage distracted, tired or unmotivated students. CognitOS on the other hand, is a sophisticated web-based working environment for students that hosts a variety of educational applications, which constitute the communication channels through which LECTOR presents the interventions.

Περίληψη

Ο πρωταρχικός στόχος ενός περιβάλλοντος Διάχυτης Νοημοσύνης (ΔΝ) είναι να βοηθά και να υποστηρίζει τους ανθρώπους που ζουν και κινούνται μέσα σε αυτό. Για αυτό το λόγο, ένα τέτοιο περιβάλλον θα έπρεπε να μπορεί να αναγνωρίσει τις ανάγκες των χρηστών του και να αντιδρά κατάλληλα για να τις ικανοποιήσει. Αυτή η ιδέα δεν είναι καινούρια, πολλές ερευνητικές μελέτες αλλά και εμπορικά εργαλεία έχουν προσπαθήσει να την υλοποιήσουν ακολουθώντας το διαδεδομένο μοντέλο «έναυσμα-δράση» (trigger-action). Ωστόσο, οι περισσότερες από αυτές τις προσεγγίσεις έχουν αρκετά μειονεκτήματα όπως το γεγονός ότι ένα έναυσμα (trigger) μπορεί να οδηγήσει σε μία και μόνο δράση (action) ή την απαίτηση τα περισσότερα εναύσματα (triggers) να προέρχονται από συγκεκριμένες καταστάσεις συσκευών και όχι από ανθρώπινες ενέργειες.

Ο τομέας της εκπαίδευσης θα επωφελούνταν τα πλείστα από ένα περιβάλλον Διάχυτης Νοημοσύνης που έχει τη δυνατότητα να παρακολουθεί τους μαθητές κατά τη διάρκεια των εκπαιδευτικών τους δραστηριοτήτων και να επεμβαίνει –όταν κριθεί απαραίτητο– για να τους βοηθήσει και να τους υποστηρίξει υποβοηθώντας με αυτόν το τρόπο την διαδικασία μάθησης. Παρόλο που η ιδέα της Έξυπνης Τάξης βρίσκεται ήδη αρκετά χρόνια στο προσκήνιο, καμία από τις έως τώρα διαθέσιμες προσεγγίσεις δεν προσφέρει ένα γενικό, επεκτάσιμο, γρήγορο και εύκολο τρόπο να συνδέονται εναύσματα (triggers) με δράσεις (actions).

Προσπαθώντας να καλύψει αυτό το κενό, αυτή η διατριβή παρουσιάζει μία υποδομή (framework) και ένα εργαλείο συγγραφής (authoring tool) για να διευκολύνει τους προγραμματιστές αλλά και τους εκπαιδευτικούς να συνδέσουν συμπεριφορές μαθητών (εναύσματα) με παρεμβάσεις (δράσεις). Ακολουθώντας μία εκτενή βιβλιογραφική έρευνα και μία επαναληπτική διαδικασία συλλογής πληροφοριών –βασισμένη σε πολλές μεθόδους όπως κατιδεασμός, ομάδες

εστίασης, παρατήρηση και κατασκευή σεναρίων– αναγνωρίστηκαν οι βασικές λειτουργικές και μη-λειτουργικές απαιτήσεις του συστήματος. Έχοντας ως βάση αυτά τα ευρήματα, αυτή η εργασία στοχεύει να εξοπλίσει την Έξυπνη Τάξη με μηχανισμούς που παρακολουθούν τα επίπεδα προσοχής των μαθητών και επεμβαίνουν όταν είναι απαραίτητο για: (α) να παρέχουν συναρπαστικές δραστηριότητες σε μαθητές που έχουν χάσει το κίνητρο να συμμετέχουν ή έχουν κουραστεί ή έχουν αποσπαστεί από τις τρέχουσες δραστηριότητες, ή (β) να προτείνουν εναλλακτικές μεθοδολογίες στους καθηγητές για να τα κερδίσουν την προσοχή του συνόλου των μαθητών της τάξης.

Συγκεκριμένα, η υποδομή LECTOR προσφέρει έναν μηχανισμό που χρησιμοποιεί τις τεχνολογίες που υπάρχουν ήδη στην Έξυπνη Τάξη για να αναγνωρίζει συμπεριφορές μαθητών που απαιτούν επανορθωτικές ενέργειες και να επεμβαίνει όταν οι μαθητές χρειάζονται βοήθεια ή υποστήριξη. Αυτός ο μηχανισμός βασίζεται σε κανόνες της μορφής «if-then» –που έχουν δημιουργηθεί είτε από προγραμματιστές είτε από εκπαιδευτικούς– για να ορίσει τη συμπεριφορά του περιβάλλοντος της τάξης. Επιπλέον, εισάγεται μία διαδικασία τριών (3) βημάτων για τη σύνδεση συμπεριφορών και παρεμβάσεων. Αρχικά απαιτείται ο ορισμός των συμπεριφορών, στη συνέχεια ο ορισμός των συνθηκών κάτω από τις οποίες μία συμπεριφορά αποτελεί έναυσμα για παρέμβαση, και τέλος η σύνδεση των συμπεριφορών με τις κατάλληλες παρεμβάσεις. Αυτός ο κατακερματισμός επιτρέπει μια συμπεριφορά να συνδεθεί με πολλαπλά εναύσματα (triggers), και ένα έναυσμα με πολλαπλές παρεμβάσεις. Επιπλέον, ένα ακόμα χαρακτηριστικό των κανόνων που μπορούν να δημιουργηθούν είναι ότι υποστηρίζουν τη δημιουργία ανθρωποκεντρικών και όχι συσκευο-κεντρικών σεναρίων, εν αντιθέσει με τους κανόνες που ήδη υποστηρίζονται από την πλειοψηφία των σχετικών εργαλείων.

Επιπροσθέτως, ο LECTOR προσφέρει ένα εργαλείο συγγραφής (authoring tool) που ονομάζεται LECTORstudio και στοχεύει να βοηθήσει τόσο τους προγραμματιστές όσο και τους εκπαιδευτικούς να δημιουργούν κανόνες που περιγράφουν συμπεριφορές, εναύσματα, και παρεμβάσεις. Ωστόσο, επειδή οι

προγραμματιστές χρειάζονται επιπλέον εργαλεία για να ενσωματώσουν τα δομικά στοιχεία που είναι απαραίτητα για τον προγραμματισμό του περιβάλλοντος της Έξυπνης Τάξης (π.χ., πλαίσιο χρήσης, παρεμβάσεις, συσκευές παρουσίασης παρεμβάσεων), το LECTORstudio προσφέρει την κατάλληλη λειτουργικότητα μέσα από εύχρηστες γραφικές διεπαφές.

Τέλος, έχοντας ως στόχο την υποστήριξη των τελικών χρηστών της Έξυπνης Τάξης (δηλ. των μαθητών και των εκπαιδευτικών), αυτή η εργασία προσφέρει επιπλέον τρία (3) εργαλεία: LECTORviewer, NotifEye και CognitOS. Τα δυο πρώτα (LECTORviewer και NotifEye) έχουν ως στόχο να προσφέρουν στους εκπαιδευτικούς τη δυνατότητα εκτενούς εποπτείας των εναυσμάτων που ανιχνεύονται από τον LECTOR και έκφρασης της δικής τους εκτίμησης για διφορούμενες συμπεριφορές ή για παρεμβάσεις που πρόκειται σύντομα να ξεκινήσουν. Τέλος, το CognitOS αποτελεί ένα περιβάλλον εργασίας για μαθητές το οποίο, μεταξύ άλλων, φιλοξενεί πλήθος εκπαιδευτικών εφαρμογών που χρησιμοποιούνται από το LECTOR ως τα κανάλια παρουσίασης και εφαρμογής παρεμβάσεων.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

General Objective

"*Ambient Intelligence (AmI) is about sensitive, adaptive electronic environments that respond to the actions of persons and objects and cater for their needs*".

Aarts & Wichert [1]

The above definition highlights the human-oriented nature of AmI environments, whose primary goal is to satisfy the needs of the people living in them. Towards that objective, an AmI environment should be able to recognize the users in it, understand their needs, identify their behavior, and act and react in their interest. Fundamentally, this process entails two basic steps (i) identifying a user need and, (ii) acting accordingly; this simplification reveals similarities with the trigger-action model [2]–[4], which in the recent years has been in the spotlight as a form of programming AmI environments, using simple "if trigger, then action" rules.

Currently, there is an abundance of commercial tools that permit not only developers, but non-technical users as well to program their environment. However, the majority of them focuses on triggers originating from physical artifacts rather than human behaviors, while they also present several constraints concerning the extensibility and scalability of the rules that can be created. Furthermore, despite the fact that the Intelligent Classrooms already

foster a variety of AmI technologies, currently there is no easy way to combine them and create custom trigger-action scenarios. To this end, this work proposes a framework and a sophisticated authoring tool that advances the way through which both developers and educators create the conditions that dictate how certain human-oriented behaviors lead to context-aware interventions, and applies such framework in the context of the intelligent classroom.

Motivation & Vision

Ambient Intelligence (AmI) environments [1] are expected to transparently interact with the users either passively, by observing and trying to interpret their actions and intentions, or actively by learning users' preferences and adapting their behavior accordingly to improve the quality of life. Particularly, according to [5] "AmI is a user-centric paradigm, it supports a variety of artificial intelligence methods and works pervasively, non-intrusively, and transparently to aid the user". In order to be able to act as described, the architecture of an AmI Environment should consist of four main layers [6], namely: (i) Sensing, (ii) Networking, (iii) Perception and Reasoning, and (iv) Acting. It is obvious that such environments evolve around the needs of their users and their main objective is to act in an appropriate manner when deemed necessary. Indeed, providing the right type of help or support as soon as the user needs it, is imperative in many application domains such as Ambient Assisted Living (AAL), eHealth, Domestic Life, Learning and Education, etc.

The domain of AAL is already benefited s from a large number of systems aiming to facilitate independent living for the elderly and people with disabilities. Indicatively, over the past few years many researchers have focused on creating reliable fall detection systems, which in case of emergency notify the emergency-personnel, care-takers, and/or family members [7]–[9]. Similarly, in the case of eHealth, many applications are being designed to provide assistance under various situations, including

diabetes [10], asthma [11], obesity [12], [13], smoking cessation [14], stress management [15], [16], and depression treatment [17]; most of these approaches are able to provide specific remedial actions (e.g., automated real-time educational and behavioral messaging) as soon as an irregularity is identified.

In the case of domestic life, the advancement of Internet of Things (IoT) [18] in combination with cloud computing [19] has led to an abundance of web-enabled devices and services for smart homes [20]. In such environments, individual users in their everyday lives interact frequently with various smart artifacts in conjunction; for that reason, smart homes became the most popular testbed for creating automated tasks based on the preferences, needs and daily routines of their inhabitants. Developers can manipulate the intelligent facility of a smart home to create useful integrations (e.g., "if nobody is at home, turn off all unnecessary electronic devices", "Save to Dropbox all attachments from incoming emails"). However, a new type of programming environment is emerging that allows non-technical users to specify their own logic over the simple, yet effective trigger-action model [21]–[27]. Through such environments, users can define triggers (e.g., "if temperature falls below 18^{o}C") that initiate specific actions (e.g., turn on the heating) when their conditions are met, thus automating common tasks.

Programming the behavior of smart homes came quickly to the spotlight and an increasing number of solutions appeared, enabling people without programming experience to create their own configurations. However, that is not the case for domains where **triggers** go beyond simple sensor readings and **actions** play the role of interventions rather than mere automations (e.g., AAL, eHealth, Education). For example, in a classroom environment a student "talking" should **not** trigger a remedial action immediately. On the contrary, other contextual information (e.g., "the fact that teacher is talking", "there is an ongoing exam") should be taken into consideration before reaching a conclusion.

In the context of this work, the notion of **behavior** seems more appropriate for describing the triggers that originate from or revolve around human activities. According to [28], in the domains of biology, chemistry, physics and psychology the word behavior is defined as "the way that a person, an animal, a substance, etc. behaves in a particular situation or under particular conditions". In addition, taking into consideration the user-oriented nature of AmI, the term **intervention** ("*involvement in a difficult situation in order to improve it or prevent it from getting worse"* [29]) seems more appropriate for describing the actions that the environment undertakes to support its users. Furthermore, since AmI environments are expected to behave intelligently and adapt their behavior according to user needs [5], any intervention should be selected carefully and be delivered in a ubiquitous manner based on the current context. Consider the following example where during a Physics course many students (60% of the classroom) show signs of tiredness; since a large percentage of the classroom is affected, the system should be able to select an intervention that targets a group of people and display it on a public artifact such as the classroom board.

The domain of Education would particularly benefit from an ambient system that monitors the learners' attention levels and intervenes, when necessary, to (i) provide motivating activities to distracted, unmotivated or tired individuals or (ii) suggest alternative learning methodologies to educators, which would be beneficial for all students. External stimuli such as pictures and sounds or internal stimuli such as personal thoughts usually distract learners during a course. Additionally, feelings of fatigue (i.e., Drowsiness and Falling Asleep), which are a common phenomenon inside a traditional classroom environment, could directly affect the behavior of a student. Observing student behaviors can reveal such attention lapses; hence, a system that is able to initiate appropriate interventions when needed seems essential. Despite the fact that AmI has already permeated the classroom environment, there is

currently no support for linking identified behaviors with interventions, nor for educators to create their own integrations.

These observations nurtured the idea of a framework that supports both developers and educators in defining the conditions under which certain **behaviors** lead to context-aware **interventions**, via a **user-friendly programming environment**. In more detail, the envisioned framework aims to:

- Support the rapid creation of **user-oriented** behavior-intervention (trigger-action) scenarios in contrast to the artifact-oriented recipes that are currently supported by the majority of IFTTT-style tools [21], [27].
- Enable behavior modelling by combining **multiple contextual information.**
- Support the connection of N behaviors with M interventions.
- Permit the definition of multimodal and ubiquitous interventions.
- Provide a mechanism for assessing the efficacy of interventions.

The approach

This thesis introduces LECTOR, a framework responsible for (i) monitoring the Intelligent Classroom environment, (ii) detecting student behaviors that require remedial actions (e.g., inattentive behaviors), and (iii) selecting appropriate interventions in order to help or support them throughout the educational process.

LECTOR exploits the potential of AmI technologies to observe student behaviors (SENSE), identify whether they require remedial actions (THINK) and intervene accordingly -when deemed necessary- in order to fulfill their needs (ACT). According to cognitive psychology, the sense-think-act cycle stems from the processing nature of human beings that receive input from the environment (perception), process that information (thinking), and act upon the decision reached (behavior) [30]. This identified pattern constitutes the

base for many design principles regarding autonomous agents and traditional AI [31].

Furthermore, the SENSE-THINK-ACT model is extended here by introducing the notion of LEARN. The fact that the nature of this system enables continuous observation of behaviors creates the foundation for a mechanism that provides updated knowledge to the decision-making components. In more detail, the LEARN-ing mechanism is able to (i) incorporate knowledge provided by end-users in order to disambiguate identified behaviors and assess the acceptance of an intervention, and (ii) auto-rank the suggested interventions according to their efficacy.

In order to support both developers and educators in defining the conditions under which certain **behaviors** lead to context-aware **interventions**, LECTOR features a sophisticated tool, named LECTORstudio. Since the decision-making mechanisms of LECTOR rely on **rule-based** conditions, LECTORstudio supports the creation of three (3) types of rules:

I. Rules that "model" a **behavior**[5] based on **physical context**[3].
II. Rules that "model" the **triggers**[6] based on the **behavior**[5] of **actors**[1] under specific **virtual context**[4].
III. Rules that describe the conditions (**triggers**[6] and **virtual context**[4]) under which an **intervention**[7] is initiated on a specific **intervention host**[2].

Even if this decomposition increases the number of steps that a user must complete in order to connect a trigger to an intervention, it offers scalability and better rule management. In particular, the three ingredients (i.e., behavior, trigger, intervention) are defined in isolation and are only connected in terms of their outcome. Therefore, an ingredient can be modified independently of the others and as long as its outcomes remain the same, no more adjustments will be required for the system to continue to operate as prior to the change. This approach not only minimizes unwanted

ramifications, but also facilitates collaboration as new rules can be easily created by different users given that their "connection points" will always be their outcomes. This is inspired by how an Application Programming Interface (API) simplifies programming and enables computer programs to evolve independently by abstracting the underlying implementation and only exposing objects the developer needs.

The core concepts of this rule-based approach are explained below:

1. **Actors** are the (groups of) users of the intelligent environment whose behavior needs to be monitored in order to decide whether an intervention is required (i.e., student, teacher and classroom).
2. **Intervention hosts** can either launch an application with specific content or control the physical environment. They are: (i) common computing devices such as smartphones, tablets, and laptops or (ii) technologically augmented everyday physical objects (e.g., interactive white boards, smart lamps, etc.), or (iii) custom made items (e.g., student desk).
3. The **physical context** encapsulates information regarding physically observable phenomena via sensors (e.g., luminance, heart rate, sound levels, etc.).
4. The **virtual context** refers to any static or dynamic information that is provided through software components (e.g., student profiles, course schedule).
5. **Behavior** is a model that represents the actions of an actor (e.g., a student talks, a teacher is walking).
6. **Trigger** is the model of a high level behavior that can initiate an intervention.
7. **Interventions** are the system-guided actions that aim to help or support students during the educational process.

In order to support the creation of such rules in the Intelligent Classroom environment, LECTORstudio permits developers to integrate the necessary

building blocks (i.e., actors, intervention hosts, physical context, virtual context, interventions).

Contribution

The primary contributions of this thesis are:

A. **Mechanisms** that empower the Intelligent Classroom to: (i) **identify behaviors** that require remedial actions by taking advantage of its AmI facilities and (ii) **intervene** when students need help or support.
B. A **user-friendly authoring tool** for supporting both educators and developers in creating –in a rapid and easy way– the conditions under which certain behaviors lead to context-aware interventions. Specifically, users guide the generation of "if-then" rules that dictate the behavior of the Intelligent Classroom.

Secondary contributions are the following:

- A three (3) step process for connecting behaviors with interventions; this decomposition, permits a behavior to be associated with many triggers and a trigger with many interventions, depending on the context of use. In more detail, the first step is to define a **behavior**, next the conditions under which the behavior becomes a **trigger** have to be described, and the last step is to create a connection between a trigger and an **intervention**.
- A rule structure that supports the creation of **user-oriented** behavior-intervention scenarios in contrast to the artifact-oriented recipes that are currently supported by the majority of IFTTT-style tools.
- Appropriate infrastructure that enables **evaluation of system decisions**, depending on user input to (i) invalidate identified behaviors, and (ii) override system suggestions in case they do not serve their needs.
- A user-friendly authoring tool that enables developers to integrate the building blocks (i.e., actors, intervention hosts, physical context, virtual context, interventions) that are required for programming the Intelligent Classroom.

- Auxiliary tools were also created to serve the needs of educators and students.

Thesis Outline

The rest of this thesis is organized in the following way:

- Chapter 2 presents a literature review that aims to (i) examine the state-of-the-art technology that concentrates on monitoring human attention, (ii) reveal methodologies for detecting human behaviors, and (iii) identify widely accepted techniques, appropriate for resetting the students' attention during lectures.
- Chapter 3 reviews the related work regarding trigger-action programming.
- Chapter 4 introduces several scenarios that motivated this work and outlines the functional and non-functional requirements of the proposed framework.
- Chapter 5 describes the System Architecture in details.
- Chapter 6 describes the functionality and user interface of the offered authoring tool. Furthermore, it explains in details the functionality of three (3) auxiliary tools which were created to serve the needs of educators and students.
- Chapter 7 concludes the thesis with a summary of the results and a discussion on possible future directions.

Chapter 2
Background Theory

Chapter 2 presents a literature review that aims to (i) examine the state-of-the-art technology that concentrates on monitoring human attention, (ii) reveal methodologies for detecting human behaviors, and (iii) identify widely accepted techniques, appropriate for resetting the students' attention during lectures.

Particularly, the survey's findings have significantly contributed to:

A. The creation of a proper knowledge base that aids LECTOR in appropriately identifying inattentive behaviors and providing suitable interventions (Chapter 5)
B. The formulation of the framework's core concepts (Chapter 5)
C. The definition of the **requirements** of LECTOR framework (Chapter 4)

Interactive technology in the classroom

In the recent past there has been growing interest in how computers and the Internet can improve the efficiency and effectiveness of education at all levels. Information and communication technologies (ICTs) are acknowledged as potentially powerful tools for educational change and reform. When used appropriately, different ICTs are claimed to help expand access to information, strengthen the relevance of education to the increasingly digital

workplace, and raise educational quality by, among others, helping make learning and teaching an engaging, active process connected to real life [32].

The students benefit particularly from the use of ICTs in education, since access to educational information is unlimited, the learning environment is enriched, collaboration is promoted, and motivation to learn is enhanced [33]. In the recent past, learning with the use of ICT was strongly related to concepts such as distance learning [34], educational games [35], intelligent tutoring systems and e-learning applications [36].

The notion of intelligent classrooms has become prevalent in the past decade [37]. Smart classroom is used as an umbrella term, meaning that classroom activities are enhanced and augmented through the use of pervasive and mobile computing, sensor networks, artificial intelligence, robotics, multimedia computing, middleware and agent-based software [38]. Following the rationale of augmented technology in the educational environments, new means of interaction - such as interactive whiteboards, touch screens and tablet PCs - have gained popularity and have become a major tool in the educational process, allowing more natural interaction. Smart classrooms, for example, may support one or more of the following capabilities: video and audio capturing in classroom [39], automatic environment adaptation according to the context of use (such as lowering the lights for a presentation) [40], lecture capturing enhanced with the instructor's annotations, delivery of personalized content [41] and information sharing between class members.

However, inside a classroom environment, students get distracted from educational activities either by internal stimuli (e.g., thoughts and attempts to retrieve information from memory) or external stimuli (e.g., pictures, sounds); hence they might not always be "present" to take advantage of all the benefits that an intelligent classroom has to offer. This observation highlights the need of a mechanism that monitors the learners, identifies inattentive behaviors and intervenes to appropriately reset attention levels.

Attention and Education

Attention is defined in psychology as the cognitive process of selectively concentrating on one aspect of the environment while ignoring other things; an indicative example is when listening carefully to what someone is saying while ignoring other conversations in a room. In other words, attention means focusing the consciousness on a stimulus or a range of stimuli by preferentially responding to them.

Attention is very often considered as a fundamental prerequisite of learning, both within and outside the classroom environment, since it plays a critical role in issues of motivation and engagement [42]. Obtaining and maintaining the students' attention is an important task in classroom management, and teachers apply various techniques for this purpose, however currently no technological support is available to monitor attention levels of students and assist teachers in obtaining optimal attention for the task at hand.

According to Packard [43], "classroom attention" refers to a complex and fluctuating set of stimulus-response relationships involving curriculum materials, instructions from the teacher and some prerequisite student behaviors (e.g., looking, listening, being quiet, etc.). Such behaviors can be rigorously classified as "appropriate" and "inappropriate" [44]. Appropriate behaviors include attending to the teacher, raising hand and waiting for the teacher to respond, working in seat on a workbook, following in a reading text, while inappropriate behaviors include (but are not limited to) getting out of seat, tapping feet, rattling papers, carrying on a conversation with other students, singing, laughing, turning head or body toward another person, showing objects or looking at another class mate. Some of the above behaviors would be in fact disruptive to some educational activities. However, the students should not be forced to spend their whole day not being children, but being quiet, docile, and obedient "young adults" [45]. On the contrary, learning can take place more effectively if students' curiosity, along with their desire to think or act for themselves, remains intact.

The application of LECTOR in the intelligent classroom does not target to creating a system obsessed with silence and lack of movement. On the contrary, an intriguing challenge is to disambiguate behaviors that could also be indicative of attempts to maintain attention. For instance, students looking around the classroom or looking away from their notes could be thinking about the material that the instructor is presenting, and fidgeting could be an attempt to increase arousal [46]. To this end, the decision-making processes for identifying attention vs. inattention should take into consideration the behavioral norms of each student before reaching any conclusions (e.g., a student doodling while thinking). Furthermore, since students who are motivated and sufficiently aroused can sustain prolonged attention [46], the main goal of the system is to motivate the students in becoming engaged in the learning activities so as to benefit most from the knowledge that the teacher and the ICTs have to offer.

How long do students pay attention?

As passive listeners, people generally find it difficult to maintain a constant level of attention over extended periods of time, while pedagogical research reveals that attention lapses are inevitable during a lecture. McKeachie [47] in his book on tips for lecturers, suggests that student attention will drift during a passive lecture, unless interactive strategies are used to hold student attention. This belief is corroborated by [48] that supports that student concentration decays in the same way during a passive lecture as does that of a human operator monitoring automated equipment, with serious implications for learning and performance. Bligh [49], in his book about how to lecture, advises that students are not likely to pay close attention to a lecture in the first 5 minutes, while they are settling down, nor during the last 5 minutes, when their attention rises and falls. Similarly, Sousa [50] suggests that students' processing of information during a lecture is dependent upon their motivation. The more motivated students pay attention longer than the less motivated. He suggests that unmotivated students pay attention for an

average of 10-20 min, which means that a teacher may see the beginning effects of attention decline after 10 minutes of lecturing. The estimation that attention degrades after 10 to 30 minutes on task has found support in several other studies [51]–[53].

However, experimental studies [54], [55] reveal that, unlike common belief [49], [50] students do not pay attention continuously for 10-20 minutes during a lecture. Instead, their attention alternates between being engaged and non-engaged in ever-shortening cycles throughout a lecture segment. This observation highlights the need for techniques that estimate students' attention lapses in real time.

Monitoring User Attention

There is a well-established body of research on monitoring human attention to determine users' vigilance, concentration level and visual focus of attention. Traditional approaches mainly focus on attention monitoring during interaction with computer-based applications, where the collected data are used either as diagnostic tools or as indications that additional actions should be taken to facilitate the task at hand.

Recent approaches are significantly influenced by the emergence of the Ambient Intelligence (AmI) paradigm and the concept of disappearing computing, and focus on real world activities (e.g., viewing exhibits in a museum, shopping, attending a meeting, driving, etc.), where the attention stimuli might be anywhere and the subjects might move unrestrictedly. The aforementioned approaches aim to facilitate a broad spectrum of human activities by:

- Analyzing statistical data that represent the user's visual fixations
- Adapting application content (e.g., learning material) based on user preferences and needs
- Suggesting actions based on users cognitive behavior (e.g., remediation)

- Interpreting attention as an input modality (e.g., human-robot interaction)
- Restoring user's vigilance in critical situations (e.g., driving)

From a technical point of view, various solutions are employed. Eye tracking is one of the most widely used methods to provide evidence of user's attention, since gaze is considered to be a good indicator of a person's attention on external objects. However, in open spaces where users' mobility is unconstrained, gaze trackers are not that accurate. As an alternative, based on the fact that humans pay attention to an external object by orienting themselves towards that object to have it in the center of their visual field [56], head pose tracking and laser-based orientation scanning are used to estimate the user's direction.

To improve accuracy, many studies augmented gaze-tracking techniques with additional cues like sound and context information. According to [56], objects which draw a person's attention can be external stimuli (e.g., pictures, sounds, etc.). Indicative examples that realize such concepts are discussed in [56], where the authors tried to predict at whom a person is looking based on who is/was speaking, and in [57] and [58], where content information (e.g., what is the next task in the list of pending tasks) is used to predict the visual focus of attention.

The next paragraphs report on state-of-the-art attention monitoring technologies and their application in laboratory and real-world scenarios.

In [59] the authors focus on real-world attention levels. Trying to clarify how things such as merchandise in a store or pictures in a museum receive attention in the real world, they propose a Sensor of Physical-world Attention using Laser scanning (SPAL). The use of a laser scanner is quite challenging, since it provides only front-side circumference of any detected objects in a measurement area, however, unlike cameras, it poses no privacy problems. SPAL includes many important factors when calculating people's attention,

i.e., lingering time, direction of people, distance between people and a target object. After extracting a human from a group of humans and detecting his/her direction in comparison with the position of the target object, the system calculates real-world attention levels by using two proposed models: the Object-based Attention (OA) and the Distance-weighted Attention (DA) models. Furthermore, they define two metrics, value and degree of real-world attention levels, to indicate and evaluate people's attention.

Much literature focuses on driver vigilance and attention monitoring. The system in [60] analyzes video sequences in order to determine the visual attention of a driver. Measurable cues, like eye blink rate and head rotation rate, are collected with a single camera placed on the car dashboard and assessed to make determination of the driver's visual attention level. As the authors claim, when compared to similar works, the proposed system goes beyond classifying visual attention with eye closure metrics, but also shows that detecting head rotation could expose other kinds of decreases in visual attention. The adopted method to track the head uses color predicates to find the lips, eyes, and sides of the face, while the results show that the system can track local lip motion like yawning. Despite the challenges posed by unexpected conditions that a moving vehicle might face, the system, as reported, performed quite well under a variety of daytime illumination levels, while it did not encounter any difficulty with the changing background.

In [61] the authors propose to track focus of attention of several participants in a meeting by modeling both the persons head movements and the relative locations of probable targets of interest in a room. Taking into consideration that when humans pay attention to an external object, they usually orient themselves towards the object of interest, so as to have it in the center of their visual field, they try to estimate a person's focus of attention based on head orientation. To this end, they extend the neural network approach to estimate the head pose in more unrestricted situations. In particular, the use of neural networks does not limit to estimating the head rotation in pan direction, but

also allows the calculation in the tilt direction as well. Concerning the detection of a user's focus of attention, they employ the hidden Markov model (HMM), which is able to incorporate knowledge about the meeting situation, i.e., the approximate location of participants in the meeting. Hence, the HMMs are applied to track whom the participants in a meeting are looking at.

In a more recent work, Stiefelhagen et al. [56] try to estimate the visual focus of attention of participants in a meeting room using multiple cues. They demonstrate that through the combination of visual and acoustic information a higher percentage is achieved on detecting the participants' focus of attention. Since visual attention is influenced by external stimuli, they also investigate whether it is possible to predict a person's focus of attention based on audio information. Hence, microphones are used to detect who is speaking while sound history is taken into account in order to improve the audio-based prediction. Similarly to their prior work [61], they use neural networks to estimate the participants' head poses which are captured by an omnidirectional camera that simultaneously tracks participants' faces. However, the proposed system is unable to handle significant movements of the meeting participants, as the current model relies on probability distributions related to participants' locations, limiting its employment in real-life applications.

Ba and Odobez [62] aim to address problems of recognizing the visual focus of attention (VFOA) of meeting participants following a different approach based on participants' head pose. To this end, the head pose observations are modeled using a Gaussian mixture model (GMM) or a hidden Markov model (HMM) whose hidden states correspond to the VFOA. Contrary to the work presented in [56], in the proposed setup, the potential VFOA of a person is not restricted to other participants, but includes environmental targets as well (a table and a projection screen), which subsequently increases the complexity of the task with more VFOA targets spread in the pan and tilt gaze space. To address the added complexity, they propose a novel approach to set the model

parameters either in GMM or HMM models that represent the associations between head poses and visual targets without collecting training data. Their method (referred to as cognitive or geometric) models the head pose of a person given his upper body pose and his effective gaze target. In this way, no training data are required to learn the parameters, but some knowledge of the 3-D room geometry is necessary. In addition, to account for the fact that people have their own head pose preferences for looking at the same given target, they adopted an unsupervised maximum a posteriori (MAP) scheme to adapt the parameters obtained from either the training data or the geometric approaches to the unlabeled head pose data of individual people in meetings.

In [63] the authors define and address the problem of finding the visual focus of attention for a varying number of wandering people (VFOA-W), determining where a person is looking when their movement is unconstrained. Trying to create a tool that automatically measures the effectiveness of printed outdoor advertisements came upon the problem of VFOA-W estimation, which poses implications in behavior understanding and cognitive science in real-world applications. The authors approach this problem by offering a multi-person tracking solution based on a dynamic Bayesian network that simultaneously infers the number of people in a scene, their body locations, their head locations, and their head pose. For efficient inference in the resulting variable-dimensional state-space, they propose a Reversible-Jump Markov Chain Monte Carlo (RJMCMC) sampling scheme and a novel global observation model, which determines the number of people in the scene and their locations. Finally, in order to determine if a person is looking at the advertisement or not, they suggest Gaussian Mixture Model (GMM)-based and Hidden Markov Model (HMM)-based VFOA-W models, which use head pose and location information.

In multi-agent multi-user environments there is a need of clarifying who is talking to whom. Vertegaal et al [64] present an experiment aimed at evaluating whether gaze directional cues of users could be used for informing

an agent that is it being addressed or expected to speak. Using an eye tracker, they measured a subject's gaze at the faces of conversational partners during four-person conversations. Results indicated that when someone is listening or speaking to individuals, there is a high probability that the person looked at is the person listened or spoken to. Hence they conclude that gaze is an excellent predictor of conversational attention in multiparty conversations, but its predictive power may depend on the individual user and the visual design of the conversational system. To support that they implemented FRED, a multi-agent conversational system that uses eye input to gauge which agent the user is listening or speaking to.

In [65] the authors present a way to extract user attention and head gestures utilizing the shape and texture parameters from a fitted Active Appearance Model (AAM). The main advantage of using an AAM is the holistic representation of the face. They focus on improving human-robot interaction and therefore apply an attention and head gesture estimation which uses the AAM shape parameters to estimate the users head pose. For the measure of attention the distribution of the head pose over time is used, while to allow a more natural dialog the head pose is also very efficiently interpreted as head nodding or shaking by the use of adaptive statistical moments. For head pose estimation an own dataset is used which consists of mixed facial image sequences of male and female people who rotate their heads around. Each image of this dataset is labeled by the current head pose, which is determined by the so-called Flock of Birds. The Flock of Birds is a two-parted system which determines the head pose using magnetic fields. One part is fixed and must be positioned near the camera; the other part must be mounted on the top of the head. Finally, in order to facilitate users that are restricted in their head movements (e.g., demented) and mostly look around only by using eye movements, they use a simple eye tracker, which operates with an ordinary webcam.

The Reading Assistant presented in [66] is a computer-based remediation tool that uses visually controlled auditory prompting to help the user with recognition and pronunciation of words. This work is a straightforward extension of the GWGazer, a system that predicts user alertness and attentiveness using eye tracking. The main objective of the Reading Assistant is to track the reader's eye movements and, using principles derived from reading research, aid the reader by pronouncing words that appear to be hard to recognize. The system takes advantage of (i) the ability of unobtrusive eye tracking systems to follow eye motion and (ii) the ability of text-to-speech software to help children learn to read. As students read text displayed on a computer screen, a video camera, mounted below the screen, monitors the students' eye motions. The eye tracking system analyzes the infrared video image of the eye and computes the coordinates of the gaze-point (the point at which the eye is looking) on the screen and sends them to the GWGazer application, which keeps track of the user's scan of the displayed text in real time. Visual and auditory cues are produced that depend on where the student is looking and whether changes in scan pattern indicate difficulties in identifying a word.

In [67] the authors present an empathic software agent (ESA) that aims to overcome the challenges of online learning and increase students' motivation by stimulating their interest. In online learning the students can lose motivation and concentration easily, especially in a virtual education environment that is not tailored to their needs and physical contact with human teachers is limited. For that purpose, ESA detects the attention information from real-time eye tracking data from each learner and modifies instructional strategies based on the different learning patterns of each learner. During the learning process, the system records the user's eye gaze and pupil dilation and based on these measures, it infers the focus of attention and motivational status of the learner and responds accordingly with affective (display of emotion) and instructional behaviors. Besides that, ESA uses eye

gaze information for non-real-time collection of eye-tracking statistics, such as how learners look at images and how long they spend looking at different objects.

Similarly to the ESA system, the AdELE [58] framework supports adaptive e-learning utilizing not only eye tracking but also content tracking technology. This framework ensures not only adaptivity to the users' preferences, knowledge level and the real-time tracking of their behavior, but also ensures the relevance, accuracy and reliability of the knowledge provided. AdELE delivers interfaces adapted to users' needs and provides content adaptation according to the gained behavioral information of the user. For that to be achieved, they combine fine-grained real-time eye tracking with: (i) synchronous content tracking, (ii) a user profiler, (iii) an adaptive multimedia learning environment, and (iv) a dynamic background library. In particular, adaptivity through eye-tracking is accomplished by observing users' learning behavior in real time and monitoring characteristics such as objects and areas of focus, time spent on objects, frequency of visits, and sequences in which content is consumed.

In [57] the authors focus on remote instructional collaborative tasks where participants are assigned to either a "helper" role or a "worker" role. The helper offers the knowledge to guide the operations, while the worker provides the physical labor. Such a relationship is similar to a teacher instructing a student in a physics experiment or an engineer guiding a technician servicing a vehicle. Trying to overcome the challenges of this type of collaboration they introduce an "intelligent" video system that provides the right camera feed at the right time during a collaborative physical task, so that the helper can have the most beneficial view of the worker's environment at each point in time. To that end, instead of tracking focus of attention (FOA) via head position, they propose that the identification is established based on the user's intention, the task properties, people's actions in the workspace, and conversational

content. They employed a conditional Markov model to classify FOA from the intention decoded from dialogue content and workers' actions.

Merten and Conati in [68] are concerned about the added value that eye-tracking technology can offer towards modeling and adapting Intelligent Learning Environments. In their work, they use eye-tracking data for assessing on-line the user meta-cognitive behavior during the interaction with a system. They mainly focus on the capability to effectively learn from free exploration and to self-explain instructional material as both these meta-cognitive skills have shown to improve the quality of student learning. Based on the findings of a formal evaluation where they compared three different student models with various predictors (i.e., a simple navigation-based predictor, a navigation- and timing- based predictor, and a combination of navigation- and timing- based predictor with eye-tracking), they concluded that more sophisticated predictors improve the recognition of user meta-cognitive skills and consequently student learning.

Attention Monitoring in Smart Educational Environments

Attention aware systems have much to contribute to educational research and practice. These systems can influence the delivery of instructional materials, the acquisition of such materials from presentations (as a function of focused attention), the evaluation of student performance, and the assessment of learning methodologies (e.g., traditional teaching, active learning techniques, etc.) [42]. However, existing approaches [58], [66]–[69] concentrate mainly on computer-driven educational activities. This highlights the importance of LECTOR, which is able to monitor student behaviors in a real classroom setting and suggesting improvements for the learning process.

Resetting Student Attention

Literature suggests several strategies to regain student attention and increase the level of engagement in learning activities; among them, Active Learning

was acknowledged as the most effective instructional method in terms of resetting the students' concentration and decreasing attention lapses during lectures. Unlike the traditional way of lecturing where students passively receive information from the instructor, active learning introduces active student participation in the main course. These activities however should be appropriately designed to promote thoughtful and meaningful engagement on the part of the student and deliver important learning outcomes. Such an example of active learning could be a case where the lecturer periodically pauses his talk and asks students to clarify their notes with a partner.

Several studies not only highlight the advantages of active learning, but also support that praise, encouragement, reprimands and multimodal cues positively affect the students' motivation.

Michael Prince in [70] examines the effectiveness of active learning. To do so, he initially clarifies what active learning is and how it differs from traditional engineering education, which is also considered "active" through homework assignments and laboratories, then defines different forms of active learning (i.e., Active Learning, Collaborative Learning, Cooperative Learning and Problem-based Learning) and finally summarizes the most relevant surveys in this field. His findings validate that: (i) the introduction of various activities during lectures (i.e., Active Learning) can significantly improve recall of information and increase student engagement, (ii) Collaborative Learning enhances academic achievement, student attitudes and student retention, (iii) Cooperative Learning is more effective in promoting a range of positive learning outcomes compared to Competitive Learning, and finally (iv) Problem-based Learning, the most difficult method to be analyzed due to the variety of practices and the lack of a dominant core element, positively influences student attitudes and study habits (e.g., students will retain information longer and perhaps develop enhanced critical thinking and problem-solving skills), but is unlikely to improve their test scores.

In [71] the authors examine four active learning strategies in a within-subjects design and explore the effect of these "interventions" on concentration on a sample of senior-level students. These evaluated interventions are based on basic questioning and peer sharing techniques, including: (i) Student-Generated Questions, (ii) Guided Reciprocal Peer Questioning, (iii) Think-Pair-Share, and (iv) Truth Statements. The findings revealed that only two interventions had an encouraging influence on students' concentration ratings, including Guided Reciprocal Peer Questioning and Student-Generated Questions. As the use of questions can encourage students to describe what they believe and how they came to believe it, questioning interventions appear to help reset student engagement by actively involving the student in substantive exchange compared to a mere sharing of what they think. On the contrary, both the Truth Statements and Think-Pair-Share interventions actually evidenced a lower average end-of-activity concentration rating compared to the pre-activity interval.

The advantages of active learning techniques are also demonstrated in [72]. The authors explain that the employment of these techniques is vital because of their powerful impact upon students' learning and they propose several solutions for incorporating them in the classroom, such as: encouraging discussion, allowing the students to consolidate their notes, etc. Another interesting observation substantiates that certain alternatives to the lecture format could further increase student level of engagement: (i) the feedback lecture, which consists of two mini-lectures separated by a small-group study session built around a study guide, and (ii) the guided lecture, in which students listen to a 20- to 30-minute presentation without taking notes, followed by their writing for five minutes what they remember and spending the remainder of the class period in small groups clarifying and elaborating the material. Finally, besides enumerating the benefits of active learning, the authors present the difficulties associated with that instructional approach, including limited class time, increased preparation time, difficulty of using

active learning in large classes and the lack of materials, equipment, and resources.

Bunce et al [54] perform an experiment in order to investigate how often during a lecture segment students report a lapse in their attention and how student-centered pedagogical approaches increase attention and motivation. College students were equipped with clickers and they were instructed to press them when their minds wandered from the material being presented in class. Furthermore, to evaluate the effectiveness of alternative teaching techniques, the instructors used different pedagogical approaches interchangeably (lecture, demonstration, clicker questions, working in student groups or pairs, etc.). This study not only concluded that students do not pay attention continuously during a 50-min lecture and attention lapses may occur at any time, but more importantly demonstrated that the positive effect of student-centered pedagogical approaches decreases student inattention and adds a carryover effect to a subsequent lecture segment. This supports the idea that changing pedagogical activities within a class period cannot only be seen as a way to present concepts in an alternate format, but may also help engage students in subsequent lecture teaching formats.

A very similar set of results was evidenced by Young et al [48] in their exploratory study about students' attention during different lecture formats. They adopted a vigilance measurement technique from ergonomics and focused on four types of college lectures: (i) traditional "chalk-and-talk" lecture, (ii) lecture introducing guest lecturer, (iii) lecture allowing "buzz group" discussion and (iv) lecture presenting case studies with video media. Their findings validate that: (i) any variation in presentation or media can help maintain attention and facilitate deeper learning, and (ii) Buzz groups and other interactive sessions do have clear advantages compared to standard 'chalk-and-talk' lecture format which appear to cause vigilance decrement.

The work in [73] compares the techniques of encouragement and reprimands as a mean to reduce off-task behavior in the classroom. Two experiments were

performed involving children with academic and/or behavioral problems in a remedial summer program. In both experiments reprimands resulted in lower rates of off-task behavior and higher academic productivity than encouragement. In contrast to reprimands, encouragement appeared to have no systematic effect, improving some students' behavior, causing others' behavior to deteriorate, and having no obvious effect on other children.

Hitz and Driscoll [74], on the other hand, highlight the advantages of encouraging students during learning activities. According to their study most students thrive in encouraging environments where they receive specific feedback and have the opportunity to evaluate their own behavior and work. Comparing encouragement and praise, they emphasize that some praise statements can potentially lower students' confidence in themselves and lead them to focus their attention on extrinsic rewards. Thus, by constantly encouraging students, teachers can create an environment in which students do not fear continuous evaluation, can make mistakes and learn from them, and do not have to strive to meet someone else's standard of excellence.

Finally, an interesting experiment is presented in [75]; it examines whether spatially non-predictive bimodal cues are more effective than unimodal cues in capturing spatial attention away from a perceptually demanding central task. The results highlight a qualitative difference between the nature of the exogenous orientation of visuospatial attention effects triggered by unimodal and bimodal cues. In more detail, as evidenced by a comparable magnitude of spatial cuing effects in the no perceptual load (i.e., baseline) and high perceptual load conditions, the unimodal cues failed to capture participants' visuospatial attention exogenously under conditions of high load, whereas bimodal cues (which were completely spatially non predictive, i.e., they were task irrelevant) successfully captured participants' visuospatial attention regardless of any concurrent increase of their visual perceptual load.

Discussion

The study of attention monitoring technologies revealed the challenges, the proposed solutions, the appropriate techniques and the results of each approach when used in real settings. Besides exposing the technological advances, this survey also provided great insights on how such systems can monitor a smart classroom environment, identify the student needs and suggest improvements for the learning process. For example, eye-tracking can monitor student's visual fixations during exercise solving and detect potential weaknesses, while head tracking is appropriate for estimating whether the student pays attention to the teacher, concentrates on the class board, or dawdles with another classmate. Nevertheless, such systems cannot effectively address the challenges of a real classroom. A non-exhaustive set of the additional constraints that should be taken into account include:

- Students' attention can be captured by multiple targets (e.g., class board, book, teacher, classmates, personal computer, etc.)
- Learning behaviors differ among students (e.g., doodling while concentrating vs. doodling while being bored)
- Contextual information might cause substantial deviation of attention indicators (e.g., Art vs. Literature course)
- Groups dynamics and collective behaviors greatly affect the decision-making process (e.g., the whole classroom is laughing at funny comment made by the teacher).

Furthermore, the conducted analysis allowed to identify several cues might signify inattention and fall under the following categories: Focus, Speech, Location, Posture and Feelings. Additionally, it was revealed that using multiple cues (e.g., gaze, head pose, body posture, sound, etc.) can increase the accuracy of predicting the student's focus of attention. This information highlights the necessity of supporting the combination of multiple physiological attributes when modeling a behavior (e.g., head to the left and increased sound indicates talking to classmate).

Another contribution of this survey was the specification of the contextual information needed to support an attention aware Intelligent Classroom that intervenes to re-engage tired or unmotivated students. In more detail, such an environment must be aware of the current **course**, the ongoing **activity**, and the characteristics of its **students**. Furthermore, it was revealed that it is of outmost importance to combine diverse contextual information to accurately determine the level of attention of each individual or the entire classroom and reduce false positives.

In addition, the study of background theory regarding attention and education showed that being able to observe discrete inattention types is really important, since it facilitates the process of selecting appropriate interventions. As an example, the intervention that will be applied to a student who is chatting will be different from the intervention that will be applied to a student who is fatigued. This leads to the specification of several student behaviors that should trigger interventions (i.e., cheat, chat, disturb, fatigue, boredom and out of seat).

Finally, adjusting the learning process is a key aspect in the intelligent classroom; thus, the use of LECTOR in this context should be based on widely accepted techniques on resetting student attention. This extensive literature survey confirmed that introducing engaging activities into the main lecture and changing pedagogies within a class period has remarkable effects on students' concentration. However, teachers are not always willing to adopt and apply active learning techniques in their daily teaching routine, since the available class time is limited (e.g., 50 minutes) and the preparation of the required material is time consuming. In an intelligent classroom environment though, where the computational resources surplus, LECTOR can automatically compile and initiate the appropriate activities based on the context of use, the students' needs and the available time.

Chapter 3
Related Work

An AmI environment should be able to recognize the users in it, understand their needs, identify their behavior, and act and react in their interest. Fundamentally, this process entails two basic steps (i) identifying a user need and, (ii) acting accordingly; this simplification reveals the similarities with the trigger-action model [2]–[4], which in the recent years has been in the spotlight as a form of programming AmI environments, using simple "if trigger, then action" rules. The goal of this thesis is to offer a framework that advances the way through which both developers and educators connect student behaviors that signify inattention (triggers) to specific context-aware interventions (actions).

Chapter 3 provides a review of research studies and commercial tools that have examined trigger-action programming or related interfaces over the last 15 years.

Trigger – Action Programming

The emergence of the Ambient Intelligence (AmI) paradigm and the proliferation of physical Internet of Things (IoT) devices have raised the need for appropriate tools that enable users to connect and manage the abundant devices and services. According to [3], one of the most straightforward approach towards that direction is the trigger-action paradigm, which enables users to configure the behavior of a system by specifying triggers (e.g., "if the

user has awakened") and their consequent actions (e.g., "turn on the coffee machine"). Trigger-action programming (TAP), has been the focus of many research approaches and several commercial devices.

The Context-aware Application Prototyper (iCAP) [76] enables the description of situations and their association with specific actions. Its objective is to support inexperienced end-users in building interesting context-aware applications for their instrumented environment without writing any code. To do so, iCAP features a visual, rule-based system that permits prototyping of 3 common types of context-aware behaviors: simple if-then rules, relationship based actions and environment personalization. In order to create the building blocks of a rule, the user has to define the people and artifacts involved in it and add them to a repository. Next, an output medium (e.g., cell phone) has to be defined by specifying several characteristics, such as its category (sound) and type (binary: on/off). As soon as this information is available, the user can start building the rule by simply dragging the respective components on a visual area (Figure 1).

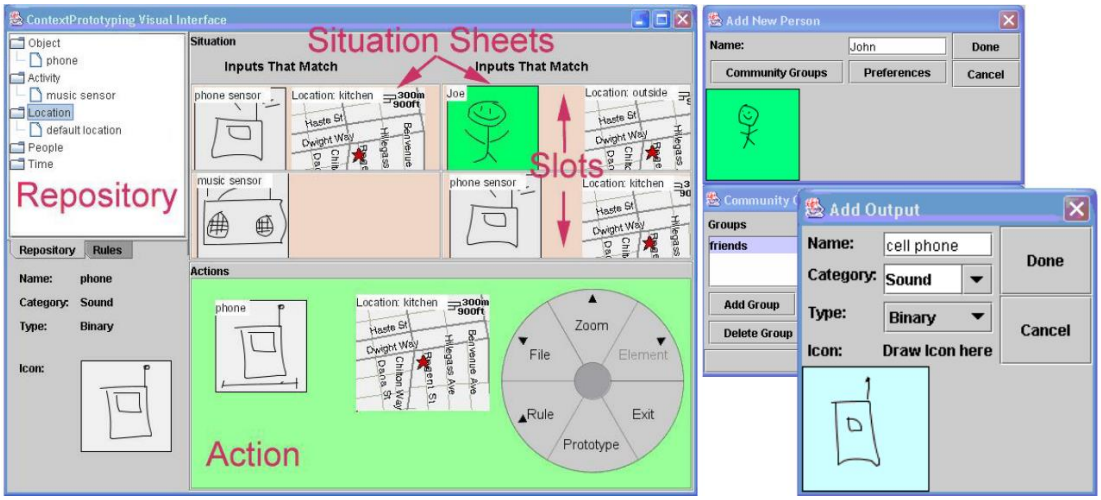

Figure 1: The user interface of iCAP [76]

In [77] the authors use the metaphor of recipes for food preparation, where users are able to produce a large and varied number of compositions from a small set of ingredients. In their approach, a "digital recipe" follows a simple yet effective programming procedure, where variables (ingredients) are

initialized before the control statements (preparation). In order to support their concept, the authors developed a prototype of a universal remote control application that allows users to discover, use, and create digital home recipes. This application, named OSCAR (Figure 2), is described in [22]; it permits monitoring and manipulation of connections between devices, and enables users to create rules that apply to frequent activities. OSCAR runs on a touchscreen-based tablet and permits users to browse through devices and media sources available on the home network. Upon selection, the user can connect the device or media source to other items, obtain its user interface, or use it as part of a reusable configuration.

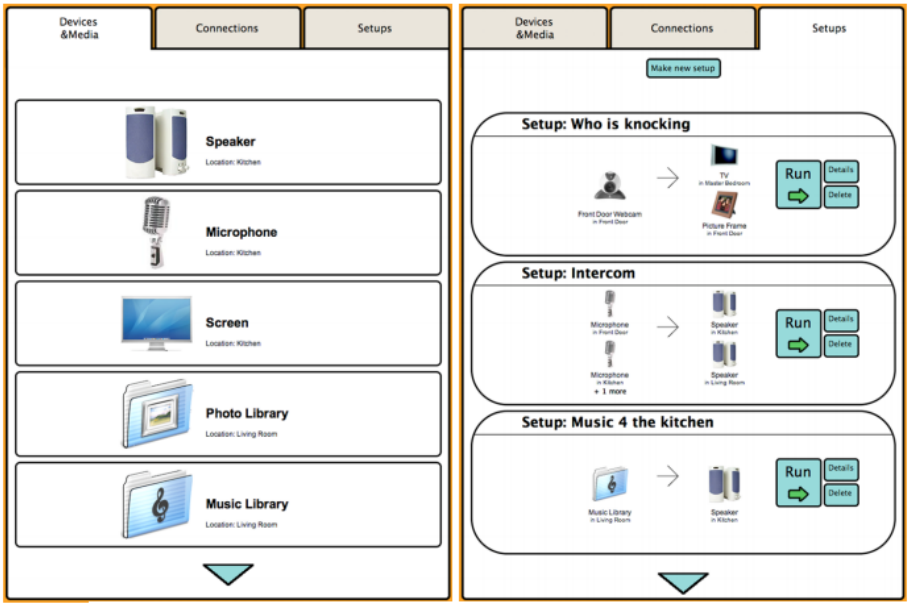

Figure 2: OSCAR, the application that supports flexible and generic control of devices and services in home media networks [22]

The work in [78] presents an early prototype of a framework that supports the creation of mashup editors for Web-enabled smart things. It enables the composition of services of different smart things as well as virtual web-services. An extended version of the Ruote open-source workflow engine is available to users for creating their mashups in a very easy manner. The framework communicates with the smart things over their RESTful API and is responsible for executing the work-flows created by end-users.

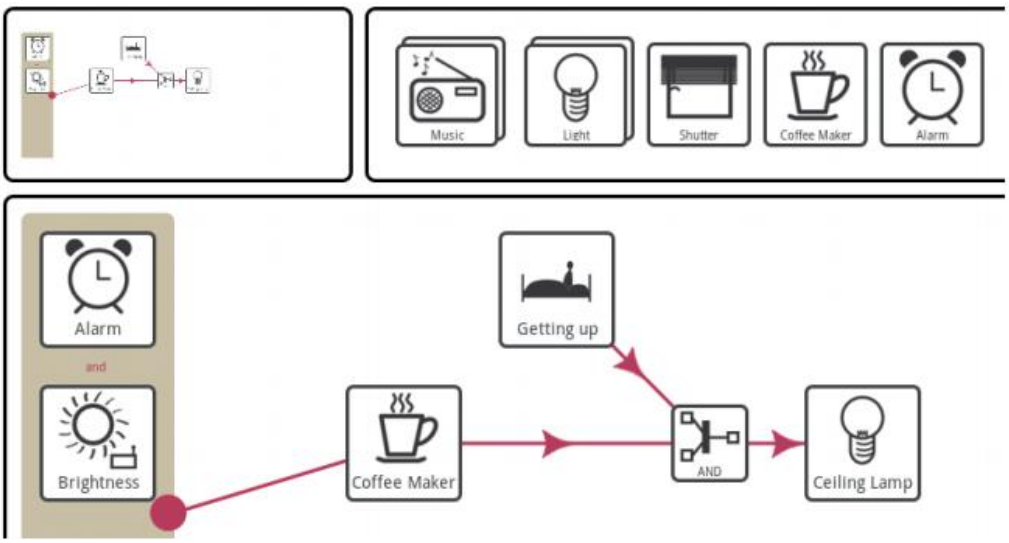

Figure 3: The user-interface of homeBLOX [23]

The authors in [23] present homeBLOX (Figure 3), a system that enables the creation of complex home automation scenarios with heterogeneous devices. In contrast to other related work, its configuration is process-driven rather than rule-based. Each automation task is represented as a sequence of events and actions connected to physical and virtual devices. These sequences are translated into BPEL code for deployment on a process engine. A table application allows users to create sequences by simply dragging the available devices on the main canvas and drawing connections between them.

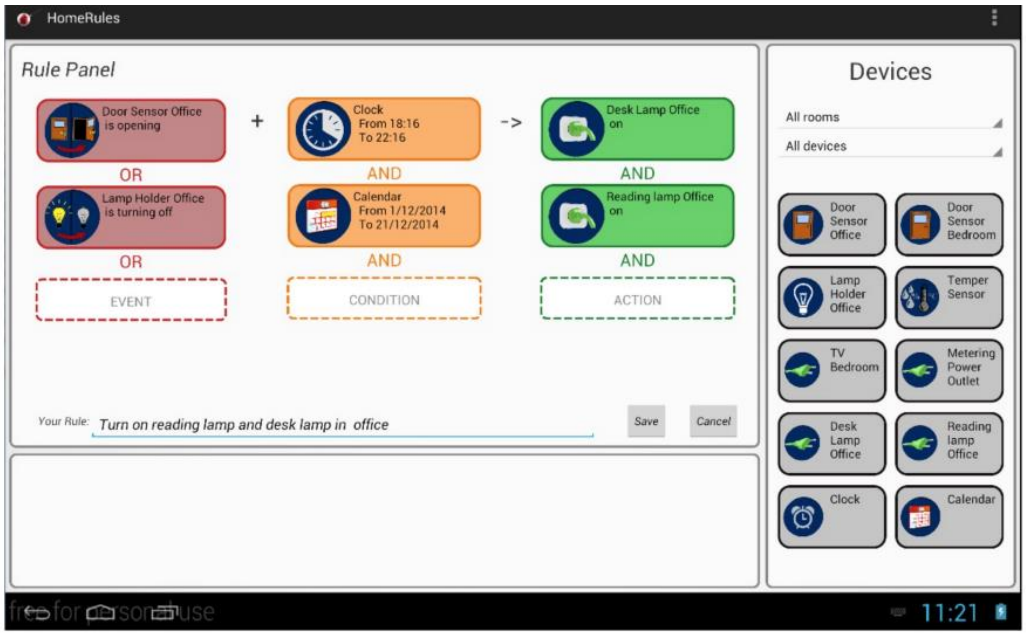

Figure 4: Snapshot of the HomeRules' User Interface

HomeRules [24] is a prototype interface for a tangible mobile application (Figure 4) that empowers users (with no technical skills) to “program” their smart home. The authors built this prototype following a set of guidelines that they came up with drawing from their own previous experiences and some well-known commercial products such as IFTTT and WigWag.

Massieu [79] in his thesis presented GALLAG Strip, an approach employing the programming-by-demonstration technique for programming sensor-based context aware applications. It permits users (with no computer programming skills) to program simple “if-then” rule-based applications by physically demonstrating their envisioned interactions within a space using the same interface that they will later use to interact with the system (i.e., GALLAG-compatible sensors and devices).

Locale [80] and Tasker [81] are Android applications (Figure 5) which allows users to create situations specifying conditions under which their phone's settings should change. The conditions are primarily related to the position and orientation of the phone, the date and time, the battery levels, the incoming calls, the location and application. For example, an "At Work" situation notices that the user’s Location condition is "77 Massachusetts Ave.," and changes the phone’s Volume setting to vibrate. In the case of Locale, developers can employ existing plug-ins and integrate directly with it through the Developer API.

a. b.

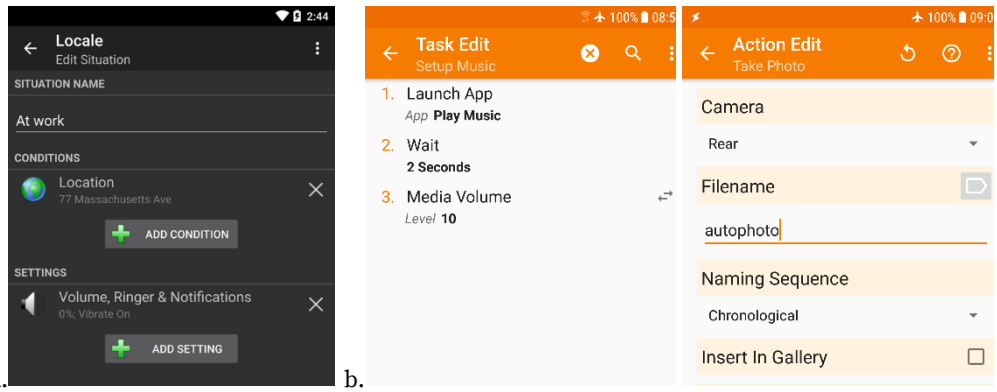

Figure 5: (a)Locale's user interface for creating a situation [80]. (b) Tasker’s UI for creating tasks and actions

Apiant [82] is a Cloud Based API Integration Platform targeting non-technical users as well as integration professionals that allows them to connect their applications in order to automate their tasks. Through an automation editor, simple users can connect their apps, and when one or more criteria is met (triggers), the automation launches sophisticated routines (actions). An assembly editor permits customization of the automation without requiring coding.

Atooma is an application that uses the 'if-then' construct for defining rules; Atooma's flagship is the Resonance AI software platform [83]. Resonance AI offers a set of AIs (not mere APIs) to make things learn routines (home, work, etc.), and use them to predict what the user will need. Through the APIs developers can listen to context changes and use them to trigger actions. The latter is achieved either by a Low Level API or Resonance Distributed Rule Engine, allows developers to activate and execute IF-DO rules when users enter in (or exit from) a specific context.

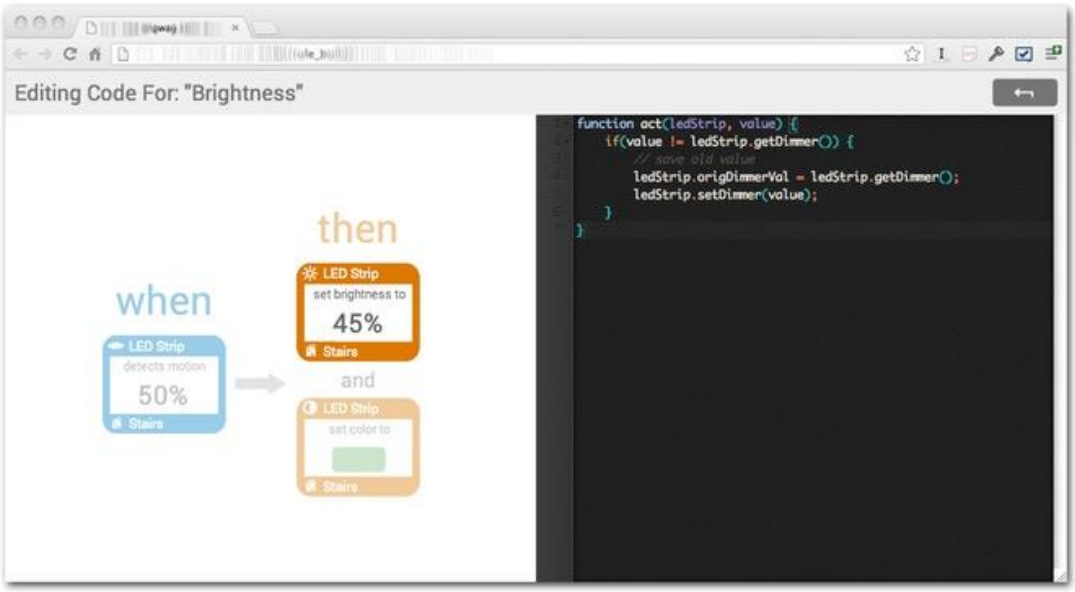

Figure 6: WigWag's rule editor for developers (image taken from: www.kickstarter.com)

WigWag [84] is an open source system that aims to bring intelligence into commercial spaces. Through a mobile application it permits users to control instantly any internet-connected sensor and device inside a house or an office.

It delivers a platform that enables the graphical definition of rules in the form: "When" [this] happens "Then" do [that]. Apart from supporting inexperienced users with no computer skills, WigWag permits developers to go deeper and write Javascript code to modify the rules (Figure 6).

Zipato [25] is a rule-based system for home control and automation. Zipato requires its own gateway, called Zipabox, to which the many devices of the system are connected. The devices could come from other manufacturers, but must adhere to specific standards. Its Rule Creator is an easy and intuitive web-based graphical tool that allows users to easily create and modify their automation rules. A library of micro blocks (control functions) is employed to provide the flexibility to develop simple and complex control sequences (Figure 7).

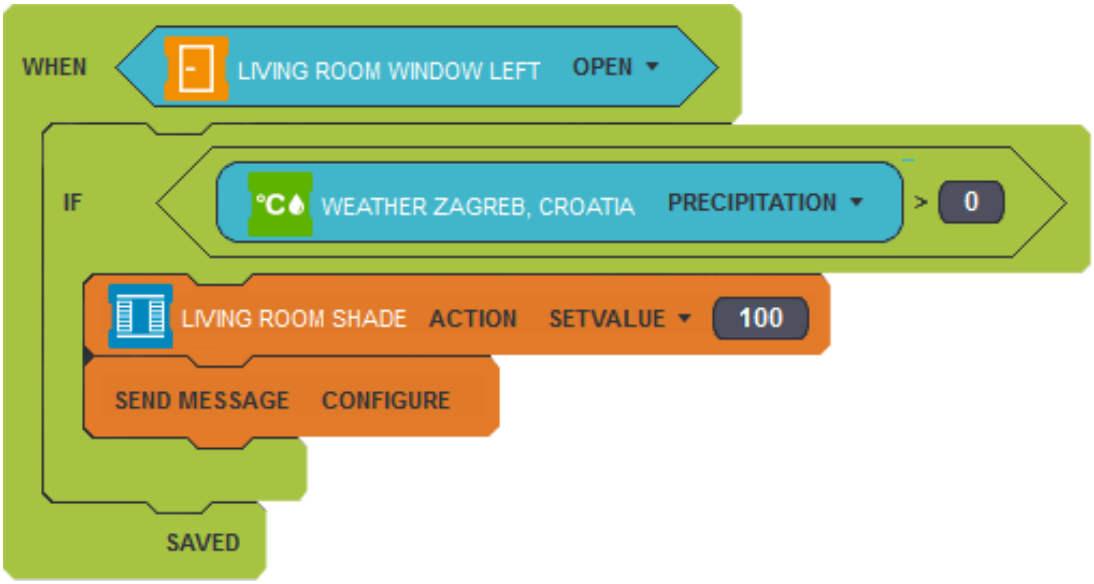

Figure 7: Zipatos' Rule Creator [25]

TWINE [26] is a wireless sensor block tightly integrated with a cloud-based service (Figure 8). Wi-Fi, temperature and orientation sensors are integrated on a durable block made of rubber, while an expansion connector enables the employment of other sensors. TWINE focuses on event detection using If-THEN rules in smart environments. It offers a web application through which the users can set up and monitor their “Twines” from any browser. “Twines” are rules that trigger messages; a palette of available conditions and actions is

available for composing the rules which appear as normal sentences: WHEN moisture sensor gets wet THEN text "The basement is flooding!"

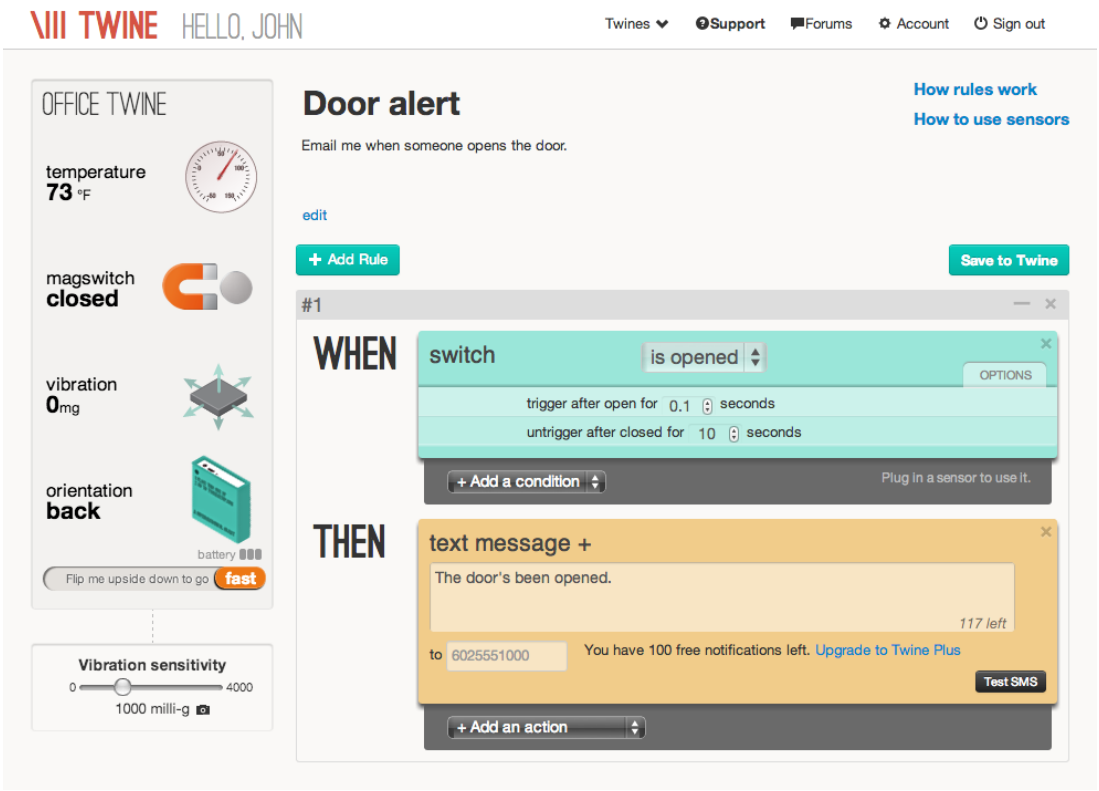

Figure 8: TWINE's User Interface [26]

IFTTT (if-this-then-that) [21] is the most popular TAP ecosystem that allows users to create programs (recipes) able to “act” appropriately when certain triggers occur. The programs created through IFTTT (Figure 9) are called recipes and can utilize many services (e.g., Gmail, Dropbox), social media sites (e.g., Facebook, Twitter), and physical devices (e.g., Alexa, Phillips Hue lightbulbs), which can be used either as triggers (e.g., if door is locked) or actions (e.g., turn of the lights). Its wide acceptance from the users that stems from the simple wizard-style interface. Furthermore, its increasing support for wearables, smartphone sensors, and other commercial devices, contributed to IFTTT becoming tightly coupled with ubiquitous computing. Lately, a beta platform has been released for developers that enables them to input Javascript code that changes how a recipe runs.

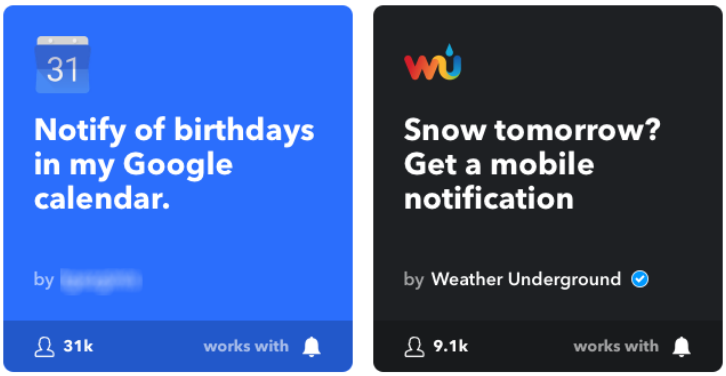

Figure 9: IFTTT Applets

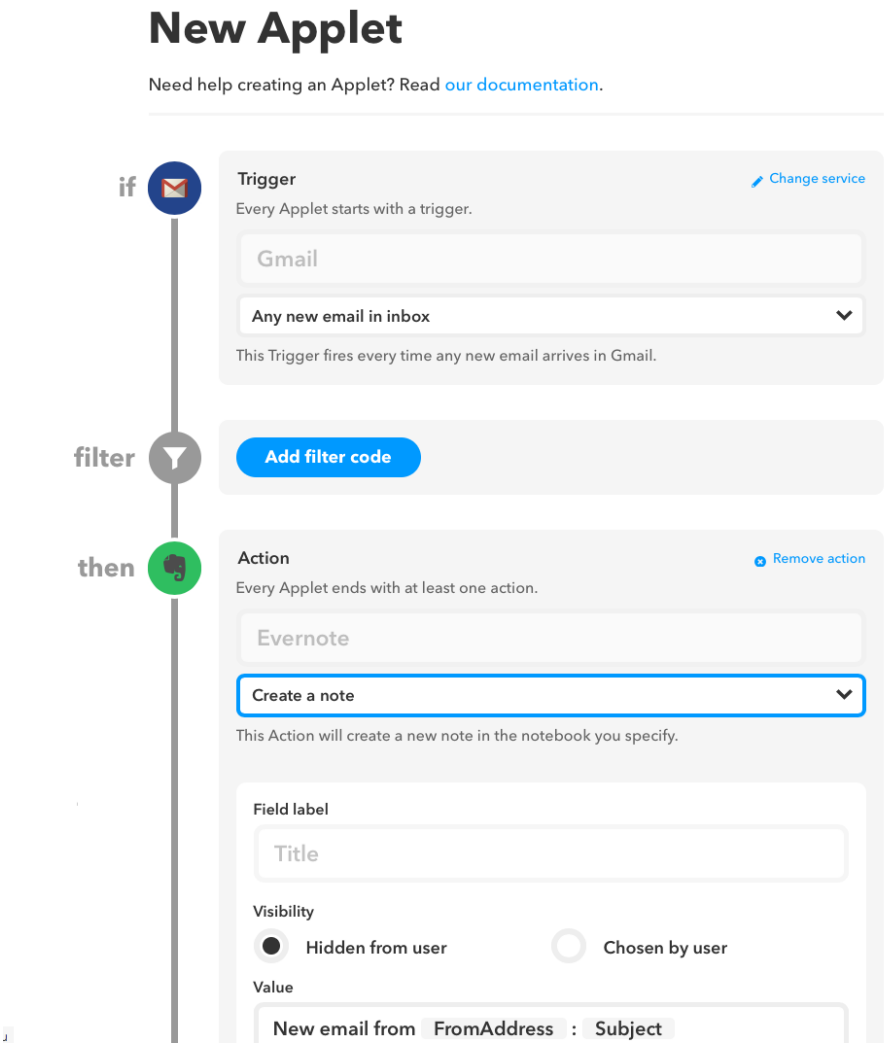

Figure 10: Snapshot of the IFTTT platform for developers

Similarly to IFTTT, Zapier [27] (Figure 11) allows users to combine triggers and actions to define various Zaps (tasks). In contrast to IFTTT that limits each task to a single trigger and action in order to make things easier for non-technical end-users, Zapier permits multiple actions for a single trigger and introduces

the notion of filters that provide additional conditional (and/or) control over triggers. According to [85], there are some other characteristics that differentiate the two systems. In more detail, IFTTT integrates a social component for sharing recipes, while it offers a mobile application that allows mobile services to be added as services. In the case of Zapier, developers are able to define their own services.

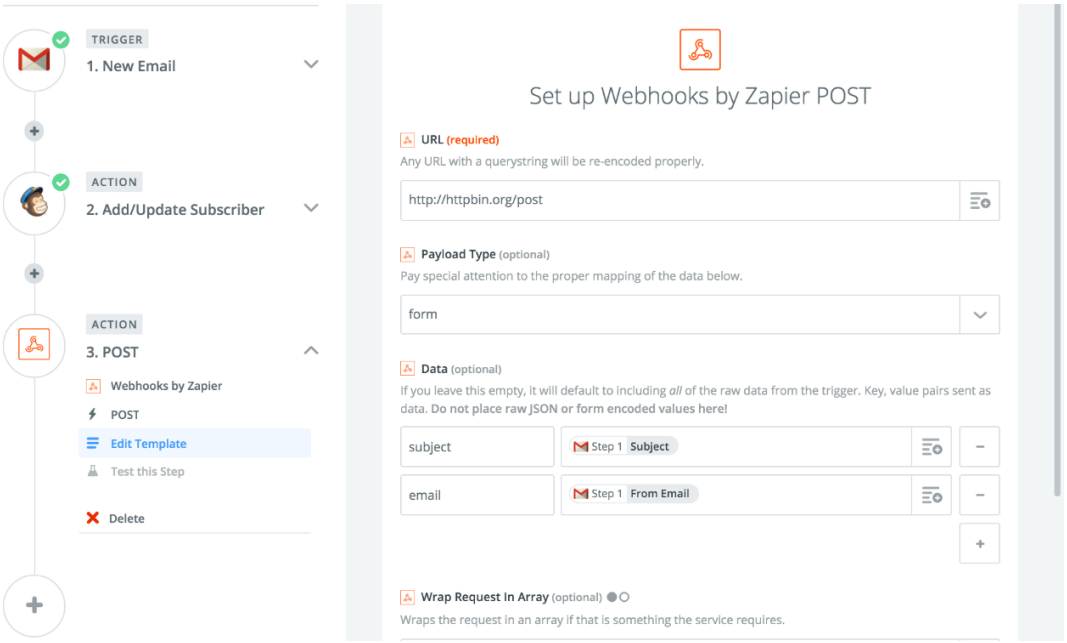

Figure 11: Snapshot of Zapier [27]

Discussion

The purpose of the above analysis was to examine current state-of-the-art technology that concentrates on trigger-action programming. Indeed, the analysis revealed that there are quite a few noteworthy approaches; the majority of them is based on rule-based conditions to enable non-technical users and developers describe their recipes / programs naturally. Despite the limitations stemming from using simple "if-then" rules, which seem to hinder the expressivity of the programs that can be created [4], the author in [86] has shown that general-purpose programming or scripting languages are not necessary or desirable. On the contrary, according to [77], "a task specific language with appropriate tool support provides an ideal environment for users to create their own applications".

Some of these systems [21], [85] are scalable enough and permit developers to integrate custom devices and services, while they also provide access to the code behind the generated rules, so as to facilitate their tweaking. However, others are bound to specific hardware and software [25], [26], [80], [81], [83] making their employment in different contexts impossible.

Interestingly, almost all of the examined systems evolve around device- or software- initiated triggers. In more detail, they require users to define, based on the status of devices or services (e.g., if temperature reaches 26°C, if incoming email from family member), the conditions under which an action is performed. However, since the main target group of trigger-action programming are unexperienced users, the rule logic should be de-coupled from the artifacts and be human-oriented instead. For example, the condition "if motion is detected in the hallway" becomes clearer to the simple user when expressed otherwise: "if I (or inhabitant) pass through the hallway". Towards that direction, it would be beneficial to support personal user profiles and grouping of users with the same needs; however, the latter is supported only in few of the studied systems [23], [76].

Finally, none of these approaches incorporates mechanisms for evaluating the performed actions. The systems are unaware whether the realized outcome was actually what the user intended when she was creating the rule. In addition, they do not take into consideration cancellations, which might indicate dissatisfaction on behalf of the user.

Table 1: Comparison of tools that permit trigger-action programming (adapted from [87])

Tool	1	2	3	4	5	6	7	8	9	10	11
iCap	✔	✖	~	✔	~	~	~	✖	~	✖	✔
OSCAR	✔	✖	~	~	✖	✖	~	✖	✖	✖	✔
HomeBLOX	✔	✖	✔	✖	✖	✖	✔	✖	✖	✖	✔
Gallag Strip	✔	✖	~	~	✔	✖	✖	✔	✖	✖	✖
Locale	✔	~	~	✖	~	✔	✔	✖	✖	✖	✖
Tasker	✔	✖	~	✖	~	✔	✔	✔	✖	✖	✖
Atooma	✔	✔	~	✖	~	✔	✔	~	✖	✖	✖
WigWag	✔	✔	✔	✖	✔	✔	✔	✔	✖	✖	✖
Apiant	✔	✔	~	✖	✔	✔	✔	✔	✖	✖	✖
Zipato	✔	✔	✔	✖	✔	✔	✔	✔	~	✖	✖
TWINE	✔	✖	✔	✖	✔	✖	✔	✔	✖	✖	✖
IFTTT	✔	✔	~	✖	~	✔	✔	✖	~	✖	✖
Zapier	✔	✔	✔	✖	✔	✔	✔	✖	~	✖	✖
LECTOR	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

✔ the characteristic is totally satisfied
~ the characteristic is partially satisfied
✖ the characteristic is not satisfied

The following list describes the characteristics based on which the examined systems were summarized. The results are presented in Table 1.

1. The tool offers a user interface through which non-technical users are able to create rules / recipes
2. The tool permits developers to tweak the rules / recipes by granting them access to code

3. The tool features a user friendly and intuitive user interface
4. The created rules / recipes are user-oriented
5. The tool permits the integration of custom or third party devices and services as triggers
6. The tool permits developers to create new software artifacts for the system
7. The tool enables users to add, edit or remove rules easily
8. The tool permits the definition of multiple conditions and actions per each rule
9. The tool permits the introduction of custom interventions / actions
10. The tool has the ability to evaluate the actions / interventions
11. The tool is free

This thesis aims to equip the Intelligent Classroom with a mechanism that takes advantage of its ambient facilities to identify when students require assistance and intervenes to support them. Towards that direction, it offers a framework -based on the trigger-action model- that aims to address the gaps identified in the current state-of-the-art. More specifically, it relies on "if-then" rules -created either by developers or educators- to dictate the behavior of the AmI environment. In order to ensure scalability and better rule management, a three (3) step process for connecting a behavior with an intervention is introduced. In more detail, the first step is to define a **behavior**, next the conditions under which the behavior becomes a **trigger** have to be described, and the last step is to create a connection between a trigger and an **intervention**. This decomposition permits a behavior to be associated with many triggers and a trigger with many interventions, depending on the context of use. Furthermore, the rule structure supports the creation of **user-oriented** behavior-intervention scenarios in contrast to the artifact-oriented recipes that are currently supported by the majority of IFTTT-style tools.

Additionally, appropriate infrastructure enables the **evaluation of system decisions**, by allowing users to (i) invalidate identified behaviors, and (ii) override system suggestions in case they are not appropriate.

Finally, a sophisticated user-friendly authoring tool is introduced for supporting both developers and educators in creating the rules; the same tool enables developers to integrate the building blocks (i.e., actors, intervention hosts, physical context, virtual context, interventions) that are required for programming the Intelligent Classroom.

Chapter 4
Requirements

Chapter 4 initially introduces several scenarios that motivated this work and afterwards outlines LECTOR's functional and non-functional requirements.

Motivating Scenarios

Scenario building is a widely-used requirements elicitation method [88] that can systematically contribute to the process of developing requirements. Scenarios are characterizations of users and their tasks in a specified context, which offer concrete representations of a user working with a computer system in order to achieve a particular goal. Their primary objective in the early phases of a development cycle is to generate end user requirements and usability aims.

The following sections present a collection of envisioned scenarios where identifying inattentive behaviors and intervening to re-engage the learners' to the educational process can benefit learning.

Monitoring the attention levels of an entire classroom

On Monday morning the history teacher, Mr. James, enters the classroom and announces the topic of the day. These days they are studying the wars of Alexander the Great, while the "Battle of Gaugamela" is next in line to analyze. During the first fifteen minutes the students pay attention to the teacher that narrates the story, the tactics and the military strategies of the young king;

soon enough, the students start to lose interest in the historical details and demonstrate signs of inattentive behavior. In more detail, John is browsing through the pages of a different book, Mary and Helen are whispering to each other, Peter stares out the window and Mike struggles to keep his eyes open.

Figure 12: Re-engaging students by introducing active learning activities

When identifying that the entire classroom demonstrates signs of vigilance decrease, the system recommends that the lecture should be paused for a while and a mini quiz game should be started. The teacher finishes up his sentence and decides to accept this "intervention". After that positive response, a set of questions relevant to the current topic is displayed on the classroom board, while their difficulty depends on both the students' prior knowledge and the studied material so far. In order to increase students' motivation, the teacher reads the questions out loud, provides the necessary clarifications and encourages them to choose one out of the four possible answers through the application displayed on his/her personal computer. During use, the system identifies the topics with the lowest scores and notifies the teacher to explain them more thoroughly.

As soon as the intervention ends, Mr. James resumes the lecture. At this point, the students' attention is reset and they begin to pay attention to the historical facts. As a result, the mini quiz not only restored their interest but also resulted in deeper learning.

Monitoring the attention levels of an individual student

During the geography class Kate is distracted by a couple of students that stand outside the classroom window. Instantly, the system recognizes that her

attention is captured by external stimuli and decides to take immediate action to attract her interest back on the lecture. To do so, it displays pictures relevant to the current topic on her personal computer while a discreet nudge attracts her attention. Kate observes a picture displaying a dolphin with weird colors swimming in the waters of Amazon and wonders how it is possible for a dolphin to survive in a river; she patiently waits for the teacher to complete his narration to ask questions about that strange creature. That way, Kate becomes motivated and starts to pay attention to the presentation of America's larger rivers.

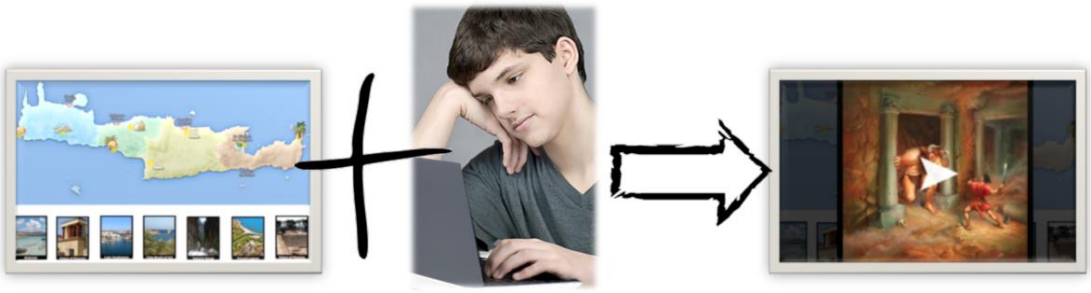

Figure 13: The educational application reacts to keep the students motivated

At the same time, Nick is drawing random pictures on his notebook and seems to not pay attention to the lecture; however, the system already knows that Nick concentrates easier when doodling and decides not to interpret that behavior as inattention. The ability to disambiguate student activities depends on information that only a human can provide. For that to be achieved, when the system identifies a behavior that can be misinterpreted it asks for the teacher's opinion; that input not only defines system's next action(s) for that particular case, but also re-evaluates the recognition algorithm for that student's learning style.

Monitoring students' fixations during exercise solving

Mrs. Brown, the Physics teacher, has scheduled to analyze Newton's first law today. When she finishes her analysis, she asks her students to work on a relevant multiple choice quiz on their course books and instructs the system to launch the appropriate application on each student's personal computer.

Mark answers correctly to the first question but seems to have trouble answering the second one. The system identifies that he reads that question over and over again and wavers among the first and the third answer. It immediately decides to intervene and asks him if he would like to proceed to the next one or to take a hint. Hoping that he will soon remember the correct answer, Mark decides to revisit that question later and accepts to move to the next question.

Meanwhile Christine, who has already answered six out of ten questions, seems to be distracted; instead of concentrating on the exercise, she stares at an irrelevant area on the computer screen for a significant amount of time. By examining these indicators, the system estimates that she is no longer interested in the task at hand and decides to reverse that situation. To do so, it uses a subtle nudge (sound or vibration) to draw her attention and at the same time displays an encouraging message that prompts her to continue with her work. Those messages are personalized to each student's progress, prior knowledge and learning abilities, thus a sentence such as: "Come on, there are four more questions to go!" or "You have answered far more difficult questions, why do you give up on these ones?" is used in Christine's case to tease her.

Monitoring attention levels during homework

On Monday afternoon, George is doing his homework in his room; five math assignments are due tomorrow and he has just started dealing with the first one. For each pending exercise the system displays on the desk surface a small reddish box with information about the assignment (e.g., topic, level of difficulty, estimated completion time, etc.). Since George is a strong student, he solves the first set of problems in less than twenty minutes and notifies the system accordingly, which triggers an automatic rearrangement of the assignments to highlight the remaining work. George is now ready to start working on the second assignment, however the system prompts him to take a small break to clear his mind and restore his energy levels.

After the short break, George successfully completes two more assignments and moves on to the fourth one. At that point, the system identifies he is distracted by external stimuli as he is constantly getting off his chair, looks out of the window and browses to social network websites while loud music is playing in the background. To reengage him to the homework activities, it displays encouraging messages such as: "If you start now, you will probably be finished by 20:00 to watch your favorite movie", or "Only two assignments to go, you will be done in less than an hour" as it recalls that George prefers to finish with his homework first and then spend some time relaxing.

Figure 14: Personalized messages to encourage homework activities

LECTOR Requirements

LECTOR framework aims to empower both developers and educators to realize the aforementioned indicative scenarios, by defining the **behaviors** that lead to context-aware **interventions**. This section presents the high-level functional and non-functional requirements that LECTOR satisfies, which have been collected through an extensive literature review and an iterative elicitation process based on multiple collection methods: brainstorming, focus groups, observation and scenario building.

Functional Requirements

FR-1: Support the modeling of the Physical Context

LECTOR should be able to facilitate the modeling of the **Physical Context**, which encapsulates information regarding physically observable phenomena (e.g., luminance, heart rate, sound levels, etc.). This aims to alleviate the diversity of input values coming from heterogeneous sources and create a shared data model that can be used by the rule creation mechanisms.

FR-2: Support the modeling of the Virtual Context

The **Virtual Context** refers to any static or dynamic information that is provided through software components (e.g., student profiles, course schedule). Since, such information may change depending on the classroom where the system is running and the characteristics of the classroom students, LECTOR should support the modeling of the Virtual Context so as to be used by the rule creation mechanisms.

FR-3: Enable the definition of user groups (Actors)

Actors are the users of the Intelligent Classroom whose behavior needs to be monitored in order to decide whether an intervention is required (i.e., Teachers, Students, Classroom). Generally, different types of actors have diverse characteristics which need to be taken into consideration when building the rules that guide LECTOR's decision-making mechanisms. To this end, LECTOR should permit the creation actor models.

FR-4: Enable the modeling of behaviors by combining multiple information from the Physical Context

The term **Behavior** is used to describe the way that a user or an artifact acts (e.g., a user talks, a device switches on). In some cases, multiple cues (from diverse sources) are required in order to identify a behavior; to this end, LECTOR should permit the definition of a behavior by combining multiple information from the Physical Context.

FR-5: Support the definition of triggers by combining multiple contextual information

The term **Trigger** is used to describe a high level **Behavior** that should initiate an intervention. A behavior can potentially become a trigger under specific context (e.g., the behavior TALKING might initiate the trigger CHATTING if a student is talking during a lecture). When defining the conditions under which a trigger is initiated, LECTOR should support the combination of multiple contextual information.

FR-6: Support the definition of user-oriented triggers

LECTOR framework should differentiate from other trigger-action programming systems that evolve around device- or software- initiated triggers. In more detail, it should not require users to define the conditions under which an action is performed based on the status of devices or services (e.g., if the microphone detects increased noise). On the contrary, since the main target group of LECTOR are non-technical users, the rule logic should be de-coupled from the artifacts and be human-oriented instead. For example, the condition "if no pressure is detected on the chair" becomes clearer to the simple user when expressed otherwise: "if the student is not sitting to his chair".

FR-7: Enable the combination of multiple behaviors performed by multiple actors when specifying the conditions under which a trigger is identified

The definition of a Trigger should not depend merely on the behavior of a single actor; on the contrary, the combination of more than one actor behaviors is required, so as to support the realization of more complex

scenarios (e.g., if the teacher is talking and the student is whispering then STUDENT IS CHATTING).

FR-8: Enable the definition of the conditions under which an intervention rule is initiated

The term **Intervention** is used to define the system-guided actions that aim to help or support the students in their activities. LECTOR should permit the definition of rules that describe the conditions (**Triggers** and **Virtual Context**) under which an intervention is initiated on a specific presentation host.

FR-9: Support the connection of N triggers with X interventions

In order to support the realization of complex scenarios, LECTOR should permit the creation of rules that combine multiple triggers with multiple interventions.

FR-10:Support the combination of multiple strategies when creating an intervention rule

The combination of multiple intervention strategies when defining an intervention rule results in the realization of much complex scenarios and subsequently in the creation of richer interventions. To this end, LECTOR should support this functionality.

FR-11:Permit the ranking and cancelation of interventions

In order to allow the educators to have the final say regarding the suggested interventions, LECTOR should enable them to rank or cancel the system suggestions.

FR-12:Provide a mechanism for assessing the efficacy of interventions

When the system employs an intervention that is estimated to be useful under a particular context of use, then after a reasonable amount of time it should be able to re-examine the student's behavior so as to identify whether the intervention was successful. Specifically, if it still detects that the student's behavior is unmodified, then the selected recommendation should be marked as ineffective in that context.

FR-13: Permit the integration of new software applications that can act as interventions

Intervention strategies are in fact applications running on private or public artifacts instantiated at a key point in time with appropriate content. In order to act as an intervention, an application is required to conform to AmI-Solertis [89] SaaS specifications, ensuring that it will be able to receive and execute LECTOR's commands. In order to be aware of the available intervention strategies, LECTOR should permit their integration so that the decision making components can employ them when deemed necessary.

FR-14: Enable the integration of new intervention hosts

Intervention hosts are (i) common computing devices such as smartphones, tablets, and laptops, (ii) technologically augmented everyday physical objects (e.g., interactive white boards, smart bulbs, etc.), or (iii) custom made items (e.g., student desk). An intervention host can either be used to launch applications (e.g., display an educational application instantiated with specific content) or control the physical environment (e.g., dim the lights during a video presentation). For LECTOR to optimally intervene, the available intervention hosts have to be properly defined in a way that conveys the information required for creating and deploying an intervention.

Non-Functional Requirements

NFR-1: Acceptance Testing Requirements

A full-scale user-based evaluation should be carried out to ensure developers' acceptance. Participants should have varying levels of experience programming applications for AmI environments. The overall score of a Standard Usability Scale (SUS) based post-evaluation questionnaire should be above 75%.

NFR-2: Documentation Requirements

A Quick Start Guide and context sensitive help should be provided.

NFR-3: Platform Compatibility Requirements

From a user perspective, the front-end tools for creating the rules should be accessible via any modern Operating System (OS) or web browser. From an engineering perspective, the generated rules should be OS-independent as long as the required hardware and software meet the necessary operational requirements (e.g., availability of the required software libraries, compliant runtime environment).

NFR-4: Maintainability Requirements

LECTOR should permit easy maintenance in the sense that faulty or worn-out components should be repaired or replaced without having to replace still working parts and any updates should be verified and validated before their final deployment.

NFR-5: Deployment Requirements

LECTOR should minimize any deployment requirements for its core components. Specifically, the automatically generated runtime components responsible for detecting behaviors, triggering and applying interventions should be self-contained and standalone, so as to ensure scalability and extensibility.

NFR-6: Interface Requirements

Software interface requirements include dealing with an existing software system, or any interface. In more detail, the LECTOR framework should respect and adhere to the formal specifications of any input-providing artifacts or artifacts that act as intervention hosts.

Chapter 5
System Architecture

This work introduces LECTOR, an extensible framework responsible for (i) monitoring the Intelligent Classroom environment to detect student behaviors that require remedial actions, and (ii) selecting appropriate interventions in order to help, or support them throughout the educational process. This chapter describes LECTOR's architecture in detail.

The Classroom behind LECTOR

Currently, LECTOR [90] is employed inside an in-vitro technologically augmented classroom where educational activities are enhanced with the use of pervasive and mobile computing, sensor networks, artificial intelligence, multimedia computing, middleware and agent-based software [41], [91], [92]. In more detail, the hardware infrastructure includes both commercial and custom-made artifacts, which are embedded in traditional classroom equipment and furniture. For example, the classroom contains a commercial touch sensitive interactive whiteboard, technologically augmented student desks [93] that integrate various sensors (e.g., eye-tracker, cameras, microphones, etc.), a personal workstation and a smart watch for the teacher, as well as various ambient facilities appropriate for monitoring the overall environment and the learners' actions (e.g., microphones, user-tracking devices, etc.).

The software architecture (Figure 15) of the Intelligent Classroom follows a stack-based model where the first layer, namely the AmI-Solertis middleware infrastructure [89], is responsible for (i) the collection, analysis and storage of the metadata regarding the environment's artifacts and (ii) their deployment, execution and monitoring in the AmI-Solertis-enabled systems to formulate a ubiquitous ecosystem. The next two layers, namely the ClassMATE and the LECTOR frameworks, expose the core libraries and finally the remaining layer contains the end-user applications responsible for delivering interventions and accepting user input. Specifically, ClassMATE [94] is an integrated architecture for pervasive computing environments that monitors the ambient environment and makes context-aware decisions; specifically, it features a sophisticated, unobtrusive, profiling mechanism in order to provide user related data to the classroom's services and applications. As far as the end-user applications are concerned, CognitOS [95] delivers to the students a sophisticated environment for educational applications hosting able to present interventions. Furthermore, two powerful tools aim to support educators in their daily activities; LECTORviewer [96], [97] provides an overview of the students' attention levels and asks the educator's opinion regarding ambiguous behaviors or scheduled interventions, while NotifEye provides notifications regarding important events occurring during the lesson time.

Currently, the Intelligent Classroom employs eye-trackers which can be used to observe students' fixations during studying on their personal computers (e.g., reading a passage, solving an exercise) to determine their attention level (e.g., stares at an insignificant area of the screen), their weaknesses (e.g., the student keeps reading the same sentence over and over again), their interests (e.g., fascinated with wild life) and their learning styles (e.g., attempts to solve the easier assignments first). The combination of eye-tracking data with the learner profile can not only reduce false positives, but also discover

personalized patterns that can be used to facilitate learning and reset attention when necessary (i.e., learning behavior).

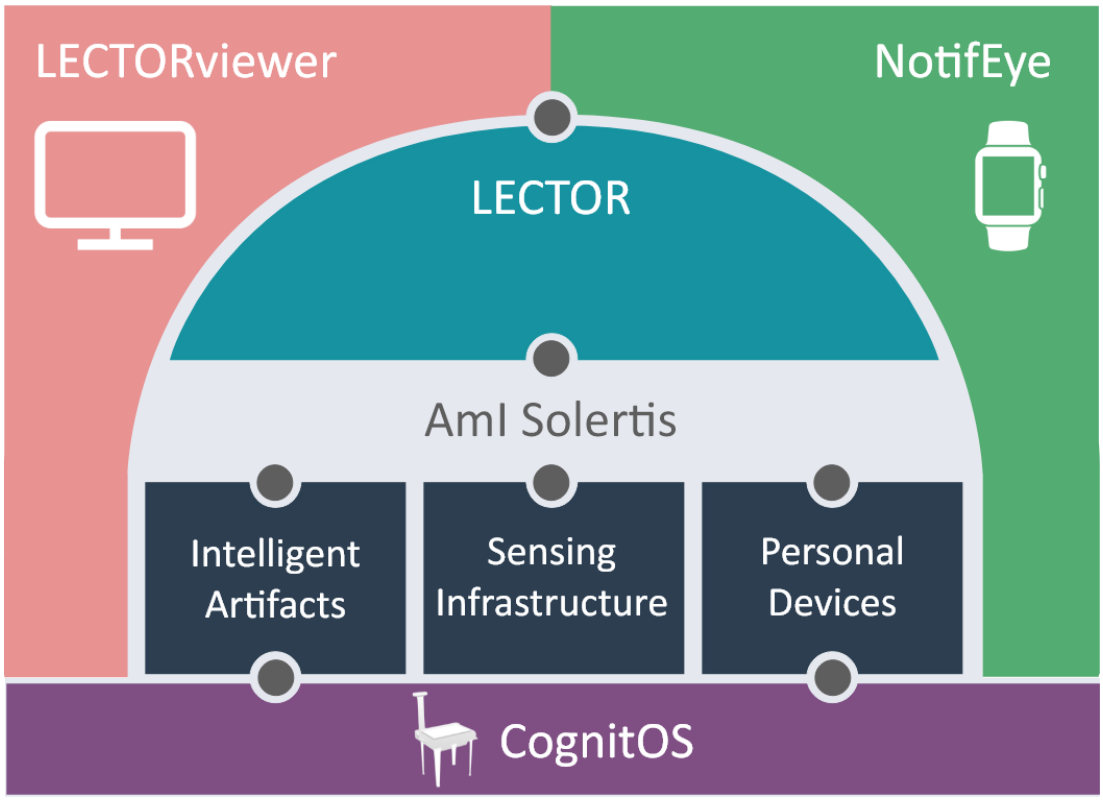

Figure 15: The architecture of the Intelligent Classroom

While being accurate in determining the direction in which the eyes are pointing within an application GUI, eye-trackers are very constraining since head motions out of specific limits could result to poor visibility of the user's eyes. As a result, these systems are not appropriate for analyzing the visual focus of attention in open spaces. To alleviate this constraint, sophisticated cameras (e.g., RGB-D camera such as Microsoft Kinect) can track the head pose of the learner to be used as a surrogate for gaze. The combination of eye-tracking and head pose tracking algorithms offers an accurate overview of what the students are looking at on the computer screen and on whom or what they are focused on (e.g., teacher, class board, etc.). Moreover, the use of cameras is ideal for tracking the body posture and the direction of an individual student, especially when taking into consideration that they constantly move inside the classroom even while seated. Besides learners' orientation, camera input also enables the identification of specific gestures

that indicate whether a student is paying attention to the lecture or not (e.g., a student raising his hand).

In addition to visual information, microphones can be used to predict of the students' focus of attention based on sound information. According to [54], the focus of attention is also correlated to sound sources, thus it is possible to estimate it based on information regarding who is talking at, or was talking before, a given moment. The microphones are placed on the teacher's and students' desks to identify who the collocutor is at any time, while this installation also permits monitoring of the classroom noise levels, which is a reliable indication of inattentive behavior on behalf of the students.

Finally, considering that students often get up from their seats, either because they conform to the teacher's requests (e.g., the teacher might ask a student to solve an exercise on the class board) or because they display inattentive behavior, it seems essential to track such situations. For that purpose, the pressure-sensitive sensors on each learner's chair can be used to identify whether the student is seated or not. This kind of information, if combined with data received from strategically placed distance sensors (e.g., near the class board, near the teacher's desk), introduces a primitive localization technique that can be used to estimate the location and the purpose of a "missing" individual (e.g., a student is off the desk near the board thus solving an exercise, the teacher might walk in the front of the classroom to assist a weak student that had just waved). However, if a full-blown localization system becomes available, the aforementioned solution will serve as an auxiliary validation system.

The next sections describe how the LECTOR framework enables the Intelligent Classroom to intervene appropriately when the students require help or support.

LECTOR Framework Outline

LECTOR exploits the potential of AmI technologies to observe either human- or artifact-oriented behaviors (SENSE), identify whether they require remedial actions (THINK) and intervene (ACT) accordingly -when deemed necessary- in order to fulfill the user needs. According to cognitive psychology, the sense-think-act cycle stems from the processing nature of human beings that receive input from the environment (perception), process that information (thinking), and act upon the decision reached (behavior) [30]. This identified pattern constitutes the base for many design principles regarding autonomous agents and traditional AI [31].

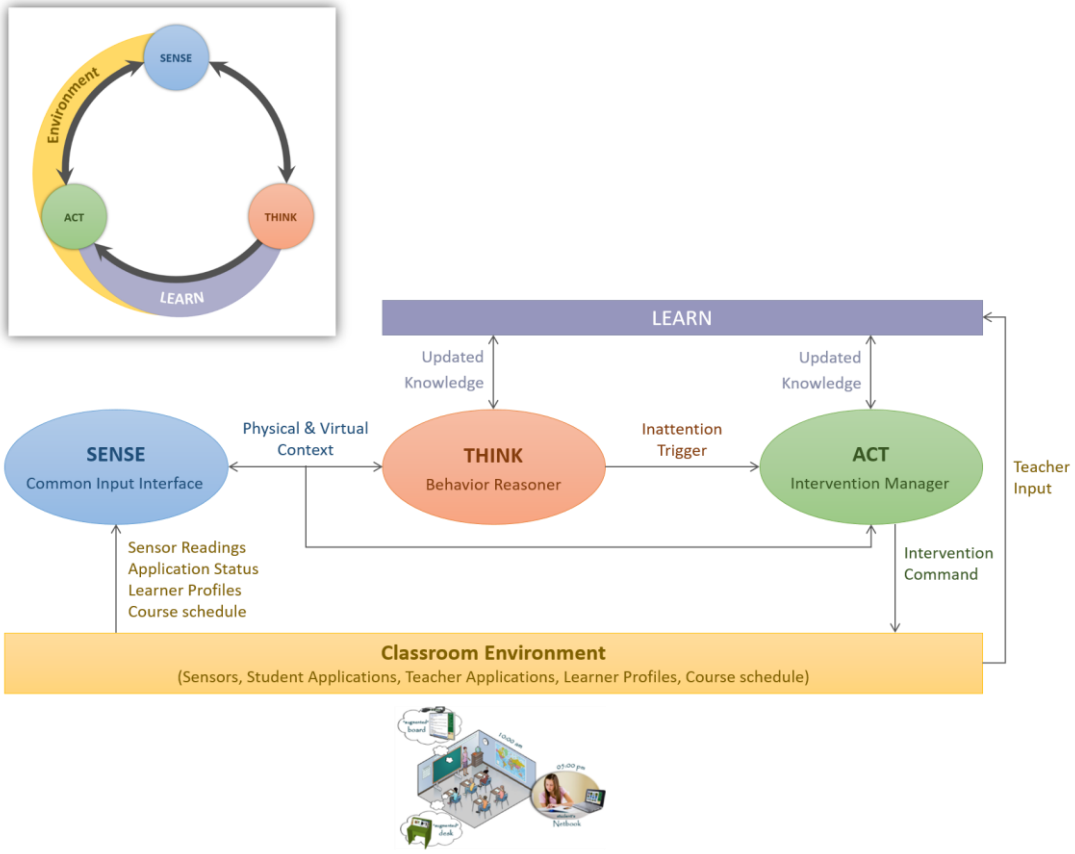

Figure 16: The SENSE-THINK-ACT model extended with the notion of LEARN

LECTOR heavily depends on contextual information in order to (i) make informed decisions regarding the meaning of an identified behavior, (ii) select appropriate interventions according to student needs. The term *Context of Use* is defined as follows: *"Any information that can be used to characterize the*

situation of entities (i.e., person, place, object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects" [98]. Based on the above, LECTOR relies on an extensible modeling component responsible for collecting and exposing the necessary information.

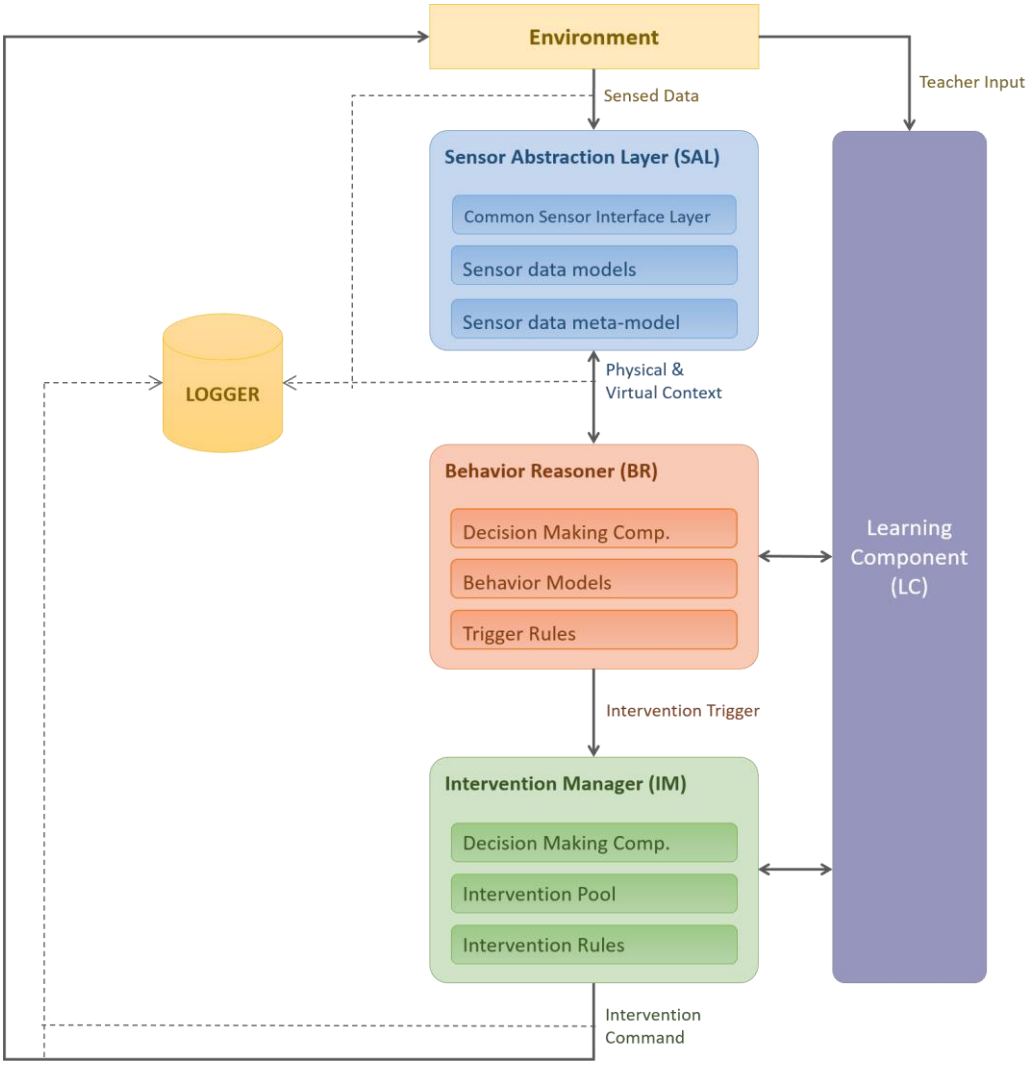

Figure 17: System architecture

Furthermore, LECTOR extends the SENSE-THINK-ACT model by introducing the notion of LEARN (Figure 16). The fact that the nature of this system enables continuous observation of behaviors creates the foundation for a mechanism that provides updated knowledge to the decision-making components. In more detail, the learning mechanism is able to (i) incorporate knowledge

provided by educators in order to disambiguate identified behaviors (e.g., staring at the ceiling might indicate that a particular student is thinking instead of being distracted) or assess the acceptance of an intervention (e.g., the educators cancels an multimedia presentation that was automatically scheduled by the system) , and (ii) auto-rank the suggested interventions according to their efficacy (e.g., if the stress levels of a student remain high despite the fact that an intervention was initiated, then that particular intervention is marked as inefficient).

Figure 17 presents LECTOR's architecture, which consists of four core components: the Sensor Abstraction Layer (SAL), the Behavior Reasoner (BR), the Intervention Manager (IM), and the Learning Component (LC). The Sensor Abstraction Layer (SAL) is responsible for monitoring the environment and transforming the raw sensor readings into meaningful high-level objects. These objects constitute the Physical Context and Virtual Context, which are processed by the Behavior Reasoner (BR) in order to detect behaviors that should trigger interventions. As soon as a trigger is identified, the Intervention Manager (IM) consults the Virtual Context and starts an exploratory process in order to select an intervention suitable for the current situation. Both the Behavior Reasoner and the Intervention Manager are open to user suggestions that can override their defaults (e.g., the educator can reject or postpone an intervention).

In order to support both developers and educators in defining the **behaviors** that lead to context-aware **interventions**, LECTOR features a sophisticated tool, named LECTORstudio. Such tool:

- Supports the creation of **user-oriented** behavior-intervention scenarios in contrast to the artifact-oriented recipes that are currently supported by the majority of IFTTT-style tools [21], [27].
- Enables the definition of behaviors that combine **multiple contextual information.**
- Permits the definition of multimodal and ubiquitous interventions.

- Supports the connection of N behaviors with M interventions.

Since the decision-making mechanisms of LECTOR rely on **rule-based** conditions in order to identify **behaviors** that trigger appropriate **interventions**, LECTORstudio supports the creation of three (3) types of rules (Figure 18); even if this decomposition increases the number of steps that a user must complete in order to connect a trigger to an intervention, it offers scalability and better rule management. The supported types of rules are:

I. Rules that "model" a **behavior**[5] based on **physical context**[3].
II. Rules that "model" the **triggers**[6] based on the **behavior**[5] of **actors**[1] under specific **virtual context**[4].
III. Rules that specify the conditions (i.e., **triggers**[6] and **virtual context**[4]) under which an **intervention**[7] is initiated on a specific **intervention host**[2].

The core concepts of this rule-based approach are explained below:

1. **Actors** are the users of the intelligent environment whose behavior needs to be monitored in order to decide whether an intervention is required.
2. **Intervention hosts** can either launch an application with specific content or control the physical environment. They are: (i) common computing devices such as smartphones, tablets, and laptops or (ii) technologically augmented everyday physical objects (e.g., interactive white boards, smart lamps, etc.), or (iii) custom made items (e.g., student desk).
3. The **physical context** encapsulates information regarding physically observable phenomena via sensors (e.g., luminance, heart rate, sound levels, etc.).
4. The **virtual context** refers to any static or dynamic information that is provided through software components (e.g., student profiles, student agenda, course schedule).

5. **Behavior** is the way that a user or an artifact acts (e.g., a user talks, a device switches on).
6. **Trigger** is the "model" of a high-level behavior that can initiate an intervention.
7. **Interventions** are the system-guided actions that aim to help, support or comfort users in their activities.

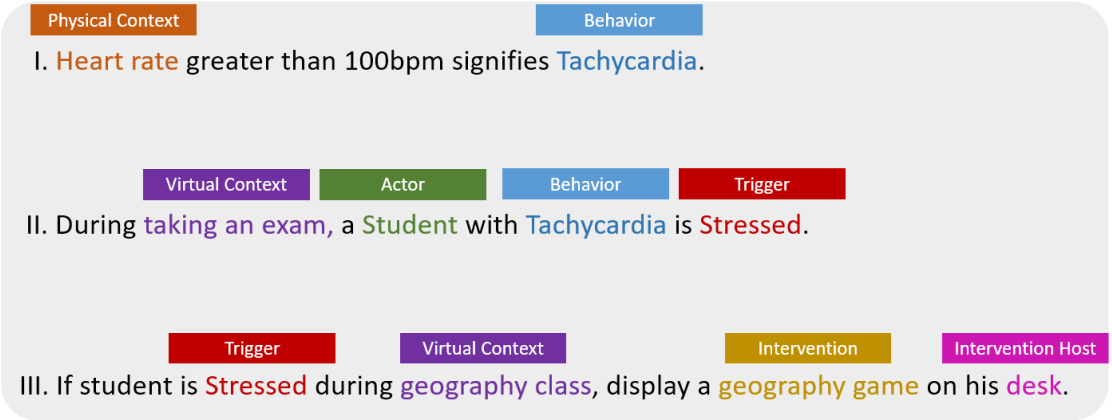

Figure 18: Indicative examples of rule types supported by LECTOR

In the next sections, the respective software components that comprise the overall system architecture of the Sense-Think-Act-Learn model will be described in details.

SENSE: Sensor Abstraction Layer

LECTOR's decision-making mechanisms heavily depend on contextual information to (i) identify the actual conditions that prevail in the intelligent environment at any given time and (ii) act accordingly. The term context has been used broadly with a variety of meanings for context-aware applications in pervasive computing [99]. The authors in [100] refer to contexts as any information that can be detected through low-level sensor readings; for instance, in a home environment those readings include the room that the inhabitant is in, the objects that the inhabitant interacts with, whether the inhabitant is currently mobile, the time of the day when an activity is being performed, etc.

However, in the envisioned Intelligent Classroom contextual awareness goes beyond data collected from sensors. Despite the fact that sensorial readings are important for recognizing **behaviors**, in some cases they are inadequate to signify whether a behavior should **trigger** an **intervention.** To this end, LECTOR utilizes static and dynamic information such as the characteristics of the users, the nature of the task at hand, the user agenda, etc., in order to reach to valid conclusions. This work employs the term **Physical Context** to indicate data collected from sensors, whereas the term **Virtual Context** is used for any static and dynamic information provided through software components [101].

The exploitation of such contextual information enables the THINK component to identify behaviors that should trigger interventions. Despite the fact that recognizing a behavior mainly relies on sensor readings, the Virtual Context is critical to interpret them correctly. For instance, inside a classroom environment, excess noise typically indicates that students talk to each other instead of listening to the teacher; however, this assumption is incorrect during the music class, where the students are expected to sing loudly.

Furthermore, the Virtual Context is essential for the ACT component, which when instructed to intervene, it selects an appropriate intervention and a

suitable host for it. These decisions depend heavily on such information; as an example, if an intervention occurs during the first ten minutes of a lecture, where the main topic has not been thoroughly analyzed by the teacher yet, the system starts a short preview that briefly introduces the lecture's main points using entertaining communication channels (e.g., multimedia content).

Physical Context (Physiological Cues)

LECTOR requires information regarding the behaviour of the classroom **actors,** as well as their **specific physical** properties, in order identify inattentive behaviors; the available actors inside a classroom are either teachers, students or the classroom itself. In order to observe how they communicate and interact during a course, LECTOR currently monitors their physical characteristics (i.e., physical context) which are considered appropriate cues that might signify inattention, and translates them –in a context-dependent manner– into specific activities classified under the following categories: Focus, Speech, Location, Posture and Feelings (Figure 19).

Focus. Identifying an individual's visual focus of attention provides rich information regarding that person, i.e., what is she interested in, what is she doing, or how does she react to different visual stimuli [62]. Inside the classroom, the visual focus of attention of students or teachers might be drawn to other individuals or static objects (e.g., the class board, the window, the world map hanging on the wall, etc.).

Speech. Being able to understand who is speaking during a course and at what sound intensity (e.g., whispering, shouting, talking, etc.) is undoubtedly helpful for deciphering student-to-teacher or student-to-student communication and interaction. That kind of information could possibly reveal behaviors such as chatting with classmates while the teacher is lecturing.

User Location & Posture. Some behaviors that could be classified as disruptive are related to the students' posture and location inside the classroom. These behaviors include (but are not limited to) getting out of seat, tapping feet, turning the head or the whole body toward another person, etc. [43].

Feelings. A student's learning capabilities can be compromised due to feelings of fatigue [102] (i.e., Drowsiness, Falling Asleep), while stress and anxiety have the same negative impact to learning [103].

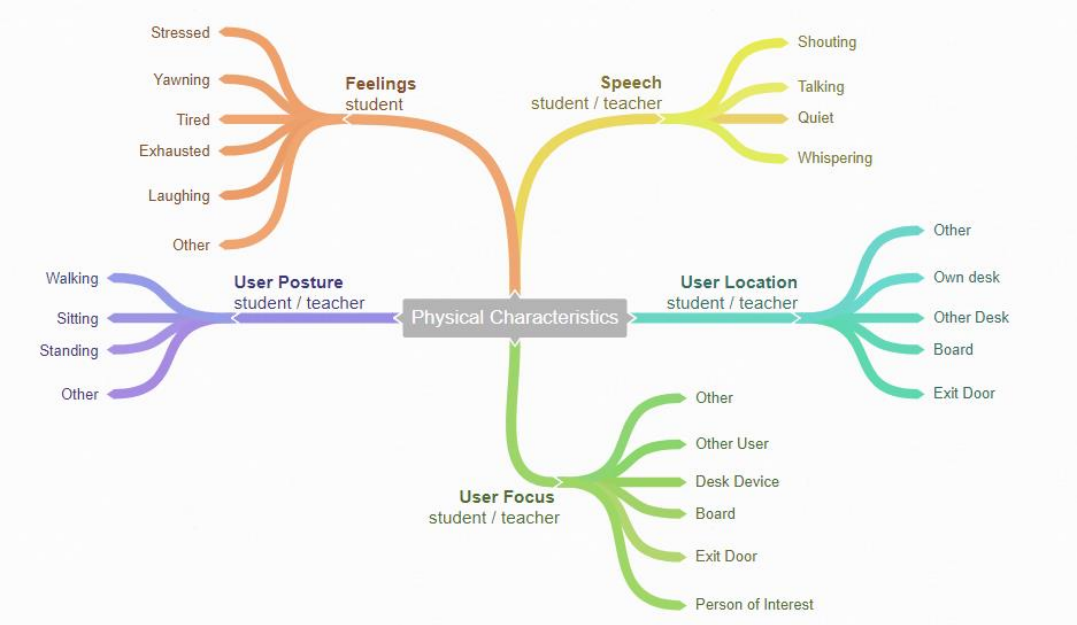

Figure 19: The user's physiological cues that should be monitored inside a classroom environment in order to identify inattentive behaviors

Virtual Context

In the Intelligent Classroom, the decision-making mechanisms of LECTOR must be able to identify the context (e.g., student status, lecture progress, task at hand, etc.) at any given time. In more detail, LECTOR must rely on detailed information regarding the nature of each course, such as: (i) topics to be covered by the course's syllabus, (ii) chapter organization, (iii) topic difficulty and related concepts, (iv) assignments' descriptions, (v) deadlines, (vi) contributions to the final score, etc., to make appropriate choices regarding behavior classification or intervention selection.

Furthermore, each learning activity type (e.g., lecture, exam, exercise solving, etc.) has its own rules and standards that ensure its smooth realization and guarantee optimal benefits for the students. To this end, each type is accompanied by a list of irrelevant and/or undesired activities, which are used to disambiguate behaviors that in some contexts are classified as attentive and in others as inattentive. For example, personal "thinking" is anticipated during a written exam, however this is not the case when the teacher is giving a lecture.

Additionally, information concerning each individual student's learning behaviors (e.g., concentrates easier while doodling, etc.), are invaluable for interpreting activities that indicate attention for some but inattention for others. In order to capture such information, the LECTOR profiling mechanism is used to store students' static personal data (full name, date of birth, etc.) along with dynamic context-sensitive data gathered through interaction monitoring, such as topics of interest or dislikes, weaknesses or strengths, general knowledge, progress and preferred types of learning resources.

Based on the above, in order to make appropriate decisions either when attempting to identify a behavior or when investigating possible interventions, LECTOR must be aware of the current **course**, the ongoing **activity**, and the characteristics of the involved **students**. Students with similar characteristics in terms of academic process (e.g., Advanced, Intermediate or Weak students), behavioral norms (e.g., Loud Speakers, Loud Thinkers), learning or behavioral disorders (e.g., ADHD), etc. are part of the same **student group(s)**.

SENSE Component Architecture

LECTOR aims to be deployed in Intelligent Classrooms that incorporate infrastructure able to monitor users and artifacts, so as to provide the necessary input to the decision-making components for estimating whether their behavior requires remedial actions. Such environments are dynamically

formed due to the abundance of services and devices that constantly change their availability. Hence, in order to ensure scalability, LECTOR is not bound to specific technological solutions, but rather enables developers to seamlessly integrate new data sources through an intermediate software layer that maps the sensed data (i.e., physical and virtual context) to abstract models that LECTOR is able to process.

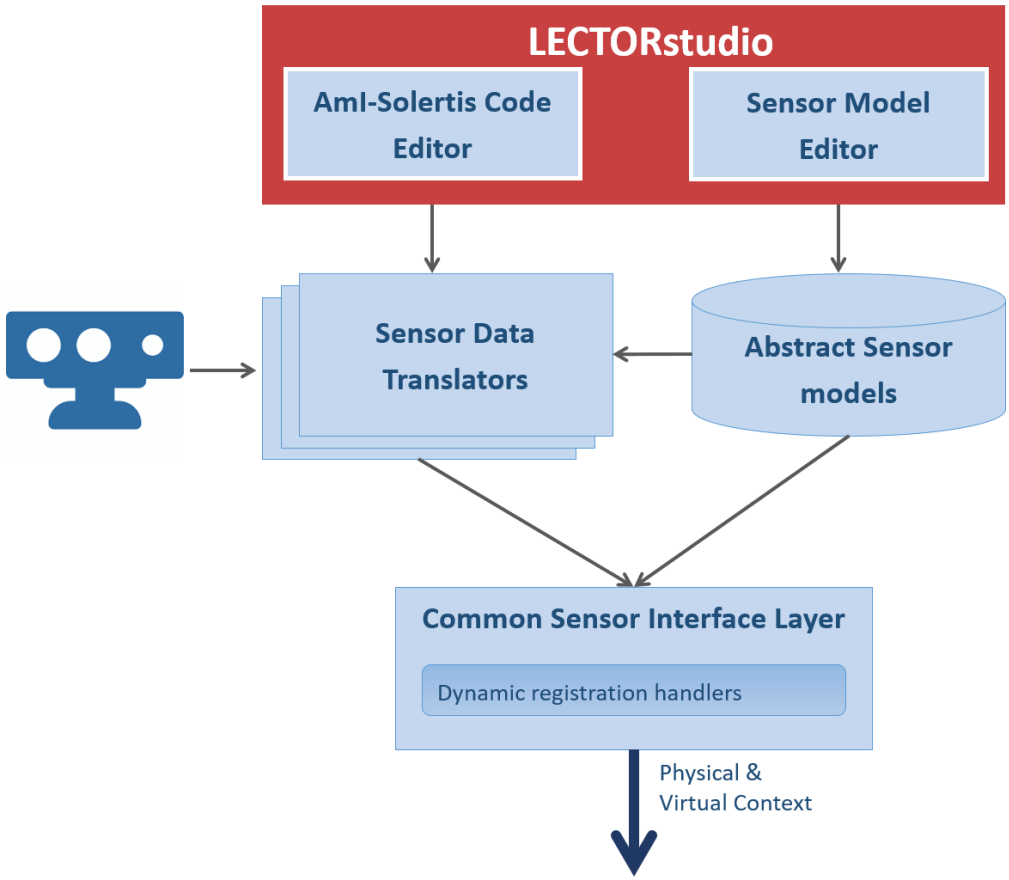

Figure 20: Sensor Abstraction Layer (SAL)

A newly imported sensor is able to advertise its availability and inform the decision-making mechanisms about the characteristics of the environment that can be monitored and assessed. Upon a new installation, the Sensor Abstraction Layer (SAL), an overview of which is depicted in Figure 20, uses standard control APIs to initialize the appropriate delegates, monitor the operation of the installed sensor and propagate its events to the decision layers. For instance, a component that performs human localization notifies

the system that it can provide the location of the user inside the environment. As soon as that happens, SAL initializes the appropriate components and whenever a new event is emitted due to user movement, it is broadcasted to the interested components (i.e., Behavior Reasoner).

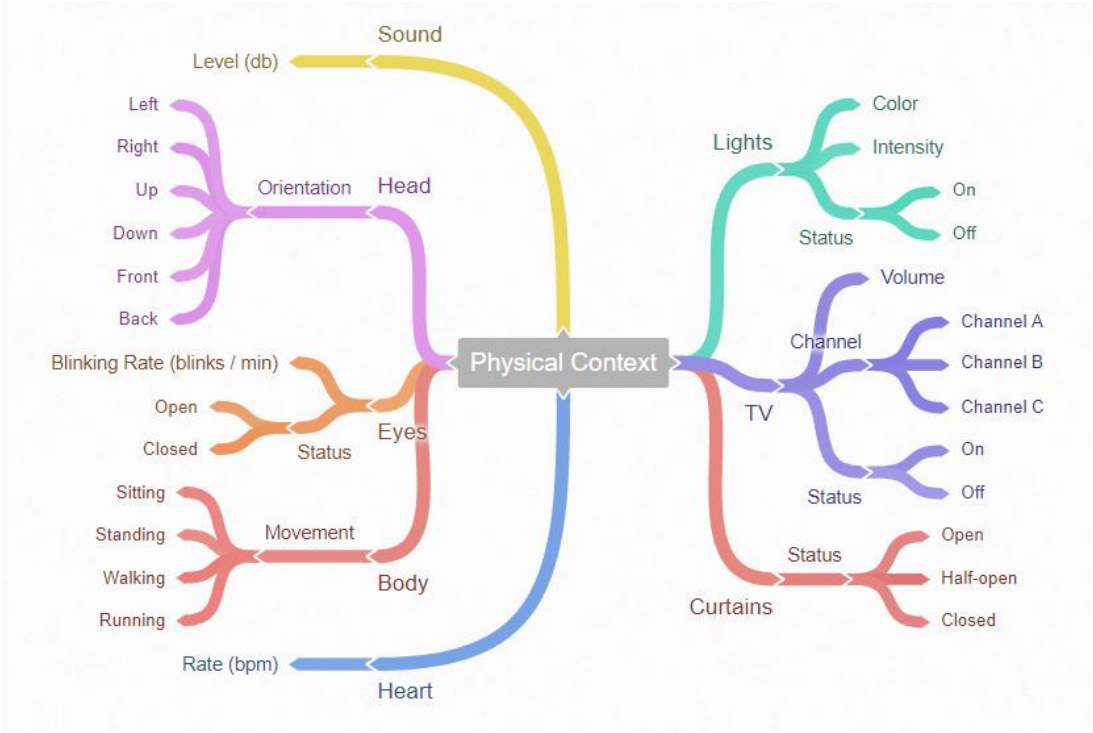

Figure 21: Examples of Sensor Models that represent the input values expected by LECTOR

Inside an AmI environment the available input sources might range from simple converters (or chains of converters) that measure physical quantities and convert them to signals which can be read by electronic instruments, to software components (e.g., a single module, an application, a suite of applications, etc.) that monitor human computer interaction and data exchange. Undoubtedly, the input types, values and formats are tightly coupled with the input source that generated them, hence, this highlights the need for an appropriate federation mechanism that will alleviate this diversity and will facilitate interoperability. To this end, LECTOR permits developers to create Sensor Models that represent the expected input values; these models are the common ground between SAL and the other decision layers (i.e., Think, Act and Learn components), which rely on high-level values instead of raw sensor data. Figure 21 presents some indicative examples of Sensor

Models; for instance, the decibel (dB) is used to the measure the sound levels of the environment, while monitoring the human body activities provides specific values about the movement of the user (i.e., standing, sitting, walking, running). For each one of these models, developers can provide one or more Sensor Data Translators (Figure 22) which are responsible for effectively translating the received data before forwarding them. In some cases, this requires a few lines of code (e.g., converting sound frequency (Hz) to sound levels (dB)), while in others developers have to provide more sophisticated algorithms (e.g., detection of human walking).

As already mentioned, the term sensor is not used only to describe hardware infrastructure, but it encapsulates software components as well. Consider the following example of a Sensor Data Translator that requires simple mapping of the data received from an end-user puzzle game application, to the variables of the appropriate Sensor Model. The application's service is able to emit the following messages: (i) INACTIVE – when the user has not interacted with the system for more than 5 minutes, COMPLETE – when the user has finished the puzzle, and (iii) HINT – when the user requests help to continue. In case LECTOR has already another model for integrating similar educational games (e.g., IDLE, DONE and HELP), the developer responsible for integrating the puzzle game must provide the code that makes the appropriate mapping (i.e., INACTIVE → IDLE, COMPLETE → DONE and HINT → HELP).

Finally, the Common Sensor Interface Layer provides mechanisms that enable the dynamic registration of sensors, allowing their run-time connection and disconnection.

LECTORstudio supports developers in providing the information required during the integration of a new sensor. In more detail, LECTORstudio offers a user-friendly interface through which they can (i) define the data models that LECTOR's reasoning components will use, by defining the variables that will carry the actual values at runtime, (ii) select one or more services that deliver information collected from a particular sensor, and (iii) provide the code that

translates the data received from the sensor services to the types described by the models. For example, a service exposed by the student's wrist band is able to transmit data regarding her physiological signals. The user interface of LECTORstudio is described in details in Chapter 6.

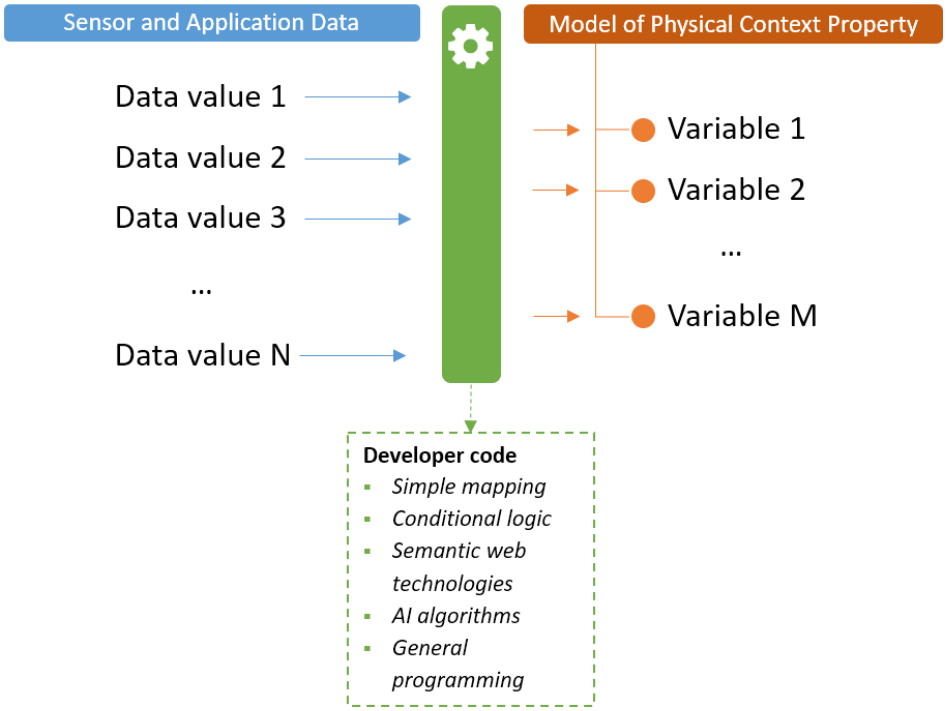

Figure 22: A Sensor Data Translator contains code responsible for effectively translating the received data before forwarding them

THINK: Behavior Reasoner

The Behavior Reasoner (BR) is responsible for identifying behaviors that require remedial actions; therefore, it constantly monitors the environment and when necessary it notifies the intervention manager to decide when and how to act. The Decision-Making Component of BR (Figure 23) constitutes the core of the THINK mechanism as it collects all the rules that describe behaviors and triggers and feeds them with the Physical and Virtual Context generated via Sensor Abstraction Layer. Whenever a stimulus is detected by the SAL component, the Decision-Making Component initiates an exploratory process to determine whether the incoming event indicates that the user needs help or support.

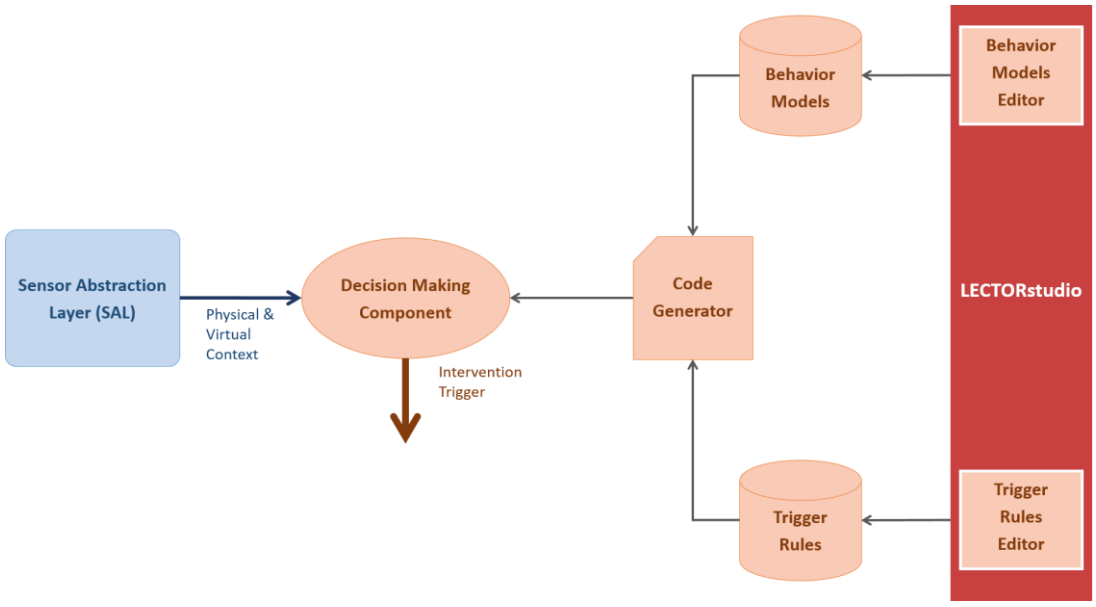

Figure 23: Behavior Reasoner

The first step of this process is to identify whether the sensed data denote a human- (e.g., the user is walking) or artifact- behavior (e.g., the oven is on). The modeling of such behaviors is realized in the form of high-level if-then rules, which combine multiple diverse information acquired from the Physical Context (Figure 24a). For instance, consider the behavior "Tachycardia" which is modeled as: "*Heart rate greater than 100bpm signifies Tachycardia*"; in this example, the "Heart rate" is a Physical Context attribute whose elevated value (> 100bpm) signifies a specific behavior (i.e., Tachycardia).

a.

b.

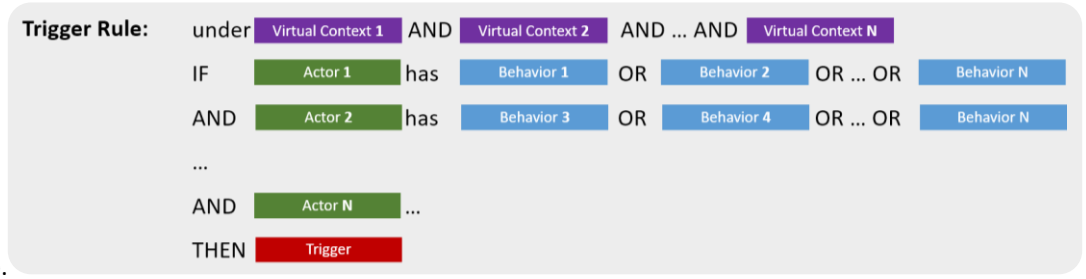

Figure 24: The conceptual representation of (a) the behavior model (b) the trigger rule

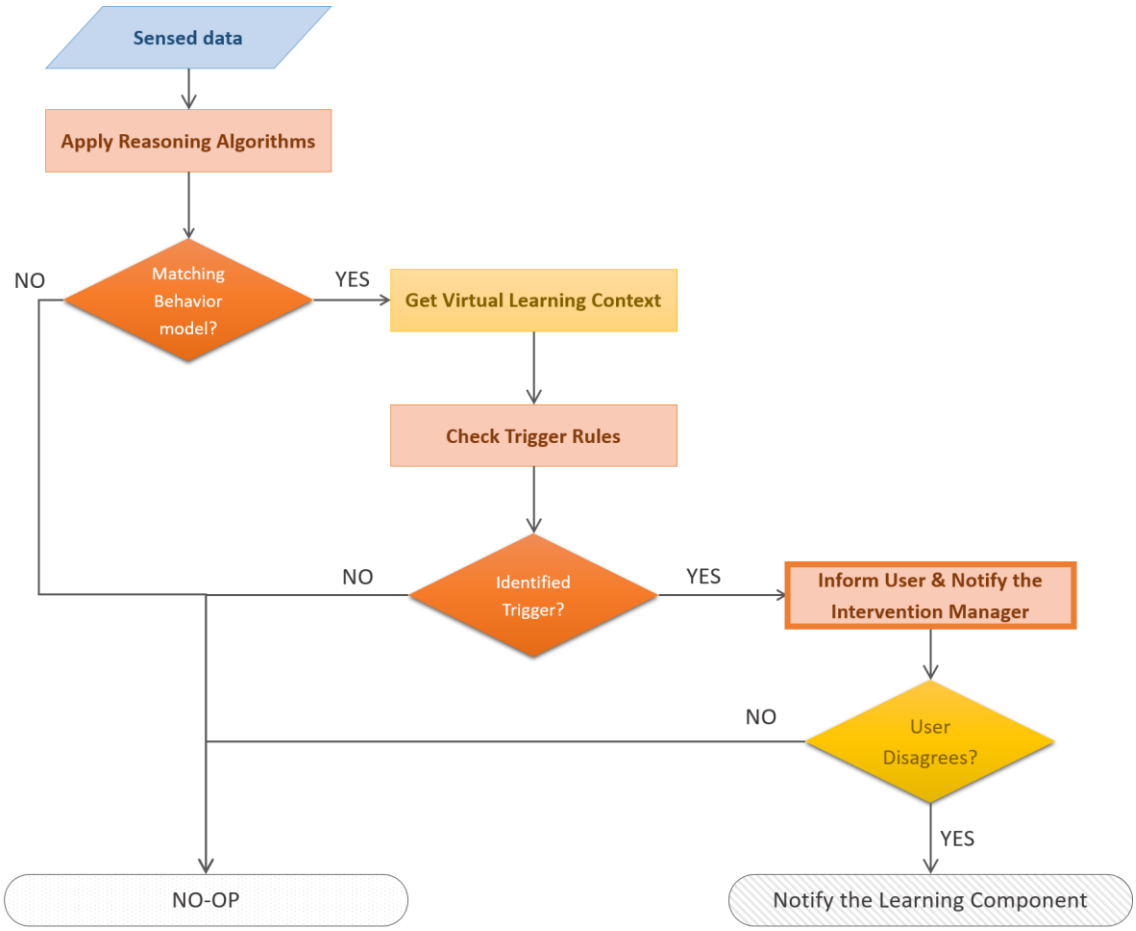

Figure 25: Detect Inattention Flowchart

As soon as a behavior is detected, the Decision-Making Component further examines the current Virtual Context, so as to identify whether that particular behavior should trigger an intervention. Similarly to the description of the Behavior Models, the decision logic that leads to the identification of

intervention triggers is also defined in the form of high-level "if-then" rules (Figure 24b). These rules combine various parameters such as identified behavior, virtual context, actor, etc., to define the conditions under which the system needs to intervene. The outcomes of these rules are high-level behaviors, named triggers. When they are identified, the Intervention Manager is notified so as to select the appropriate interventions. Consider the following rule that describes the trigger "Stress": "*While taking an exam, a Student with Tachycardia is Stressed*". In the former example, the virtual context (i.e., exam period) under which the behavior "Tachycardia" appears, reveals that the user is in a situation (stress) that requires a remedial action. The existence of various triggers is really important since they denote situations that require special treatment from the Intervention Manager. For example, Tachycardia due to Stress requires different handling than Tachycardia due to a medical condition. The entire process of detecting a trigger is depicted in the flowchart of Figure 25.

Observing the physical characteristics of the classroom actors described previously in this Chapter, empowers the identification of specific behaviors (e.g., Talking, Looking at the Door, Looking at a student, Walking) that might occur during a course. Out of context, such behaviors do not provide evidence that a student is distracted from the educational process. On the contrary, when examined in conjunction with the behavior of any other implicated actors and combined with appropriate contextual information (e.g., current activity, current course), they might reveal a situation that requires remedial actions (e.g., CHEAT, CHAT, DISTURB, FATIGUE, BOREDOM, OUT OF SEAT). For example, identifying that a student is talking does not signify that he is being inattentive; however, identifying that a student is talking to a classmate while the teacher is lecturing provides strong evidence that he is chatting.

In order to support extensibility, LECTORstudio accommodates the composition of new behavior models and trigger rules respectively (see Chapter 6). This is an invaluable asset not only for developers, but for non-

technical users as well, which are enabled to create their own behavior-intervention scenarios.

Upon creation, both behavior models and trigger rules are stored in a database. Figure 26, presents the employed database schema; in the case of a behavior model, the key “properties” is used to store information regarding the physical characteristics (i.e., physical context) that should be monitored so as to identify a behavior. As far as the trigger rules are concerned, the key “context” refers to the virtual context under which the rule should be evaluated and the key “actors” describes the users that should be monitored for specific behaviors. In both cases, the “outcomeId” is the foreign key to the table containing the general details of the behavior / trigger (e.g., name, description).

Before deployment, the stored data are retrieved and the relevant executable code is generated so as to be used by the decision-making component. In more detail, the data are translated to Javascript code (Figure 27) using Handlebars.js [104], an extension of the Mustache web template system [105]. Both Handlebars and Mustache are logic-less templating languages that keep the view and the code separated, while Handlebars is considered one of the most advanced libraries available. Its main difference from Mustache is that it permits the developers to add their own helpers (i.e., custom logic). Using such a templating system in order to generate the executable code for both behavior models and trigger rules ensures that LECTOR is not bound to a specific implementation. On the contrary, by simply replacing the templates, the system can either employ alternative reasoning techniques or be ported to different platforms.

```
var RuleSchema = new mongoose.Schema({
    name: String,
    ruleCategoryId: {
        type: mongoose.Schema.ObjectId,
        required: true
    },
    context: {}.
    actors: [{
        actorId: String,
        outcomes: [{
            oid: { type: String },
            cid: { type: String },
            time: { type: Number },
            timeType: { type: String },
            percentage: { type: Number },
            type: { type: String } //Include, Exclude
        }]
    }],
    properties: [{
        propertyId: String,
        attributes: [{
            name: { type: String },
            value: {},
            unit: { type: String },
            type: { type: String },
            function: { type: String },
            duration: { type: Number },
            timeType: { type: String },
            percentage: { type: Number }
        }]
    }],
    outcomeId: {
        type: mongoose.Schema.ObjectId,
        required: true
    },
    enabled: { type: Boolean, default: true },
    activations: { type: Number, default: 0 },
    success: { type: Number, default: 0 },
    creationDate: { type: Date, default: Date.now },
    author: { type: String, default: 'Maria Korozi' }
});
```

Figure 26: The database schema for both the behavior models and trigger rules

a.

```
//INITIALIZE ACTORS
var ruleActors = [{{#each Actors}}'{{@key}}'{{^if @last}}, {{/if}}{{/each}}];
var actorProperties = {};
{{#each Actors}}
actorProperties.{{@key}} = [{{#each this}}{ {{! Iterate Actor Properties}}
{{#each this}}"{{@key}}": {{#xif @key 'Type'}}{{@this}}{{else}}"{{@this}}"{{/xif}}{{^if @las
{{/each}}
}{{^if @last}}, {{/if}}{{/each}}];
{{/each}}

//INITIALIZE CONTEXT
var ruleContext = {};
{{#each Context}}
ruleContext.{{@key}} = [{{#each this}}'{{name}}'{{^if @last}}, {{/if}}{{/each}}];
{{/each}}

var conditions = {};
{{#each Actors}}
conditions.{{@key}} = {};
{{/each}}

//INITIALIZE SERVICES
{{#Services}}
var {{Service}} = require('{{Service}}');
{{/Services}}

//CREATE FUNCTIONS
var actors = context.getClassroomActors();

{{#each Actors}}
// {{@key}}
if(_.indexOf(actors, '{{@key}}') && _.indexOf(ruleActors, '{{@key}}')) {
  var {{@key}}Hosts = context.getAll{{@key}}Hosts();

  for (var i = 0; i < {{@key}}Hosts.length; i++) {
    conditions.{{@key}}[{{@key}}Hosts[i]] = {};
  }

{{#each this}}
  // {{@../key}} {{Category}}
  for (var i = 0; i < {{@../key}}Hosts.length; i++) {
    var tmp{{Category}} = {{Category}}.create{{Category}}({{@../key}}Hosts[i]);
    tmp{{Category}}.onChange((value) => {
            conditions.{{@../key}}[{{@../key}}Hosts[i]].{{Category}} = value;
            checkForInattention();
        });
```

b.

```
//INITIALIZE ACTORS
var ruleActors = ['Teacher', 'Students'];
var actorProperties = {};
actorProperties.Teacher = [{
    "Category": "Speech",
    "Outcome": "Talking",
    "Duration": "30",
    "Type": true
}];
actorProperties.Students = [{
    "Category": "Speech",
    "Outcome": "Talking",
    "Duration": "30",
    "Percentage": "25",
    "Type": true
}, {
    "Category": "Focus",
    "Outcome": "Teacher",
    "Duration": "30",
    "Percentage": "25",
    "Type": false
}];

//INITIALIZE CONTEXT
var ruleContext = {};
ruleContext.Courses = ['Maths', 'History'];
ruleContext.Activities = ['Lecture', 'Exam'];
ruleContext.Outcomes = ['Cheat'];

var conditions = {};
conditions.Teacher = {};
conditions.Students = {};

//INITIALIZE SERVICES
var Speech = require('Speech');
var Focus = require('Focus');

//CREATE FUNCTIONS
var actors = context.getClassroomActors();

// Teacher
if (_.indexOf(actors, 'Teacher') && _.indexOf(ruleActors, 'Teacher')) {
    var TeacherHosts = context.getAllTeacherHosts();

    for (var i = 0; i < TeacherHosts.length; i++) {
        conditions.Teacher[TeacherHosts[i]] = {};
```

Figure 27: (a) Snapshot of the Handlebars template, and (b) the generated *.js code

ACT: Intervention Manager

Intervention Types

The term intervention is defined as "*involvement in a difficult situation in order to improve it or prevent it from getting worse*" [29]. Here it is used to describe the system-guided actions that subtly interrupt a course's flow in order to (i) draw the educator's attention in problematic situations, and (ii) re-engage distracted, unmotivated or tired students in the educational process. Actually, interventions are applications running on private (e.g., student's desk, teacher's watch) or public (e.g., classroom board) hosts, instantiated at a key point in time with appropriate content. Literature review reports that several types of interventions can prove to be beneficial in various situations occurring in an educational setting (summarized in Table 2).

Currently, LECTOR features two types of interventions that have been created in order to ensure active student participation in the main course. Particularly, the student desk and the classroom board are able to instantiate **quizzes** and **multimedia presentations**, whose appropriate content can keep students motivated. These interventions can be applied either to individuals on their private artifacts or the entire classroom when displayed on the publicly available board.

Furthermore, taking into consideration the fact that most students thrive in encouraging environments where they receive specific feedback, their private artifacts (i.e., desk and smart watch) are equipped with a messaging mechanism able to provide **encouraging messages** when deemed necessary. The same mechanism is employed on the teacher's smart watch in order to display subtle messages suggesting changes in the lecture format. In more detail, LECTOR is able to suggest **recapitulation** of the lecture topics, **initiation of a discussion relevant** to the current course, **repetition of specific** material and continue **lecturing at a lower pace**.

Table 2: List of available educational interventions

Intervention	Technique	Recipient	Hosts	Range
Quiz	Active Learning	Students	Student Desk, Classroom Board	Individual, Class-wide
Multimedia Presentation	Active Learning	Students	Student Desk, Classroom Board	Individual, Class-wide
Encouraging Messages	Encouragement	Students	Student Desk, Student Smart Watch	Individual
Initiate Discussion	Changing Pedagogies	Teacher	Teacher Smart Watch	Class-wide
Recapitulation	Changing Pedagogies	Teacher	Teacher Smart Watch	Class-wide
Repeat course material	Changing Pedagogies	Teacher	Teacher Smart Watch	Class-wide
Lower pace	Changing Pedagogies	Teacher	Teacher Smart Watch	Class-wide

ACT Component Architecture

As soon as an intervention trigger is detected, the Decision-Making Component of the Intervention Manager (Figure 28) initiates an exploratory process to identify the most appropriate course of action. Evidently, selecting a suitable intervention and the proper artifact for hosting it is not a straightforward process, as it requires multi-stage analysis of the Virtual Context. The first step to accomplish is to consult the “Intervention rules”; similarly to the “Trigger rules”, they are high-level “if-then” rules (Figure 29) describing the conditions under which one or more interventions should be initiated (e.g., if the user has been sitting watching movies for over an hour, display a notification to the TV suggesting a short walk).

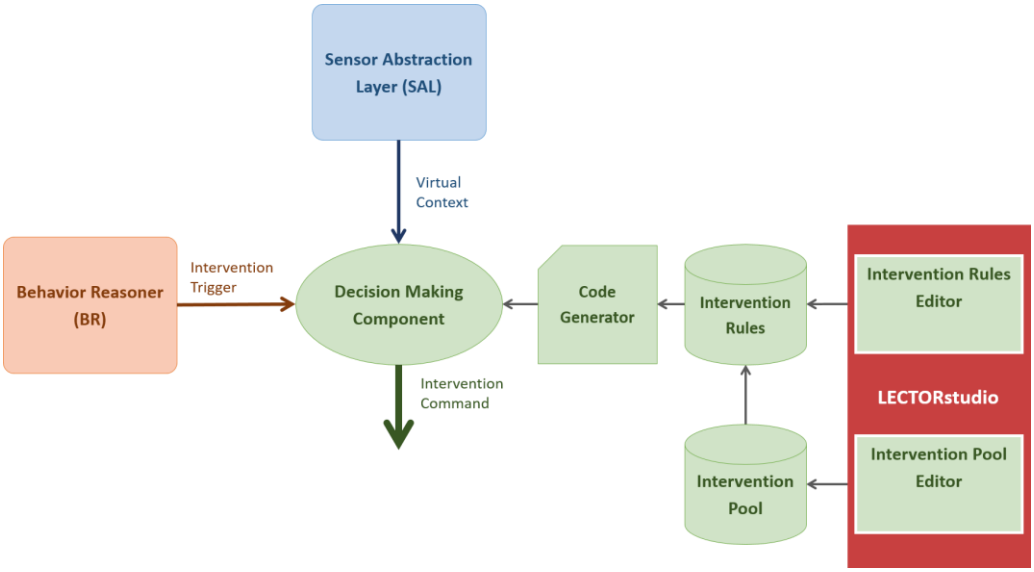

Figure 28: Intervention Manager

Each intervention rule, upon evaluation, points to one or more intervention strategies residing in the "Interventions' Pool" (IP). The IP includes high-level descriptions of the available strategies, which are in fact applications that can be used as channels to present interventions dictated by LECTOR. In order to act as an intervention, an application is required to conform to AmI-Solertis [89] SaaS specifications, ensuring that it will be able to receive and execute LECTOR's commands.

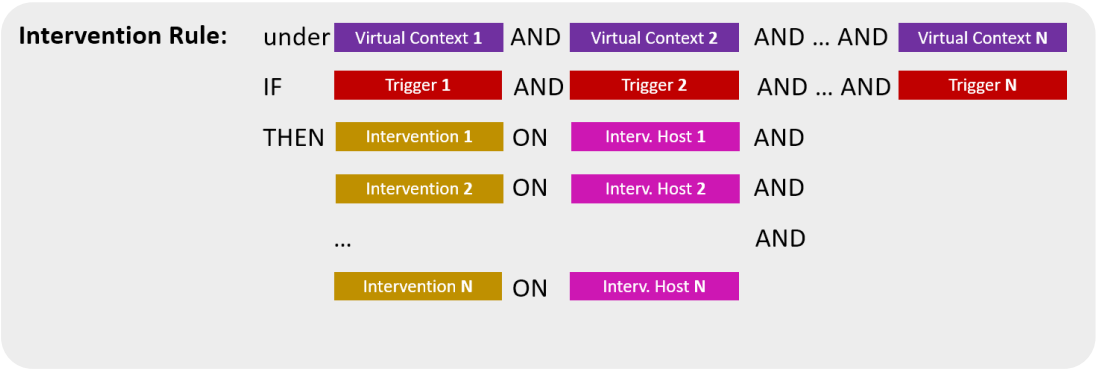

Figure 29: The conceptual representation of an intervention rule

Since a specific behavior can originate either from a single user or a group of people, the Intervention Manager is able to evaluate and select strategies targeting either private or public intervention hosts. Finally, after selecting the appropriate intervention(s), the system is able to personalize its (their) content according to the current context of use.

LECTORstudio enables developers to integrate new intervention strategies through the Intervention Pool Editor; that process includes defining (i) the scope of the intervention (i.e., private vs. public), (ii) the target user groups, (iii) the artifacts that can act as hosts, and (iv) the actual code that initializes and runs the respective application to each of the selected hosts. Such information permits LECTOR to make informed decisions regarding the appropriateness of a strategy under a specific context and initiate it when deemed necessary. Furthermore, LECTORstudio features the Intervention Rules Editor, which not only facilitate developers, but also permits non-technical users to tailor the intervention mechanism to their needs and preferences. In more detail, they can (i) define the context under which LECTOR intervenes, (ii) select one or more intervention strategies, (iii) configure them so as to better meet their needs (e.g., determine the video sources of a multimedia presentation), and (iv) at the same time customize other physical aspects of the intelligent environment (e.g., lights' intensity, blinds' status, etc.). The user interface of LECTORstudio is described in details in Chapter 6.

Upon creation, the intervention rules are stored in a database. Figure 30, presents the employed database schema; the key "triggers" is used to store information regarding the behaviors that trigger an intervention, while the key "context" refers to the virtual context under which the intervention rule should be evaluated. Information regarding the actual strategies that should be followed are described in "interventions" and "environment" keys. The introduction of the **Environment** as a separate form of interventions, despite the fact that there is no essential difference from an implementation perspective between environmental interventions and other software applications, aims to assist non-technical users in conceptualizing whether their decisions will affect other users of the environment.

```
var InterventionRuleSchema = new mongoose.Schema({
    name: String,
    triggers: [{ id: { type: String }, included: { type: Boolean } }]
    context: { },
    interventions: [{
        id: { type: mongoose.Schema.ObjectId },
        artefactId: { type: mongoose.Schema.ObjectId },
        attributes: [{
            name: String,
            attrType: String,
            Value: [],
            fromValue: String,
            toValue: String,
            Tags: [],
            defaultUnit: String,
            Function: String
        }]
    }],
    environment: [{
        artefactId: { type: mongoose.Schema.ObjectId },
        interventionId: { type: mongoose.Schema.ObjectId },
        attributes: [{
            name: String,
            attrType: String,
            Value: [],
            fromValue: String,
            toValue: String,
            Tags: [],
            defaultUnit: String,
            Function: String
        }]
    }],
    enabled: { type: Boolean, default: true },
    activations: { type: Number, default: 0 },
    success: { type: Number, default: 0 },
});
```

Figure 30: The database schema for the intervention rules

Furthermore, for each intervention rule the database stores statistics such as success rate, number of activations and number of cancelations. This type of information is important so as to evaluate the efficacy of a rule. Regarding the success rate, the Learning component is able to compare the behavior of the user before and a short time after the application of an intervention, so as to understand whether the remedial action was effective. For example, consider a student that lost interest in the task at hand, and chats with his classmate

instead of completing the multiple choice exercise which is launched on his personal computer. The system decides to intervene and motivate the student by displaying an encouraging message that prompts him to continue with the exercise. In case the user's behavior remains the same (i.e., the user is still in chatting) the success rate of that particular intervention decreases. Hence, in the future if a similar situation occurs the Intervention Manager can select an intervention with a higher success rate. Similarly, the number of manual cancelations can help the decision-making component understand whether the intervention is considered appropriate under specific context.

Before deployment, the stored data are retrieved and the relevant executable code is generated so as to be used by the decision-making component of the Intervention Manager. Similarly to the translation of the behavior models and trigger rules, the trigger rules are translated to javascript code using Handlebars.js.

LEARN: Learning Component

Both the Behavior Reasoner and the Intervention Manager should be able to "learn" from previous decisions and refine their logic, while they should also be open to user suggestions that can override their defaults. In order to introduce the notion of LEARN, the proposed system provides a mechanism that (i) is able to compare behaviors before and after interventions so as to understand whether the selected strategies were efficient, and (ii) takes into consideration user input so as to understand whether BR correctly identified behaviors or IM selected appropriate interventions (Figure 31).

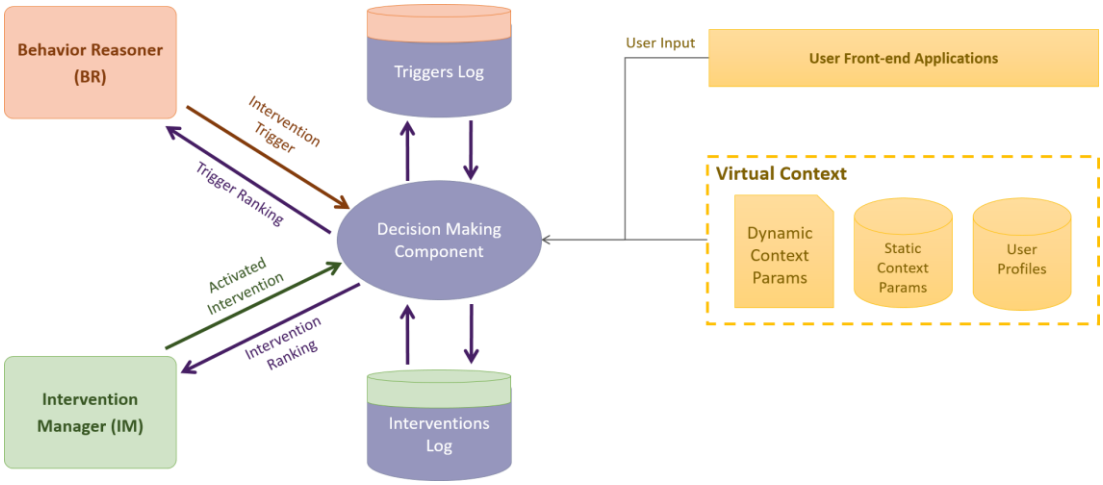

Figure 31: Learning Component

Triggers and Interventions History

The Learning Component (LC) keeps history logs of identified triggers and initiated interventions. When the IM applies an intervention estimated to be useful under a particular context of use, then after a reasonable amount of time (which is specified by the intervention rule) LC is responsible for re-examining whether the user's behavior remains the same. Specifically, if it still detects that -within a specific time frame- the Behavior Reasoner still identifies the same trigger, then the selected recommendation is marked as ineffective in that context (Figure 32). To do so, it informs the Intervention Manager to decrease the success rate of the rule that led to that particular intervention; from that point, IM can select different rules with greater efficiency for that

particular context instead of the one that was proven to be unsuccessful (Figure 33).

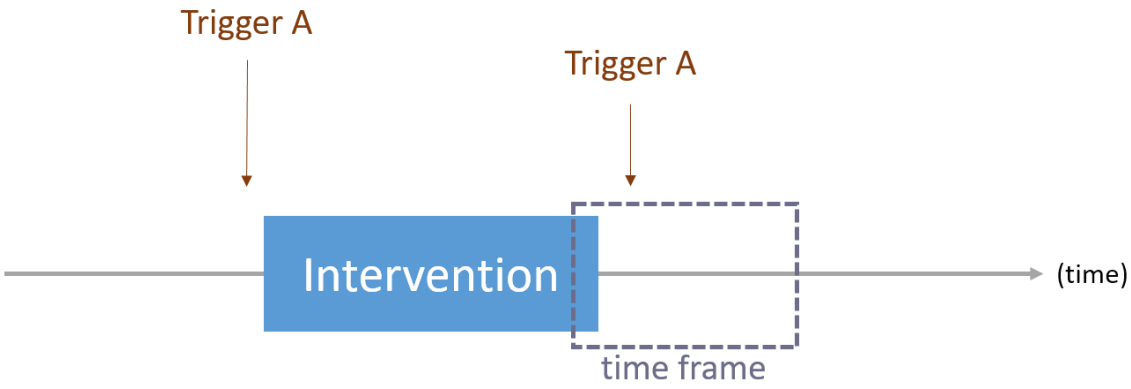

Figure 32: After the initiation of an intervention, if the same trigger is identified within a specific time frame, then that intervention is considered unsuccessful.

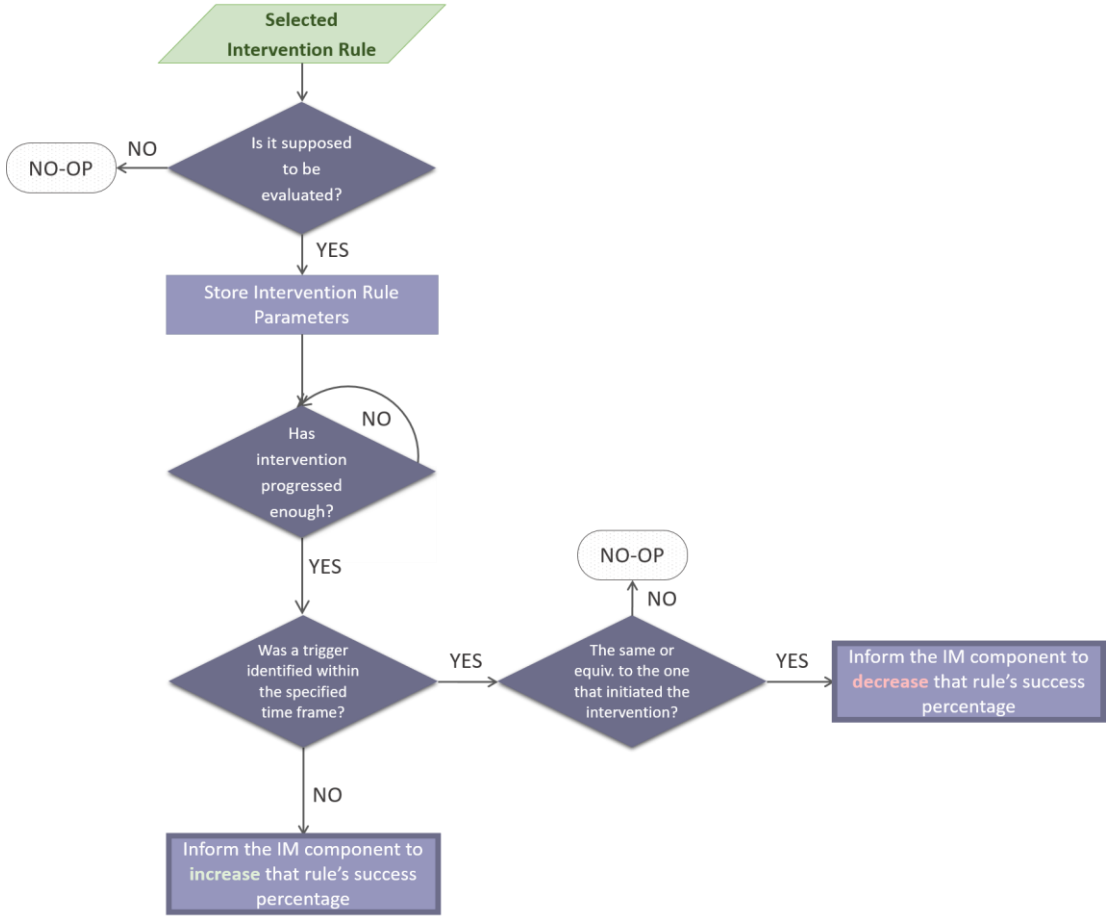

Figure 33: Determining the efficacy of an applied intervention

However, this approach does not apply to every intervention rule. For instance, consider an intervention programmed to turn off the classroom

lights as soon as a presentation starts on the board. In that case, it is irrelevant to examine whether "presentation start" is identified again after the lights turn off. To this end, each intervention rule is appropriately annotated so that the LEARN Component can decide whether it should start its evaluation process or not.

Apart from studying the history log of behaviors immediately after a remedial action has been taken, in some cases it is important to track the long-term effects of interventions. This approach has the advantage of ignoring false positives, such as "fake" behaviors, where students pretend to be engaged to an educational activity, when in fact they are not. Instead, the effectiveness of an intervention can be verified as soon as the system validates that the student has acquired the knowledge taught during the period that the intervention in question was applied (i.e., that could occur much later in time e.g., when that student scores exceptionally well in a relevant exam). To this end, the Learning Component is able to combine performance statistics with past contextual information to identify whether the applied interventions had positive results.

User Input

One of LECTOR's main objectives is to help and support teachers by enabling complete classroom overview and automatic suggestion of engaging activities. Towards that direction, teachers can provide valuable input, and namely they can (i) reject system decisions regarding identified triggers, and (ii) override system suggestions in case they do not serve their needs. Firstly, the users will be able to notify the system in case it has falsely identified an intervention trigger. For example, in the case of a student that stares at the ceiling, the system wrongly estimates that he is mind wandering and triggers the rule that initiates a multiple-choice quiz to re-motivate him. At this point the teacher, knowing that this particular student was "thinking" while staring at the ceiling, notifies LECTOR that he was not being inattentive; hence, the Behavior Reasoner will to incorporate that knowledge for future reference. Secondly, in

order to allow the teachers to have the final say regarding the suggested interventions, the system permits their ranking, modifications and cancellations. To this end, the LEARN Component is responsible for informing the Intervention Manager to update the statistics of each intervention rule.

Chapter 6
LECTOR Tools

This chapter describes the functionality and user interface of LECTORstudio [106], a sophisticated authoring tool that permits both developers and non-technical users to take advantage of LECTOR's services. Furthermore, LECTORstudio permits developers create the appropriate infrastructure depending on the target audience and the available ambient facilities.

Additionally, LECTOR interoperates with three (3) other powerful tools which are specifically created to serve the needs of educators and students of the intelligent classroom. These tools are explained in details in the following sections.

LECTORstudio: Creating Inattention Triggers and Planning Interventions

LECTORstudio aims to assist developers and educators in the process of realizing an attention-aware intelligent classroom that intervenes when necessary to re-motivate students.

To this end, it offers an intuitive UI through which developers can:

(i) create the classification scheme for organizing the rules responsible for detecting behaviors,

(ii) integrate the available artifacts that can host interventions,

(iii) model the physical context which encapsulates information regarding physically observable phenomena (e.g., luminance, heart rate, sound levels, etc.),

(iv) describe the actors, i.e., the users whose behavior needs to be monitored in order to conclude whether an intervention is required,

(v) integrate the intervention types that are available through the existing hosts.

Furthermore, it permits both developers and non-technical users to:

(vi) create rules that reveal intervention triggers,

(vii) fine-tune the rules responsible for identifying behaviors in the monitored environment,

(viii) create rules according to which specific interventions are initiated.

Any of the above tasks is a complex procedure that requires the configuration of many heterogeneous parameters, which could be overwhelming and cumbersome even for experienced developers, let alone non-technical users. To this end, LECTORstudio features a wizard-style interface in order to transform this complex set of conditions into understandable steps, which is ideal for users that lack the necessary technical knowledge since they can just follow a preplanned route [107].

In more detail, each of the aforementioned tasks is decomposed into autonomous chunks, which form logical step-by-step processes to guide the users towards achieving their goals; to avoid creating tiring and tedious processes, the number of chunks is kept to a minimum, leading to a maximum of 3 or 4 steps. Specifically for the tools that target educators, LECTORstudio reduces complexity by avoiding the use of technical terms and instead speaking the users' language using words and phrases familiar to teachers. Finally, adding up to the clear and informative UI, input controls were carefully selected in order to make sure that any user input is well-formed according to the functional requirements of the system (e.g., expected types,

predefined values, etc.). These practices, along with the adoption of general HCI design principles (e.g., consistency, user control and freedom, error prevention, etc.), ensure that LECTORstudio reduces, to a large extent, the cognitive load and simplifies the overall process for both developers and educators.

Creating the Behavior Classification Scheme

A two-level classification scheme is employed for organizing the rules responsible for detecting behaviors. The first level contains the abstract behavior types (e.g., Attention, Focus, Speech, Location, etc.) that should be monitored by LECTOR in order to collect valuable information. Yet, the concrete behavior outcomes (e.g., Quiet, Whispering, Talking, etc.), are described at the second level and contain the respective rules. The same outcome might be recognized by several rules; that is because employing different algorithms or monitoring miscellaneous contextual parameters (i.e., physical, or virtual context) can lead to the same result.

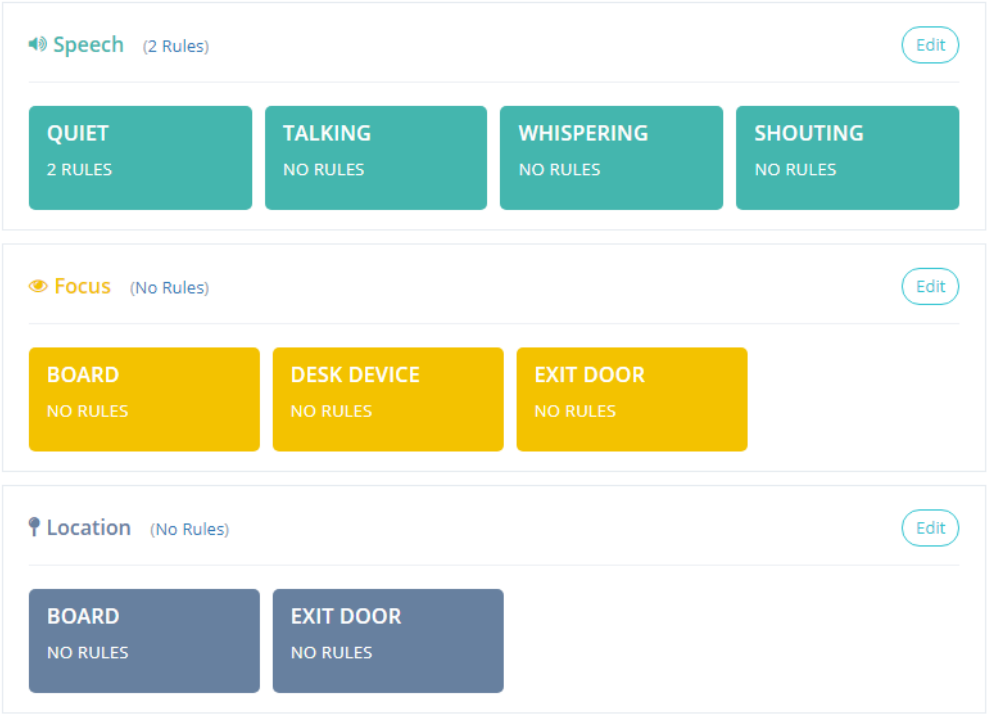

Figure 34: Snapshot of LECTORviewer's UI displaying a subset of the available behavior types along with their outcomes

Figure 34, presents LECTORviewer's user interface displaying a subset of the available behavior types along with their outcomes. Notably, the selected classification scheme leads to an intuitive UI that permits both developers and educators to find rules quickly on the basis of the behavior type and outcome. As shown, a specific theme is followed per behavior type in order to support their recognizability and enhance intuitive interaction. LECTORstudio permits developers to create the aforementioned scheme (Figure 35) by providing the following:

- A descriptive **name** of the behavior type.
- **Presentation details** such as description, icon and theme (i.e., font and background color), which will be used to create an intuitive UI.
- The **behavior outcomes** that also constitute the labels to be employed in LECTOR's reasoning mechanisms.

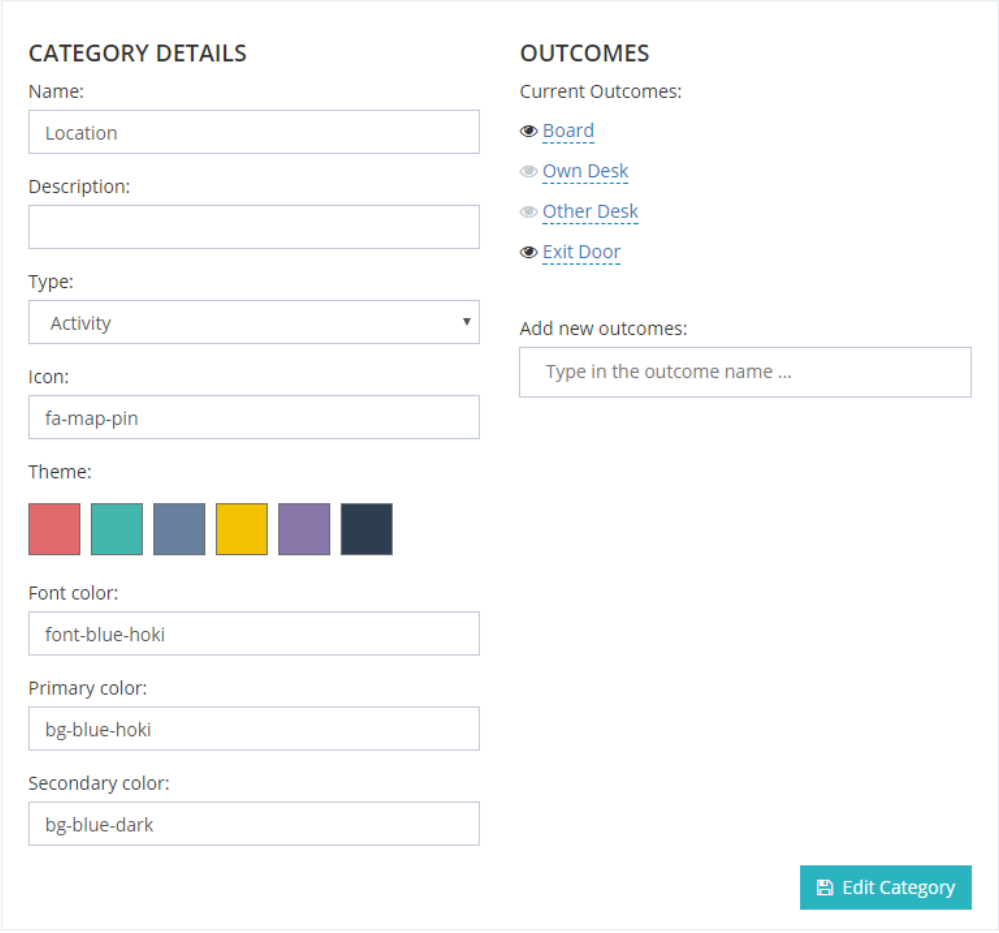

Figure 35: The process of creating the Behavior Classification Scheme

As soon as an outcome is defined, the developer can set its lock status to ON (default) or OFF; a locked outcome is not available to educators for adding or modifying the relevant rules. For example, the "OWN DESK" outcome of the behavior type "LOCATION", should be locked since only experienced developers can provide the appropriate code that identifies whether a student sits on her desk. On the other hand, the "BOARD" outcome of the same type can remain unlocked, since the educator can fine-tune any rules concerning whether an individual's whereabouts are near the board by manually selecting its position on the classroom's floor plan.

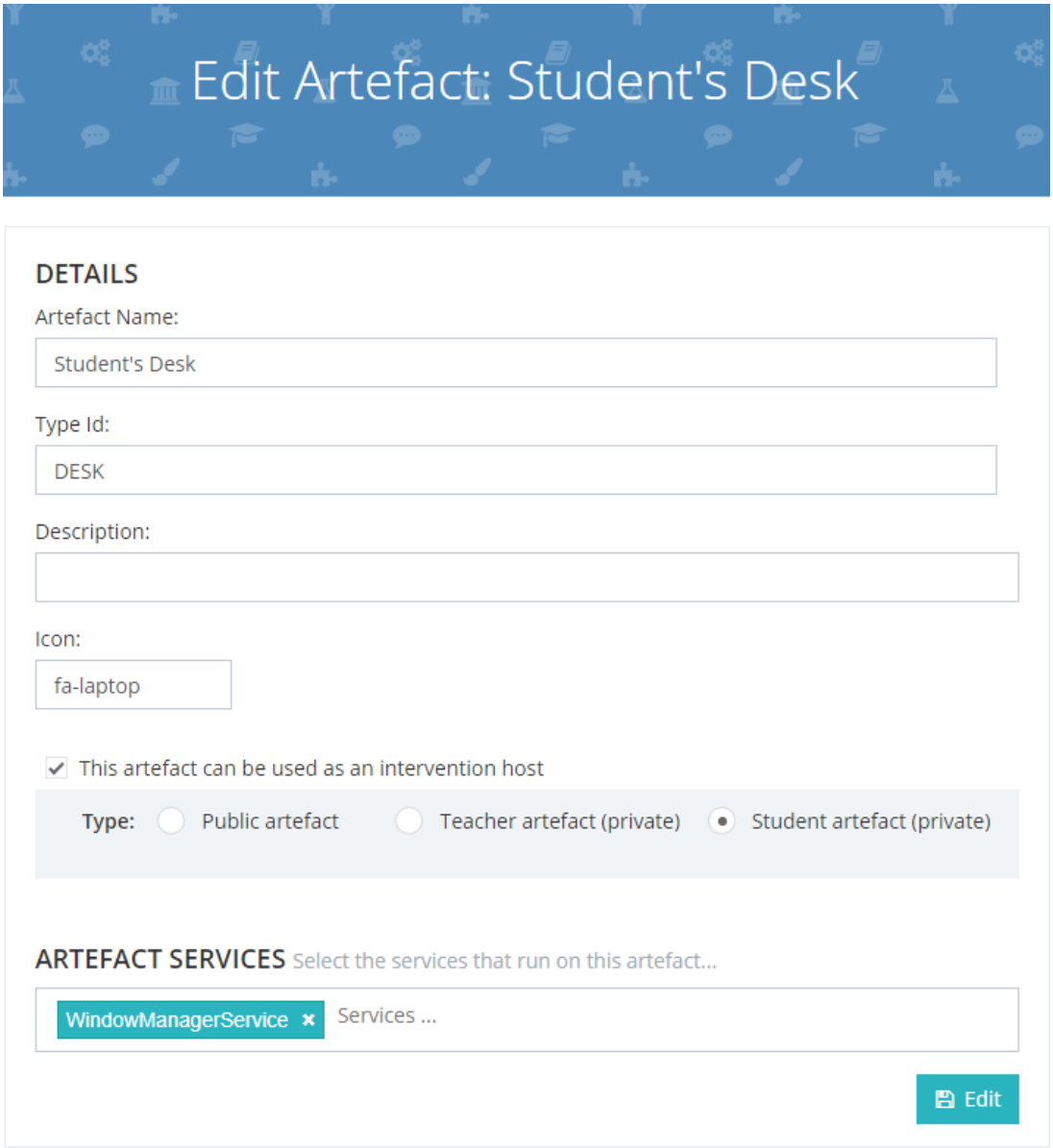

Figure 36: The process of editing the integrated intervention host "Student's Desk"

Integrating the available Intervention Hosts

The term intervention host is used here to describe (i) common computing devices such as smartphones, tablets, and laptops, (ii) technologically augmented everyday commercial objects (e.g., interactive white boards, smart bulbs, etc.), or (iii) custom made items (e.g., student desk). An Intervention host either launches an application with specific content or controls the physical environment (e.g., dim the lights during a video presentation). For LECTOR to optimally intervene, the available intervention host types have to be properly defined. To this end, LECTORstudio allows the modeling of each type in a way that conveys the information required for creating and deploying an intervention (Figure 37).

Figure 37: The available intervention hosts are displayed in the form of tiles containing useful information

Specifically, developers are expected to define the following (Figure 36):

- **General details** about the intervention host, i.e., name, description and icon.
- A **unique ID**, which will be used by intervention hosts to advertise themselves in AmI-Solertis. This information enables LECTOR to

communicate with intervention hosts of the same kind in a seamless manner. For example, when an intervention is scheduled to be applied on every student desk, LECTOR simply asks AmI-Solertis to deliver the command to all hosts identified as “DESK”.

- The host-specific **services** that should be exposed to developers responsible for building and deploying interventions on that particular host. For example, a student desk employs various services such as CognitOS, Microphone Recorder, Seat tracker, Sound Controller, etc.; subsequently, if CognitOS and Sound Controller services are exposed, a developer can define an intervention for a student that continuously browses out-of-context videos using the former to launch an educational application (e.g., Quiz) and the latter to mute the speaker output.
- The **intervention host type**, which determines whether it can be used as an intervention host or as an intelligent device that can be used to modify the conditions of the environment (e.g., lights, blinds, curtains, etc.).
- The **privacy type** which could be: (i) public intervention host, (ii) private teacher intervention host or (iii) private student intervention host. This information is intended to be used by LECTOR’s Intervention Manager to identify appropriate hosts for the planned interventions.

Modeling the Physical Context

LECTORstudio facilitates the modeling of the physical context (Figure 38), which encapsulates information regarding physically observable phenomena (e.g., luminance, heart rate, sound levels, etc.). This aims to alleviate the diversity of input values coming from heterogeneous sources and create a shared data model that can be used by the rule creation mechanisms.

In more detail, developers can define the type of information that LECTOR will be able to process, independently of its source. For instance, the system does not need to know that a student has turned his head 23 degrees towards south but that he stares out of the window.

The first step of creating a physical context property requires:

- A descriptive **name**.
- The relevant **services** that deliver information collected from the input sources. For example, a service exposed by the student's wrist band is able to transmit data regarding her physiological signals.
- The property's **variables**, which are the available fields that will carry the actual property value at runtime. The definition of each variable requires its name, type (e.g., Number, String, Boolean), default value and available values, as well as default unit and other available units.

The second step of this process includes the translation of the data types received from the sensors to the types described by the property's variables. This process requires developers to provide the code that performs the mapping from one type to the other. To do so, LECTORstudio employs the sophisticated web-editor offered by AmI-Solertis, which automatically generates some of the required blocks of code based on information acquired from the first step. This alleviates some effort from the developers, while along with its built-in autocomplete functionality the editor provides several instructions (in the form of commented code) that guide developers in building their code.

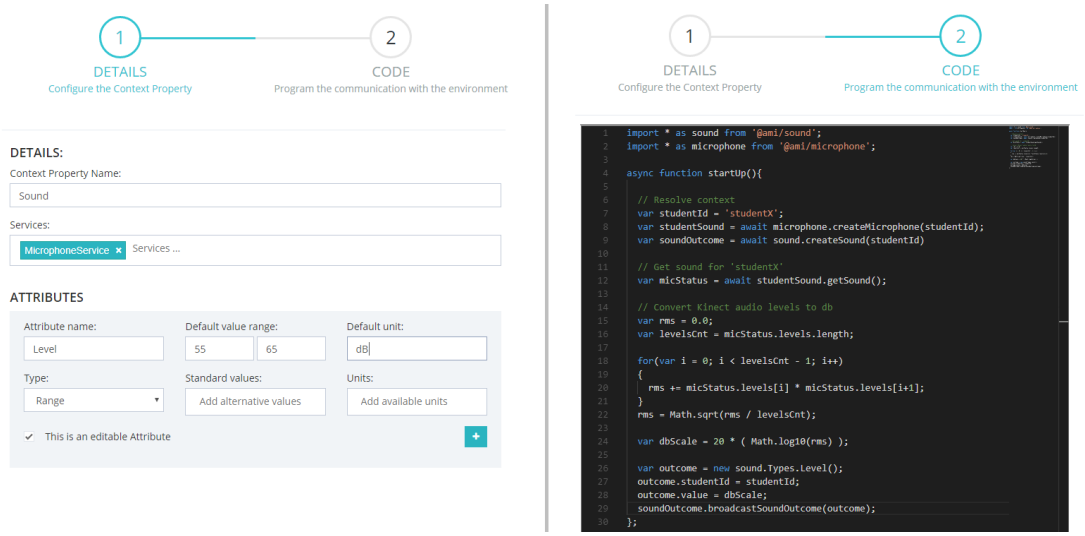

Figure 38: The process of defining the 'SOUND' Physical Context Property

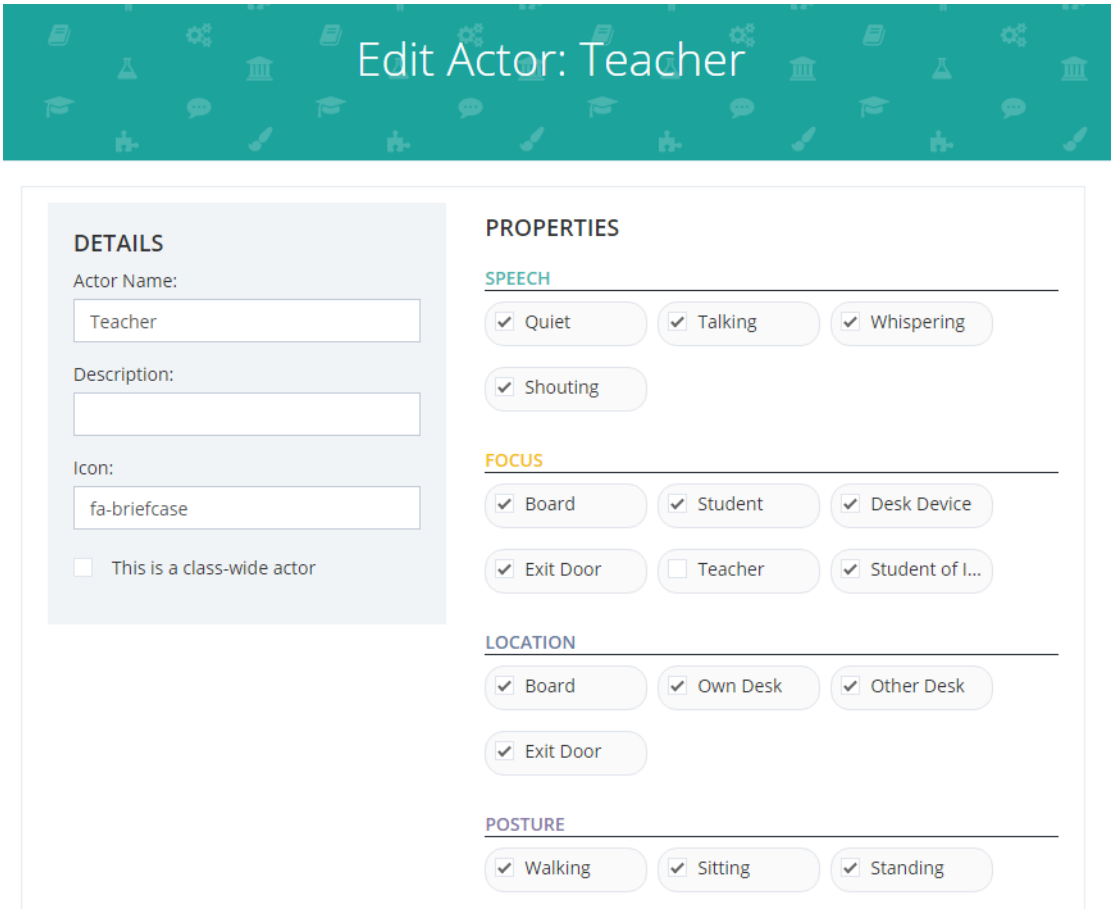

Figure 39: The process of editing the actor "Teacher"

Describing the Actors

The term Actor is used to describe the users of the intelligent environment where LECTOR is employed, whose behavior needs to be monitored in order to decide whether an intervention is required. Generally, different types of actors have diverse characteristics which need to be taken into consideration when building the rules that guide LECTOR's decision-making mechanisms. To this end, LECTORstudio permits developers to insert the following information (Figure 39):

- **General details** regarding the actor, i.e., name, description and icon.
- The **actor type** (i.e., individual or group), which determines whether the system's decisions must rely on the actions of individual users or the collective behavior of a group of people. Indicative actor examples, drawn

from the intelligent classroom case study, are: (i) Teacher (individual), (ii) Student (individual), and (iii) Classroom (group of students).

- The **outcomes** that designate which actor behaviors (i.e., virtual properties) should be monitored and which should be excluded. Narrowing down the virtual properties that need monitoring not only simplifies the process of rule creation, since the interface is not cluttered with irrelevant options, but also improves the system's runtime performance as unnecessary checks are eliminated. For example, the FOCUS of a student might be the "BOARD", another "STUDENT" or the "TEACHER"; the first two virtual properties are legitimate values for the teacher as well. However, the latter is a behavior that will never occur.

Figure 40: The available intervention types are displayed in the form of tiles containing useful information

Integrating Intervention Types

As already mentioned in Chapter 5, the term intervention is used to describe the system-guided actions that subtly interrupt a course's flow in order to (i) draw the educator's attention on problematic situations, and (ii) re-engage distracted, unmotivated or tired students in the educational process. Table 2, displays the available interventions, which in fact are applications running on private (e.g., student's desk, teacher's watch) or public intervention hosts (e.g., classroom board) instantiated at a key point in time with appropriate content. In order to act as an intervention, an application is required to conform to AmI-Solertis SaaS specifications, ensuring that it will be able to receive and execute LECTOR's commands. At the same time, LECTOR needs to be aware of the available intervention types; for that purpose, LECTORstudio offers a

wizard-style interface that permits developers to integrate them in three simple steps.

The first step (Figure 41) requires the general details of the intervention, i.e., name and description, as well as the definition of its customizable properties. A customizable property is an attribute exposed to educators for configuration. For example, when creating an intervention rule that utilizes the "Multimedia presentation" application, the educator can select which of the available video sources (e.g., YouTube, Dailymotion, etc.) will be used.

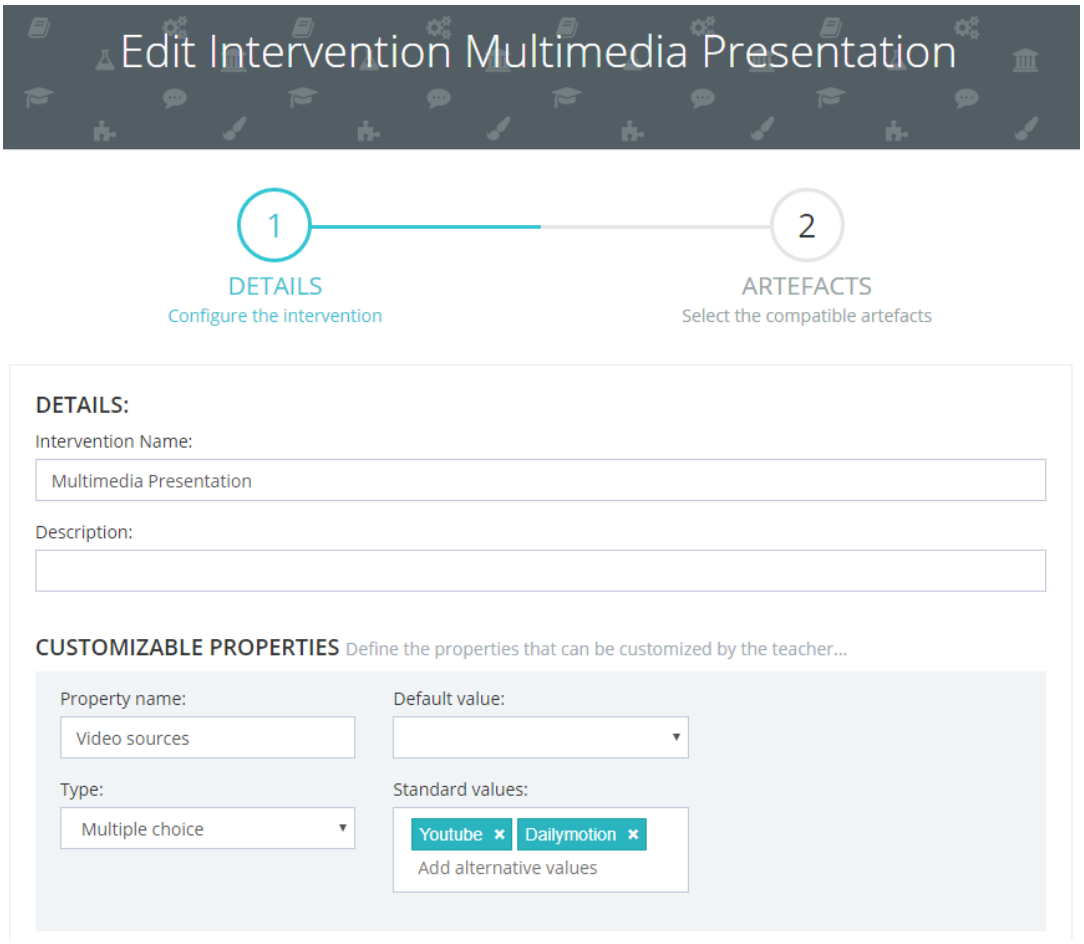

Figure 41: The first step of editing the intervention type "Multimedia Presentation". This step requires the definition of the details of that particular intervention type.

As a second step (Figure 42) developers are expected to define the target (i.e., student, teacher, and environment) of the intervention type as well the artifacts that are able to employ it. In case the targeted users are students, the range (i.e., public or private) of the intervention has to be defined too. Being

aware of the actual recipients and the range is essential for LECTOR's Intervention Manager, which among others is responsible for selecting the appropriate intervention type and suitable host(s). For example, if an individual student needs motivation, a private intervention is appropriate; on the contrary, if the entire classroom requires a remedial action, LECTOR can either instantiate a public intervention at a public host or apply collectively private interventions to all private student hosts.

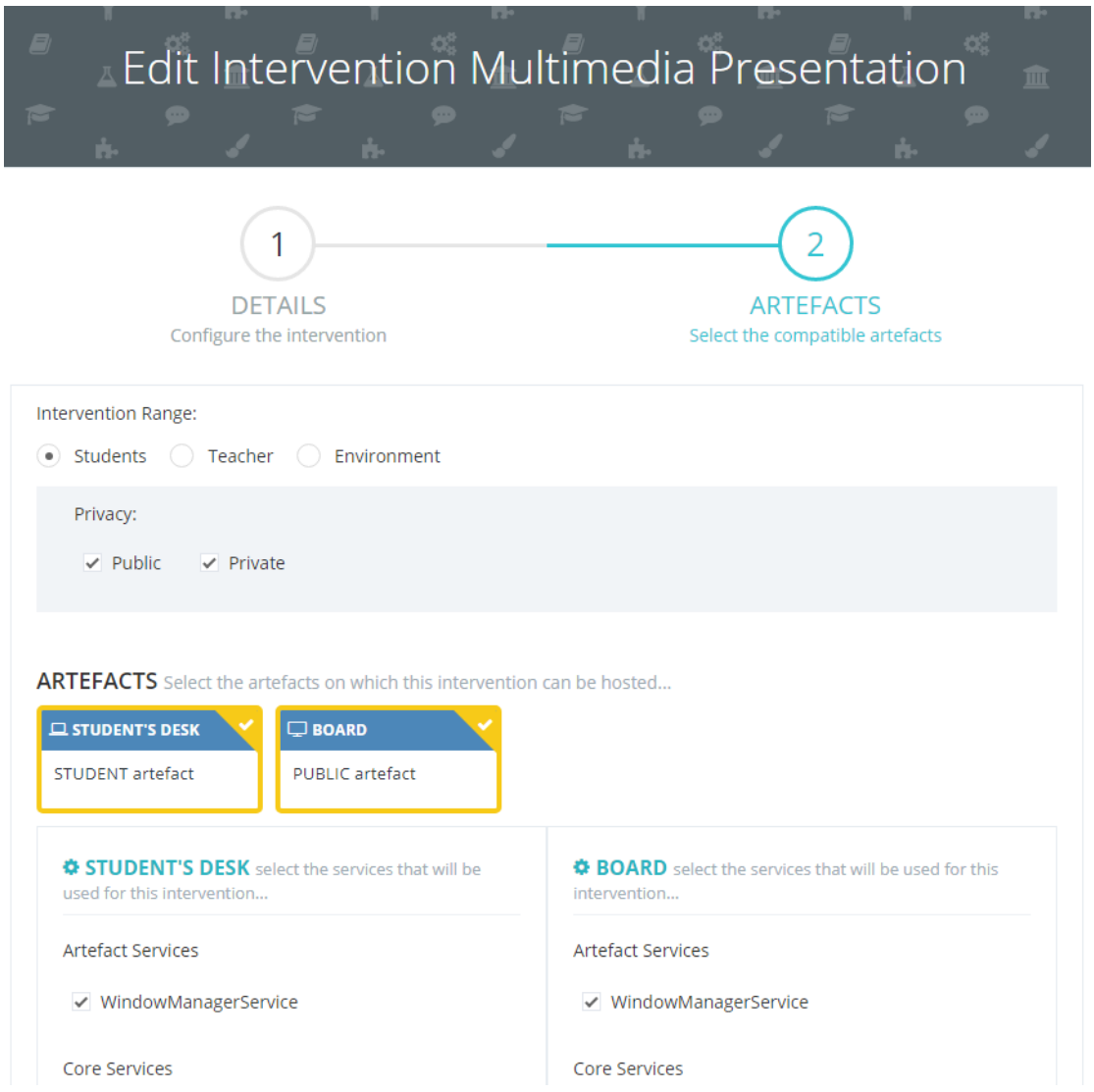

Figure 42. The second step of editing the intervention type "Multimedia Presentation". This step requires the definition of the artifacts that can host that particular intervention type.

As soon as the intervention recipient is defined, the available intervention hosts are filtered appropriately narrowing down the acceptable options. At this point, the developer must select the host(s) that contain implementations

for that intervention type, while at the same time she must define the host-specific service(s) responsible for instantiating that intervention. For example, the "Mini Quiz" intervention can be applied either privately on a student's desk (i.e., CognitOS_App_Launcher service) or publicly on the classroom board (i.e., AmIBoard_App_Launcher service).

Finally, for each of the selected intervention hosts the developer has to provide the actual code that initializes and runs the respective application. To do so, all the collected information is forwarded to the AmI-Solertis web-based editor, which prepares the ground for inserting the necessary lines of code. Specifically, the editor initializes the intervention and the selected services per host.

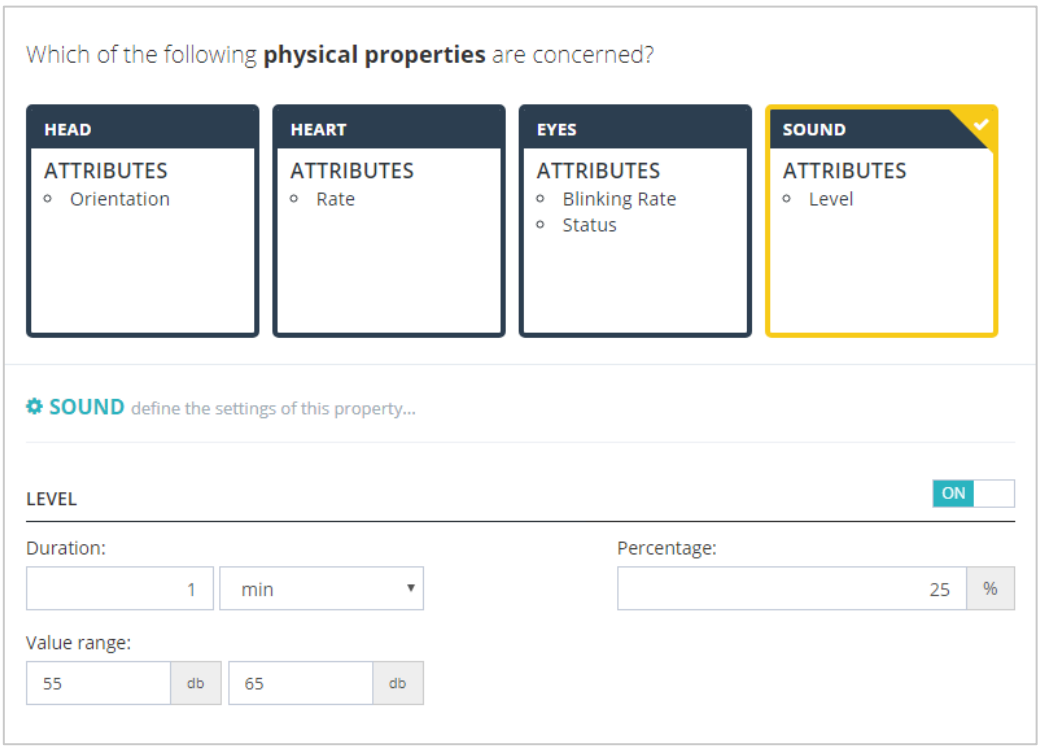

Figure 43: The step of selecting and configuring the physical properties that need fine-tuning.

Fine-tuning Behavior Recognition Rules

LECTORstudio permits the modelling of human, agent and artifact behaviors. This is achieved through "if-then" rules describing the conditions under which specific behaviors occur. As soon as a behavior is detected, the decision-

making mechanisms of LECTOR further examine the "Trigger rules" set, so as to identify whether at that particular context the identified behavior indicates inattention. Creating a behavior recognition rule from scratch would require excellent programming skills, implementation of specific scientific algorithms and accurate knowledge on what data are being collected by each technological artifact. To this end, this functionality is deliberately excluded from the version of LECTORstudio that targets educators.

However, fine-tuning those rules and being able to configure some of their high-level attributes is available through a 3-step wizard. The first step requires the definition of the context following a process similar to the one described for the "context definition" step of the "Create inattention triggers" wizard. Next, the educators are expected to select any physical properties they wish to configure. Upon selection, a panel with configurable attributes enables them to fine-tune the conditions under which a (sub) – activity is denoted (Figure 43). Finally, the wizard provides a summative overview and the educator can create and deploy the new rule.

The necessity for the described fine-tuning facility is illustrated by the following motivating example. The default rule for the activity **SHOUTING** – which belongs to the Speech category– specifies that "*if an actor's sound levels range between 85 and 95 decibels, then the actor is shouting*". Since this rule is the default one, it applies to every context and triggers as soon as the THINK component of LECTOR receives increased sound levels for a specific amount of time. However, an exception seems imperative for the Music course where students are expected to sing, thus raising the noise levels of the classroom higher than usual. Through LECTORstudio, educators can easily create a variant of that rule, where only during the music course (i.e., modify a contextual property) the threshold of the SOUND physical property that triggers the rule will overpass 100 db.

Creating Inattention Triggers

LECTORstudio enables the enrichment of the "Trigger Rules" set by permitting developers and educators to easily create and modify rules that signify inattention via the respective creation wizard. To fully specify such a rule, an user is required firstly to define its conditions, namely (i) the context under which the rule should be evaluated, (ii) the implicated actors (e.g., individual or group of students, teacher, etc.), (iii) the physical attributes that should be monitored for relevant cues, and secondly determine the type of inattention that this rule can detect (e.g., disturbing classroom, cheating, chatting, etc.). Defining an inattention type not only helps the educators understand the source of the problem when an alarm is triggered, but is also used by LECTOR's Intervention Manager to decide which intervention should be applied. An indicative example of such an "Inattention Trigger" is the following: "*During a lecture, if the teacher is talking to the students and some of them are too talkative or do not focus on her, then this is a sign that they are possibly chatting*". This sentence contains all the information needed to build the respective rule: "during a lecture" implies the context, the implicated actors are the teacher and the students, whose sound levels should be examined to identify whether they are talking, while the students' focus should also be under inspection. Finally, the closing words indicate that "chatting" is the inattention type of such behavior.

Figure 44, presents all the necessary steps to define a new rule that aims to detect inattention. Firstly, the user is required to define the **context** under which the newly created rule will be evaluated. Three affirmative statements –one for each supported context type– are displayed, prompting the user to select whether she agrees with them or not: (i) It (the rule) concerns specific **courses**, (ii) It (the rule) concerns specific **activities** and (iii) It (the rule) concerns specific **student groups**. Each statement is accompanied by a simple yes/no input control with the default value set to "No". As soon as that value changes to "Yes", a set of relevant options appear representing the expected

values for that context, out of which the user can select which will be included or excluded from the rule.

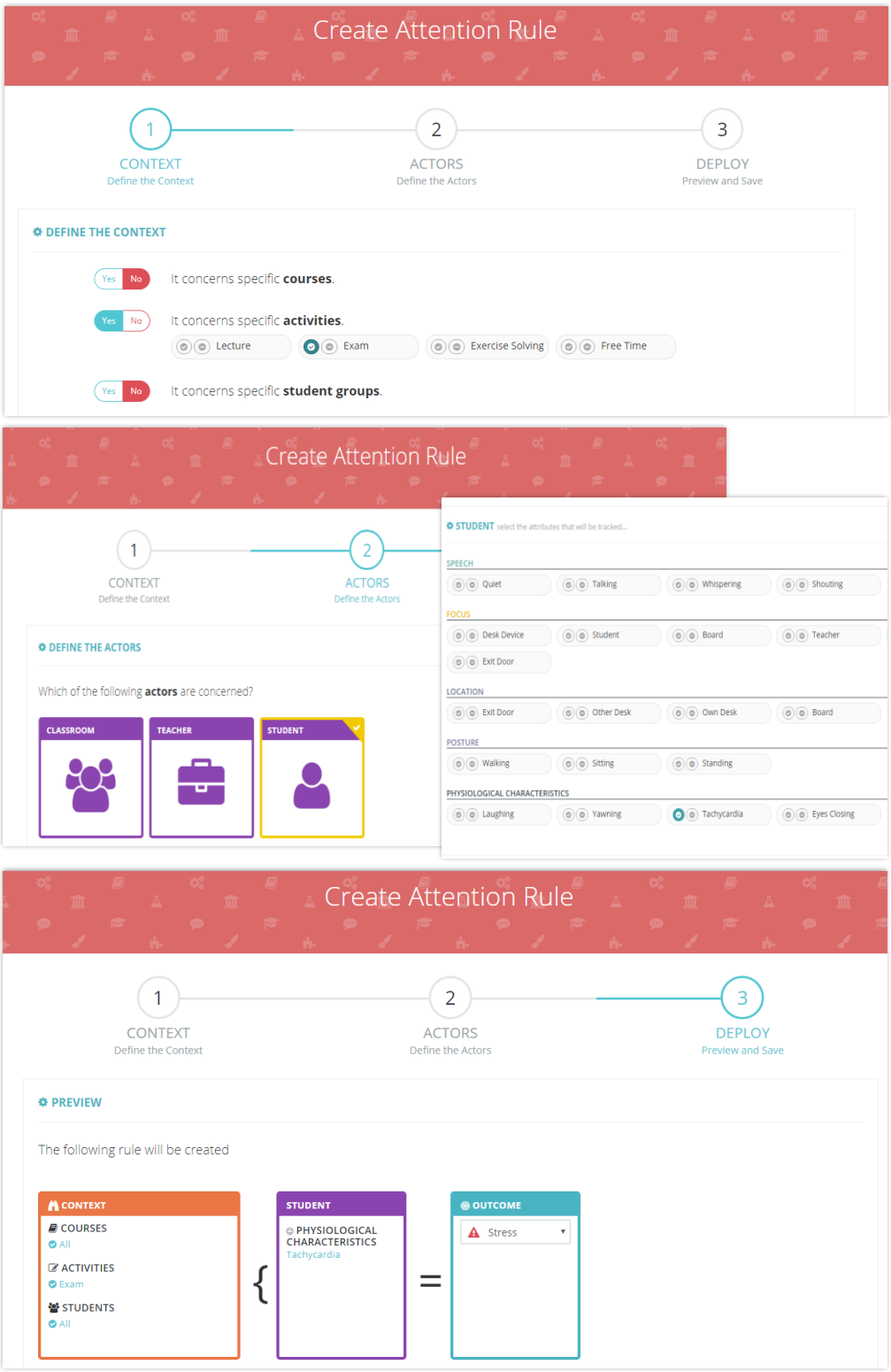

Figure 44: The process of creating an inattention trigger

Apart from the context under which an educational activity takes place, a rule responsible for triggering inattention alarms requires information regarding the **implicated actors** as well as their **specific physical** properties that should be under inspection; the available actors inside a classroom are either teachers or students. In order to observe the way they communicate and interact during a course, LECTOR currently monitors some of their physical characteristics (i.e., physical context) and translates them, in a context-dependent manner, into specific activities classified under the following categories: Focus, Speech, Location, Posture and Feelings, which are considered appropriate cues that might signify inattention (Chapter 5).

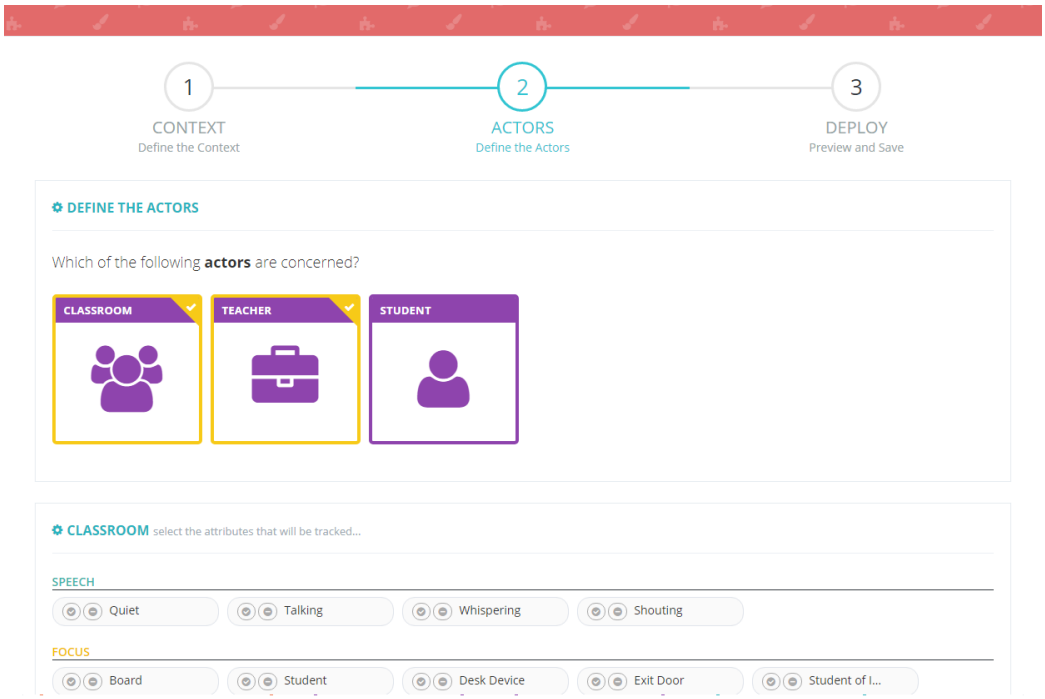

Figure 45: The second step of the "Create inattention triggers" wizard requires the selection of the implicated actors

As soon as context is properly defined, the “Create inattention triggers” wizard (Figure 45) presents to the user a list of the available actors in the form of selectable tiles. When an actor is selected, a panel with his/her relevant physical properties is displayed, giving the educator the opportunity to define which of them will be included in the rule’s activation condition; the values could be used as-is (e.g., focusing at the teacher) or as logical negations of their initial value (e.g., not focusing at the teacher). Additionally, the user can

customize the rule by modifying quantitative attributes such as the threshold after which that behavior is consider inattention or the cumulative percentage of students not paying attention after which an inattention alarm should be triggered.

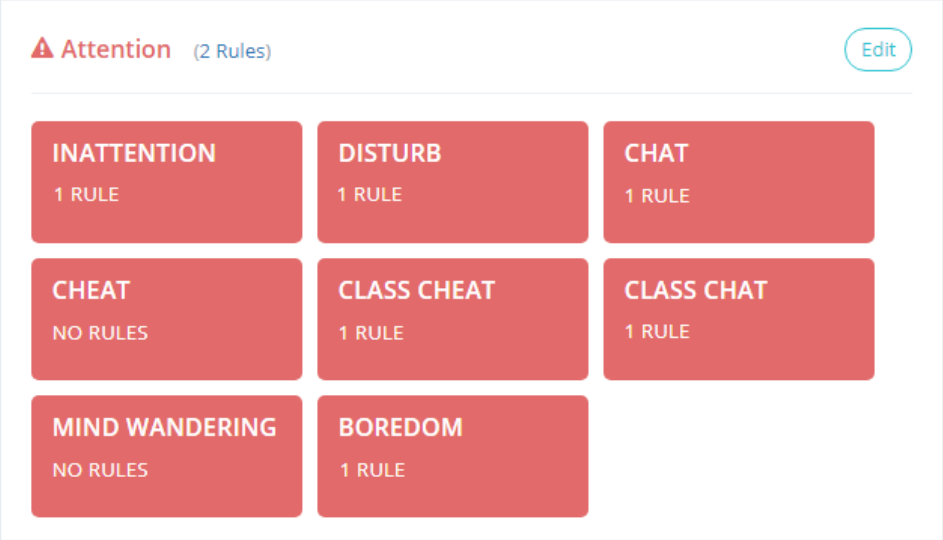

Figure 46: Snapshot of the UI displaying the available inattention Triggers

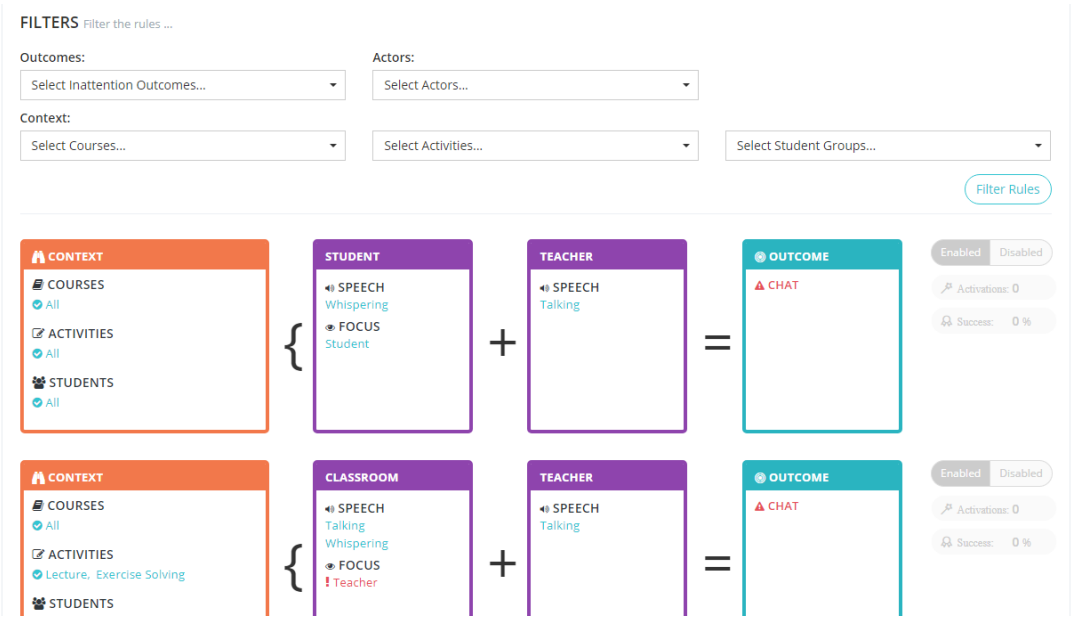

Figure 47: List of rules that indicate the trigger "Chat"

Finally, the user can use the wizard to “DEPLOY” the newly created rule. An intermediate confirmation step permits the user to preview the rule’s details, revise her selections, and if necessary even return to previous steps to make

adjustments. The discrete parts of the rule (i.e., context, actors and outcome) are represented in the form of tiles containing key information regarding their configured aspects, while simple mathematical symbols (e.g., plus sign, equals sign, braces, etc.) denote its structure. As a last step, the user must determine what kind of inattentive behaviour this rule can detect (e.g., disturbing classroom, cheating, chatting, etc.). A finalized rule is immediately activated and gets incorporated into LECTOR's "Trigger Rules" set (Figure 47); nevertheless, educators can manually deactivate or re-activate rules when deemed necessary. In order to classify the rules and facilitate browsing, LECTORstudio presents the available triggers in the form of tiles, where each tile redirects the user to an appropriately filtered list of rules (Figure 46).

Scheduling Interventions

Finally, through LECTORstudio, developers and educators can: (i) define the context under which LECTOR intervenes, (ii) select one or more intervention strategies and configure them so as to better meet their needs, and (iii) at the same time customize other physical aspects of the intelligent environment (e.g., lights' intensity, blinds' status, etc.) so as to create the optimal conditions for re-motivating distracted students (Figure 48).

An indicative intervention rule that can be created through LECTORstudio is the following: "*if students are chatting during a lecture, then launch a multimedia presentation (drawing content from YouTube and Google Images) on every student's desk, dim the lights to 50% and half-close the blinds*". This sentence specifies all the available information for building the respective intervention rule. Specifically, the foreseen intervention will be applied only in case the detected inattention type is "Chatting" and the educational activity is a "Lecture" (i.e., context). As regards the actions to be taken, the intervention will initiate a customized (from the educator) multimedia presentation (i.e., type of intervention) on each desk (i.e., the presentation artefact). Finally, certain aspects of the environment will be modified to better serve the selected intervention.

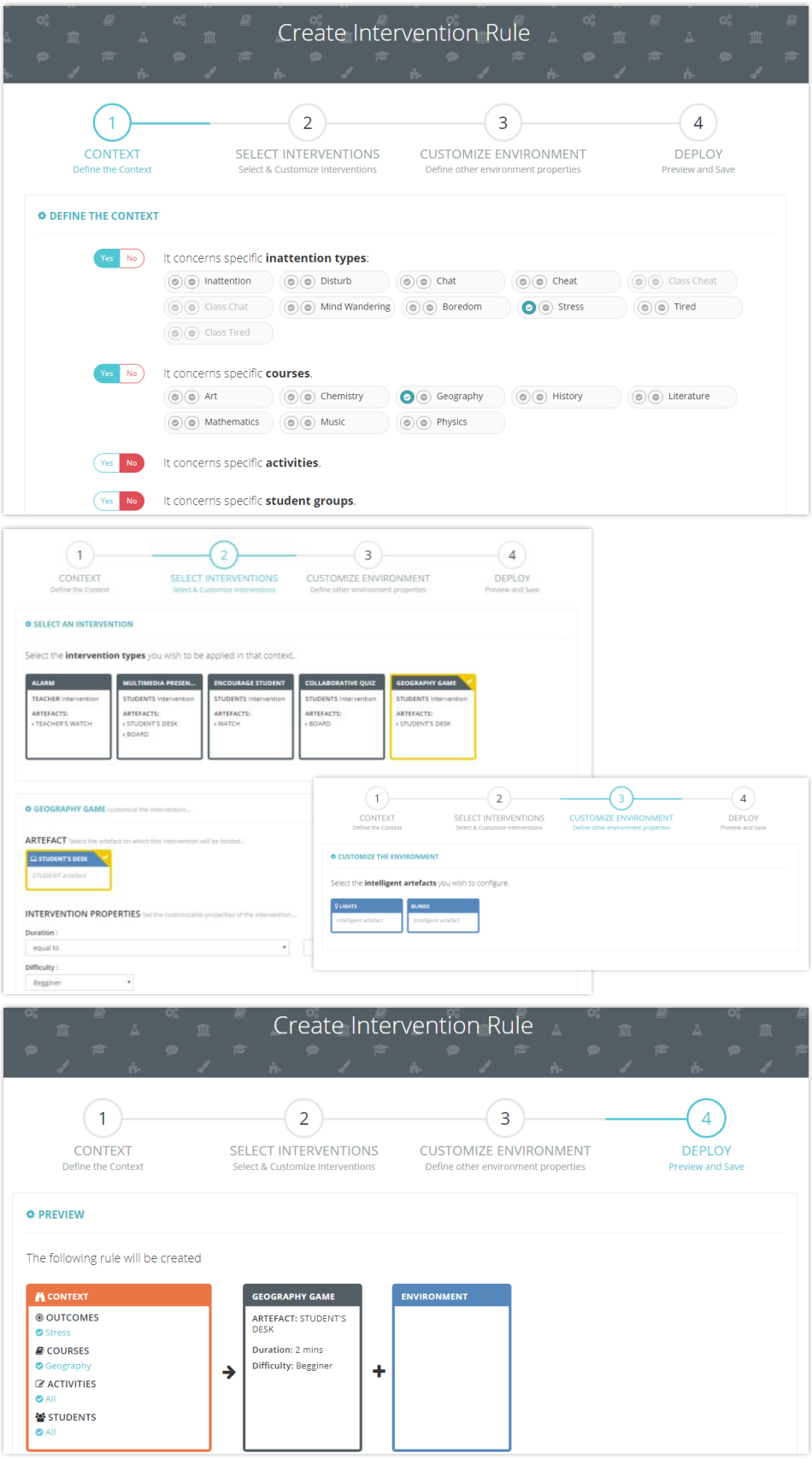

Figure 48: The process of scheduling an intervention.

An indicative intervention rule that can be created through LECTORstudio is the following: "*if students are chatting during a lecture, then launch a multimedia presentation (drawing content from YouTube and Google Images) on every student's desk, dim the lights to 50% and half-close the blinds*". This sentence specifies all the available information for building the respective intervention rule. Specifically, the foreseen intervention will be applied only in case the detected inattention type is "Chatting" and the educational activity is a "Lecture" (i.e., context). As regards the actions to be taken, the intervention will initiate a customized (from the educator) multimedia presentation (i.e., type of intervention) on each desk (i.e., the presentation artefact). Finally, certain aspects of the environment will be modified to better serve the selected intervention.

Towards defining the context, the user firstly has to specify the situation(s) under which the system should intervene. To that end, a list of **inattention types** (e.g., Cheating, Chatting, Disturbing a classmate, etc.) that can be detected is presented along with relevant statistics (e.g., frequency, amount of rules that can identify it, etc.). Then, she can further scope that intervention by customizing the courses, activities and student groups similarly to the other LECTORstudio wizards.

Next, the wizard presents the available interventions in the form of tiles, where each tile displays both the intervention recipient (i.e., teacher or student) and the intelligent artefacts that can host it. Upon selection, a panel with customizable properties appears, enabling the user to select the presentation artefact and configure the intervention according to her needs. Multiple intervention types can be selected at once, giving the opportunity to create complex scenarios.

Finally, the user can optionally customize the physical aspects of the intelligent environment (e.g., lights' intensity, blinds, etc.), confirm the various parameters and deploy the new rule.

Heuristic Evaluation

The wizards targeting both developers and educators were evaluated using the heuristic evaluation method in order to eliminate any major usability errors before proceeding with user testing. Heuristic evaluation [108] is the most popular of the usability inspection methods and is carried out as a systematic inspection of a user interface design for usability. The process involves having a small set of evaluators examine the interface and judge its compliance with recognized usability principles, namely "heuristics". According to [109], involving three to five evaluators, is adequate to identify the majority of errors since larger numbers do not provide much additional information. The process requires that each individual evaluator inspects the interface alone and compares it with the "heuristics". As soon as all the evaluators have completed the aforementioned process, the discovered usability errors are aggregated in a list with references to those usability principles that were violated. Next, each evaluator is asked to provide severity ratings [110], ranging from zero (“not a usability problem”) to four (“Usability catastrophe”), for each problem independently of the other evaluators. Finally, the development team ranks each problem with an ease-of-fix ranking ranging from zero (“would be extremely easy to fix”) to three (“would be difficult to fix”) to designate the amount of effort needed to address it. LECTORstudio was evaluated by four User eXperience (UX) experts who inspected the interface and judged its compliance with the "heuristics". Their findings revealed 26 usability issues out of which 16 were ranked as cosmetic problems only. The remaining 10 have been prioritized in the list below, with the most severe and hardest to fix problems listed first.

Severity 3

- When defining the context of the rule, the purpose of the exclusion ‘–’ button was not obvious to the user (*ease of fix: 1*)
- It is not clear to the user which rules trigger inattention alarms and which identify student behaviors (*ease of fix: 1*)

- There should be the possibility to associate an intervention rule with an existing attention rule (*ease of fix: 1*)
- The wizard should also have a back button available for each step (*ease of fix: 0*)

Severity 2

- Each completed step of the process should change the color of the circle to something else to show that it is saving the information each step of the way (*ease of fix: 1*)
- When defining the context of the rule, one should be able to select all items in a section, i.e., All Courses or All Activities, etc. (*ease of fix: 0*)
- When browsing for a rule the name of its creator should also be visible (*ease of fix: 0*)
- When creating a rule, the user should be able to provide an identifying name (*ease of fix: 0*)
- When deploying a rule, the purpose of the left brace sign is not obvious to the user (*ease of fix: 0*)
- Some literals need revisions because they might be confusing for the users (*ease of fix: 0*)

User-Based Evaluation

Overview

Apart from the heuristic evaluation of LECTORstudio's rule-creating wizards, targeting both developers and non-technical users, the complete functionality of LECTORstudio for developers was assessed through a user-based evaluation experiment. Through this experiment, several usability errors where identified, while great insights where drawn by observing the users interacting with the system and noting their comments and general opinion.

In more detail, five (5) users of ages 25-30 years participated in the experiment. According to Nielsen [109], testing a system with five (5) users permits the

detection of approximately 85% of the problems in an interface, increasing the benefit-cost ratio. All the participants were developers, having worked in projects regarding Ambient Intelligence, and therefore familiar with the related concept and principles. Particularly, two (2) of them were junior developers, two (2) were mid-level developers, while the remaining one (1) was a senior developer.

The experiment was performed in two (2) phases. In the first phase, the concept of the LECTOR framework, as well as the functionality offered through LECTORstudio, was explained to the users. This process was necessary, since the developers should get acquainted with various new concepts such as Actor, Trigger, Behavior, Intervention, Rules, etc., before being able to use the system. After the introduction of all necessary information, the users were requested to browse freely through LECTORstudio in order to get familiarized with it, while they were also encouraged to ask any questions or express any comments.

The second phase of the evaluation experiment was performed the next day after the introductory phase for all users, in order to grant them enough time to assess the acquired information. In this phase, the users were requested to follow a scenario including tasks regarding the major functions of LECTORstudio, while they were also encouraged to express their opinions and thoughts openly following the thinking-aloud protocol [111]. In order to make participants feel comfortable and ensure that the experiment progresses as planned, a facilitator was responsible of orchestrating the entire process, assisting the users when required and managing any technical difficulties. Furthermore, two note-takers were present in order to record any qualitative data – such as impressions, comments and suggestions, along with a number of quantitative data. More specifically, the completion time, the number of help requests and errors made were recorded for each task of the scenario.

After completing the scenario, the users were handed a 10 item questionnaire with five response options ranging from “Strongly agree” to “Strongly

disagree". This questionnaire, namely the System Usability Scale (SUS) [112] provides a "quick and dirty", reliable tool for measuring usability since it:

- Is a very easy scale to administer to participants
- Can be used on small sample sizes with reliable results
- Is valid – it can effectively differentiate between usable and unusable systems.

Finally, the experiment included a debriefing session during which the participants were asked some questions regarding their opinion on the tool, what they liked or disliked the most, and whether they had suggestions about its enhancement.

The following sections present (i) the findings regarding each system function, and (ii) a general discussion for the overall experiment.

Evaluation Findings per System Functions

Each user was requested to complete eight (8) tasks (Appendix C) covering the majority of LECTORstudio's functionality. In reality though, the system would be used by developers that have been assigned specific roles (e.g., rule creator, intervention host integrator), meaning that each developer would have to perform only a subset of the available tasks (e.g., a rule-creator would not have to integrate a new intervention host). The latter, along with the fact that LECTORstudio addresses frequent and not first-time users, will play a key role in interpreting the results of this experiment. The findings of this process are described below categorized per system function.

Function 1. Browsing trigger and behavior outcomes

Evaluated through scenario task 1, 2, 3, 4

Browsing through the available trigger and behavior outcomes was the first thing that users had to do in order to complete Scenario Task 1. Since it was their first encounter with the system, most of them thought that the tiles displaying the trigger and behavior outcomes were representing the actual

rules that lead to the identification of those outcomes. Despite the fact that the errors made (Figure 49) –during performing subsequent tasks that required browsing through trigger and behavior outcomes– were minimized to zero (0), a new representation should be considered.

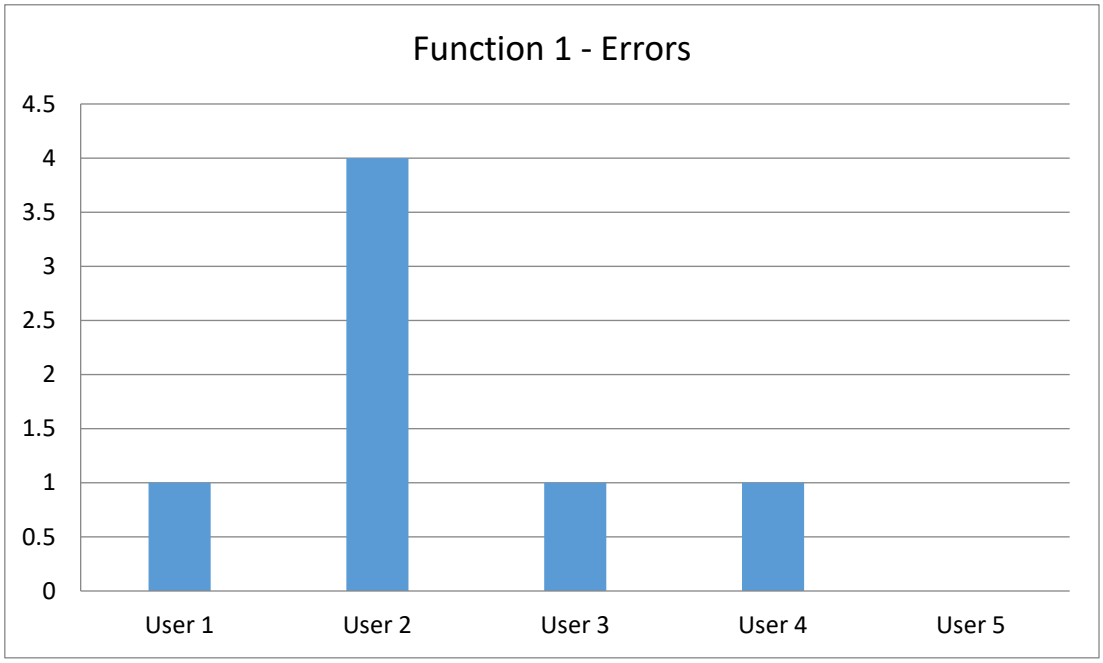

Figure 49: Errors per user for Function 1

Function 2. Creating a new behavior outcome

Evaluated through scenario task 2

All users experienced difficulties in creating a new behavior outcome. Particularly, two (2) issues were identified:

A. The users expected to find and "Add" button under the respective behavior.

B. When editing a behavior it was not clear that the user could add a new outcome through the available tags input control.

Function 3. Creating a behavior rule

Evaluated through scenario tasks 1, 2

Creating a behavior rule was the main objective of the first task that was handed to the users. Most of them made minor errors before being able to complete it; however, during their second attempt of creating a behavior rule as a sub-task of scenario task 2, the errors made were minimized to zero(0) for users 1, 2, 3 and 5 (Figure 50). User 4 made the same mistake that he had made the first time, i.e., he concentrated of the available Filters thinking that through the available fields he could create the desired behavior. In general, despite the errors made, all users agreed that if they were regular users of this system they would be able to create behavior rules easily, since the UI is simple and intuitive. Particularly, user 2 stated that he liked the overview of the created rule, since it provides all necessary information.

Regarding the overview, the usability issues that were identified were the following:

A. Each completed step of the wizard should change the color of the circle to something else in order to provide adequate feedback that it is saving the information at each step of the process.
B. A next button should be available, permitting users to navigate among the wizard steps.
C. When a rule list is empty, the filters should not be visible since the user might get confused.
D. The “create new” button should be in a more obvious location since 3 out of 5 users had trouble locating it.
E. When defining the context of the rule, it was not clear to the users that when all context properties of a category are unselected, they are automatically included to the rule.
F. When configuring a physical property, the purpose of the field “Percentage” is not clear.

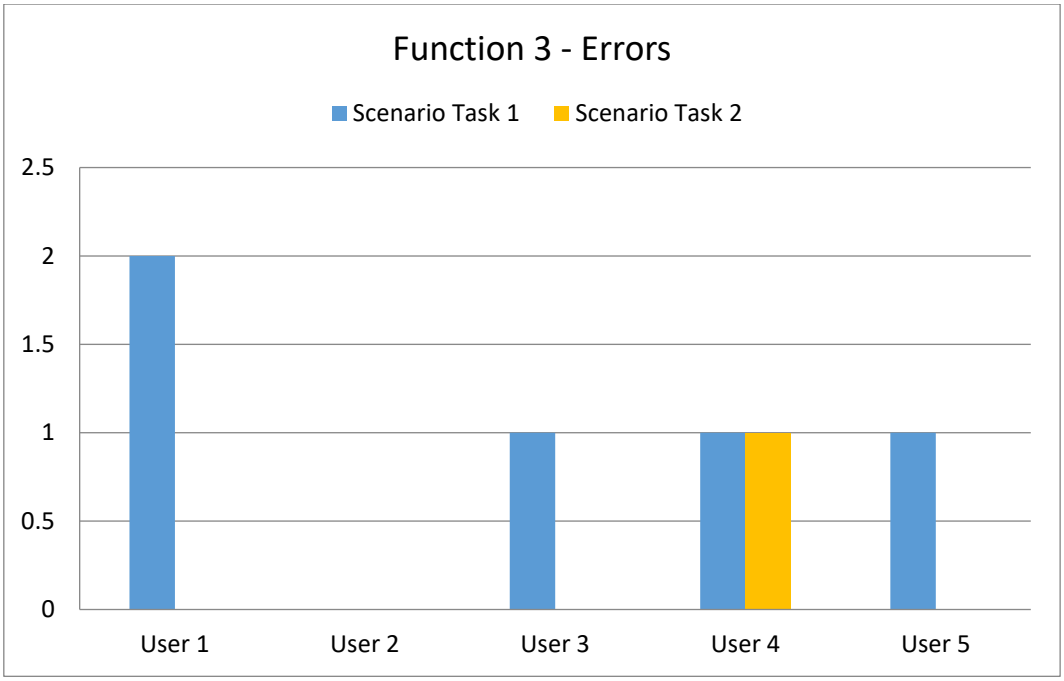

Figure 50: Errors per user for Function 3

Function 4. Integrating a physical context property

Evaluated through scenario task 2

The concept of Physical Context Property troubled almost all users. Particularly, the majority of them thought that in order to integrate such a property, they first had to integrate an artifact. However, the term artifact is used to describe only artifacts that can be used as intervention hosts. To this end, the main menu items should be reconsidered in order to be more intuitive.

Regarding the process of integrating a Physical Context Property, the usability issues that were identified were the following:

A. There should be an indicator regarding the required fields.

B. Most users pressed the "Add new attribute" icon by mistake and then they could not remove the additional form.

Function 5. Creating a trigger rule for an individual student and a group of students (classroom)

Evaluated through scenario tasks 3, 4

Some steps of creating a trigger rule are similar to the process of creating a behavior rule. To this end, the majority of the users completed the related tasks without experiencing any difficulties.

However, some minor issues were identified, in addition to the A, B, C, D, E issues described in Function 3:

A. When selecting an actor, a panel containing its customizable properties appears. However, when the user selects more than one actors, the order in which those panels appear should be reconsidered.

B. When the overview of the rule is displayed, the dropdown expecting input for the rule outcome should be initialized to the value that was implicitly selected by the user when he was browsing through the trigger outcomes.

Function 6. Updating an actor

Evaluated through scenario task 5

The respective scenario task was completed with no errors or help request by all users.

Function 7. Integrating an artifact

Evaluated through scenario task 6

The respective scenario task was completed with no errors or help request by all users, however minor issues were identified:

A. The field expecting the artifact icon was not clear that the user could insert the code of a font icon instead of an actual image

B. The user should not be able to save an artifact if he / she has not provided all the required data.

Function 8. Integrating an intervention type

Evaluated through scenario task 7

The concept of Interventions troubled almost all users. Particularly, the same literal is used to describe both "Intervention Rules" and "Intervention Types". An intervention type is the description of an application that can be used as an intervention, while an intervention rule describes the conditions under which several interventions can be initiated. Therefore, a different literal should be used for each of these functions.

Furthermore, some minor issues were identified:

A. The users didn't understand the purpose of the "core services" input control

B. The users didn't understand the purpose of the "Customizable Properties" form

C. The users didn't understand the difference between Units and Default Unit of the "Customizable Properties" form.

Function 9. Creating an intervention rule

Evaluated through scenario task 8

Some steps of the process of creating an intervention rule are similar to the process of creating a behavior or a trigger rule. To this end, the majority of the users completed the related tasks without experiencing any difficulties. One user though (user 3), suggested that customizing the environment should not be an extra wizard step, instead he suggested that it should be a part of step 2. This suggestion should be further investigated in future user-based experiments.

Additionally, a minor issue was identified, in addition to the A, B, C, D, E issues described in Function 3:

A. When selecting an intervention, it is not clear that the user has to select an artifact for presenting the intervention.

Concluding, some of the identified issues regarding all the examined functions (1-9) stem from the lack of experience that some users have in programming AmI Environments and their difficulty in understanding the broader concept about how LECTOR operates. However, since LECTORstudio targets less experienced users as well, their comments will be taken into consideration for future improvements.

Discussion

Since the experiment included scenario tasks covering the majority of LECTORstudio's functionality, each user should get acquainted with various new concepts such as Actor, Trigger, Behavior, Intervention, Rules etc. To this end, the experiment included an introductory phase, explaining those concepts in details; Nevertheless, all users –during the first 2-3 tasks– experienced difficulties in understanding the terms "Triggers", "Behaviors", "Interventions", "Artifacts" and "Physical Properties". However, all agreed that with frequent use, one gets easily familiarized with the entire concept. This observation is really important, since in reality the system would be used by developers that have been assigned specific roles (e.g., rule creator, artifact integrator), and use the system frequently.

The general opinion of the users, as extracted through the debriefing section was that LECTORstudio is an intuitive tool, with a pleasant UI that they would definitely use for (i) integrating the necessary building blocks and (ii) create the rules that dictate the behavior of an AmI environment. This is also corroborated by the SUS score (77), which indicates that the tool was marked as highly usable. The best way to interpret a SUS score is to convert it to a percentile rank through a process called normalization. The graph presented in Figure 51 shows how percentile ranks associate with SUS scores and letter

grades (from A+ to F) [113]. According to this graph, LECTORstudio's score (77) converts to a percentile rank of 80% and it can be interpreted as a grade of B+.

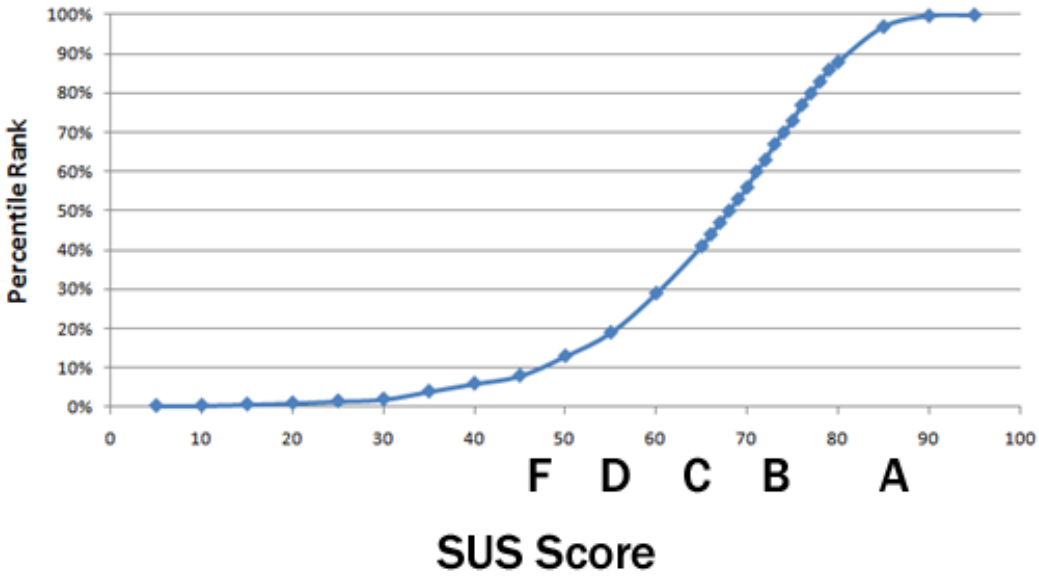

Figure 51: SUS scores association with percentile ranks and letter grades [113]

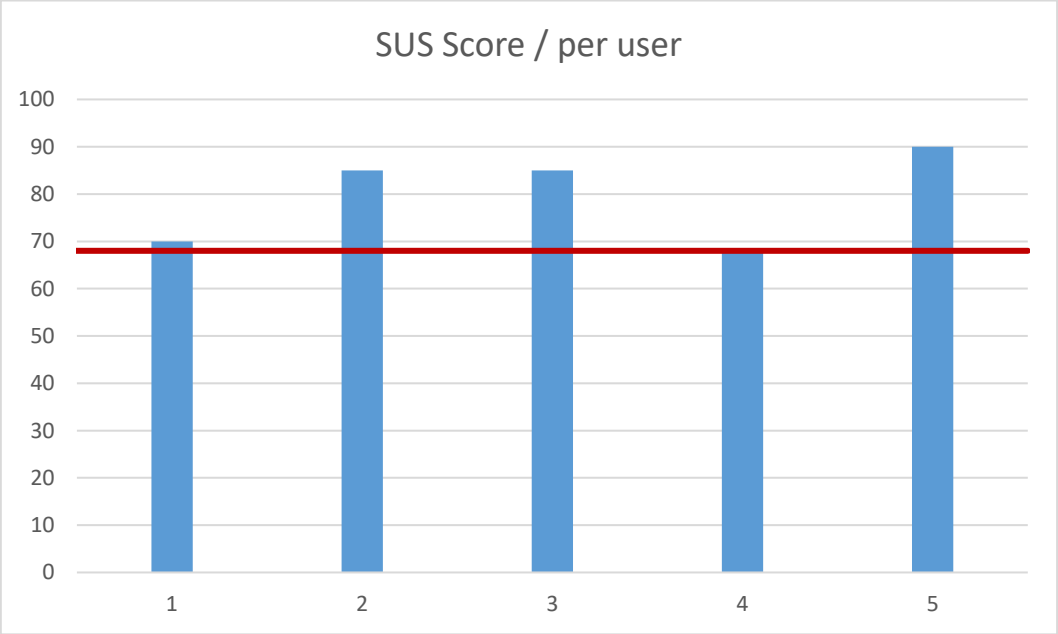

Figure 52: SUS score per user

Figure 52 presents the SUS score per user; considering that SUS has a usability threshold of 68%, users 1, 3 and 5 graded the system with a score high above the threshold, while users 2 and 4 (Junior developers) provided a score close the threshold (70% and 68% respectively). The latter was anticipated, since

these users had the minimum experience in programming AmI Environments and they faced difficulties understanding the broader concept about how LECTOR operates.

Some of the most notable user comments are presented below:

- "*Very useful tool*" (user 1)
- "*It is very simple to use as soon as you get acquainted with the new concepts*" (user 1)
- "*The design is very nice*" (user 2)
- "*I would definitely use it if I were an educator*" (user 2)
- "Nice and Simple" (user 3)
- "*Without the tool, adding such functionality to the environment would be a cumbersome process*" (user 3)
- "*The rule-creating wizards are nice tools especially for someone with no programming skills*" (user 4)
- "*I would definitely use it!*" (user 5)

Finally, despite the fact that no user mentioned it, it is believed that introducing a dashboard to LECTORstudio would be beneficial. A dashboard is defined as "a visual display of data used to monitor conditions and/or facilitate understanding" [114], therefore it would enable developers to get an at-a-glance view of the system's available components and help them form a better conceptual model regarding the entire functionality.

LECTORviewer: Managing the Attention-Aware Intelligent Classroom

Description

LECTORviewer [96], [97] is a web-based tool for managing the attention-aware intelligent classroom. It is deployed on the teacher's personal workstation and allows the observation and customization of LECTOR's decisions regarding either individual students or the classroom as a whole. In more detail, LECTORviewer offers the following:

- One-click enabling or disabling of the LECTOR's monitoring facility.
- One-click enabling or disabling of the LECTOR's intervention mechanism.
- An overview of the attention level of the entire classroom that also facilitates focusing on particular students.
- A mechanism that asks the educator's opinion regarding ambiguous student behaviors.
- A mechanism that gives educators control over approving or dismissing an intervention.

These functionalities are provided through an intuitive user interface which mainly consists of (i) a main dashboard that displays information regarding all the classes an educator teaches, and (ii) the representations of each class (i.e., class view) containing details about its students, displayed either in a seating chart layout or a list view.

All the classes that an educator teaches can be found in a sortable list on the main dashboard, where valuable information is available to the teacher: (i) the schedule of the class (e.g., the assignments that are close to a deadline), (ii) reminders of important events (e.g., scheduled exam), (iii) details about the fluctuation of the attention levels during the last course, and (iv) number of successful interventions. This type of information not only helps educators to have an overview of the class and better organize future lessons, but also judge the efficiency and quality of past courses based on the students' attention

levels. Moreover, by viewing the statistics about the effectiveness of past interventions, educators can acquire an understanding of the kind of interventions that are appropriate for a specific class or student, and therefore more efficiently choose and manage interventions in the future.

During a course, through LECTORviewer's class view the educator can get insights regarding attentive students or students that are not paying attention due to fatigue, mind wandering, or lack of motivation. However, in some cases the ability to disambiguate student activities depends on information that only a human can provide. For example, students laughing at a teacher's joke is not an indicator of inattention. To that end, when the system identifies a behavior that can be misinterpreted, it asks for the teacher's opinion. These three states (i.e., attentive, not attentive and needs revision) are coded with appropriate colors (i.e., green, red and orange) which are used throughout the user interface so as to help educators easily distinguish the status of the students.

At the top of the "class view" (Figure 53), the educator can see at a glance the attention percentage of the classroom as a whole. A pie chart, located at the top left of the page, uses the aforementioned colors to display the percentage of attentive or inattentive behaviors, and situations that require revision. At the center of the chart the percentage of attentive students is displayed using bold and large fonts so as to ensure that the educator will be able to see it even from a distance. Furthermore, the legends of the chart can be used as filters that modify its contents, thus enabling educators to customize it according to their needs. The representation as a pie chart was considered as the best alternative to communicate this type of information to educators by displaying all the data simultaneously; that is because a person's visual system needs less time to understand graphs (rather than tables), which give numbers shape and form [115].

In addition, in order to ensure that educators can freely activate or deactivate the monitoring and intervention mechanisms according to the classroom's needs, the top of the page contains the appropriate controls so as to be easily

accessible. This functionality is important for an environment full of students where unforeseen situations can emerge; for example, the educator could observe that interventions are not effective or disrupt the courses flow at a given moment, and may wish to stop the system from making suggestions. Apart from merely (de)activating interventions, educators can select to start a specific intervention when deemed necessary. The latter ensures that educators do not rely on the system decisions alone; on the contrary, they can initiate custom interventions in case the system (i) fails to identify that the students require remedial actions, and (ii) suggests an inappropriate one.

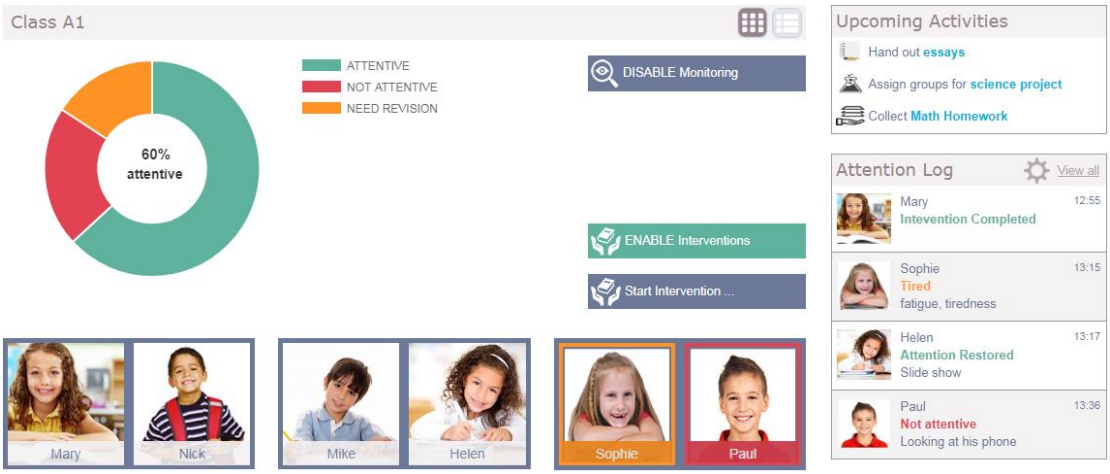

Figure 53: Snapshot of LECTORviewer's class-view.

Apart from managing the classroom as a whole, the educators can focus on individual students as well. In more detail, there are two alternative layouts available for browsing through the classroom students and observing their status. By default a "seating chart" layout is displayed, where students are represented in a form that resembles their actual seating arrangements, while the educator can easily switch to a "list view" layout, with a rich sorting functionality (e.g., alphabetical order, attention level order, etc.). For each student, LECTORviewer displays useful information regarding their status, as well as the likely reason a student is inattentive.

When the list view of the class is enabled, more functionality regarding each individual student is displayed. For each student, additional information is available, such as details regarding her learning style, her attention level, and

the reason that led the system to identify that she has lost focus if that is the case. Furthermore, in order to provide enough context to the educator, in case of inattention or behaviors that need revision, relevant tags that reveal the reason are available. An indicative tag is "Mobile", which is used to annotate the behavior of students who are not paying attention because they are looking at their smartphones. Finally, next to each student the educator can find the appropriate controls for enabling or disabling LECTOR's monitoring and intervention mechanisms for that individual. This is required in a class that is constituted of different students with varying backgrounds, personalities, behaviors, needs and learning patterns [116].

Additionally, a detailed log (Figure 54) is available for each classroom that allows educators to revisit –even at a later time– LECTOR's decisions and mark them as accurate or not. A mini view of the log is always available at the sidebar of the "class view", enabling educators to observe at real time LECTOR's decisions without navigating to a new page. However, if needed the educator can select to view the entire attention log, through which she can (i) confirm or invalidate an identified student behavior, (ii) stop an active intervention and optionally replace it with another one, and (iii) rate elapsed interventions. Providing such information is really important for "calibrating" LECTOR with a specific classroom environment and its students, since this process makes the decision-making mechanisms more accurate and less prone to false positives. This is a cumbersome task, which requires recalling various incidents that occurred during a significant amount of time. In order to minimize the amount of information someone has to remember, LECTORviewer's log is equipped with a sophisticated filtering mechanism, while each log entry is accompanied with abundant contextual information (e.g., timestamp, teacher's activity at the time, etc).

Finally, on the top right of the screen important upcoming activities concerning the current course are visible. This enables the educator to have a quick overview of tasks that are time-critical, thus giving him the opportunity

to better organize the activities, while also serving as a reminder. Icons visible next to each upcoming activity aid the fast recognition of the activity at just a quick glance.

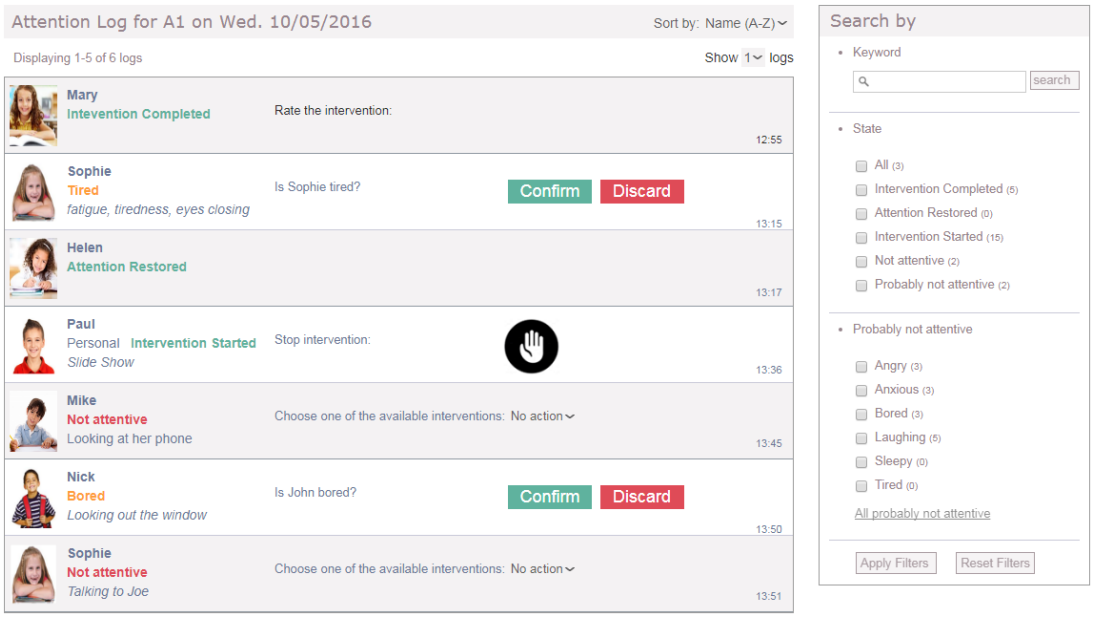

Figure 54: Snapshot of LECTORviewer's detailed log.

Design Process and Heuristic Evaluation

An iterative design process was followed throughout the development lifecycle of LECTORviewer. The first phase of this process involved the creation of low fidelity paper prototypes exhibiting the entire functionality of the system. These were initially assessed by three (3) Human Computer Interaction experts during a cognitive walkthrough evaluation experiment in order to uncover any usability errors. Firstly, the evaluators were asked to browse through the paper prototypes and express their thoughts and questions about the design, while two coordinators were taking notes of their comments. Secondly, they were given a scenario with some tasks to complete and they were asked to follow the Thinking-Aloud protocol and pinpoint any usability-related issues that they identify. Finally, as soon as their comments were consolidated in a single list, they were asked to grade them in terms of severity so as to compile a prioritized list with the issues that have to be addressed. The evaluation process uncovered various problems regarding not

only the design of the User Interface, but also regarding the overall concept. The major findings are summarized below:

- The functionality available to the teacher during class hour should be limited to configuring the attention and intervention mechanisms. Other operations could possibly overwhelm the user while they would take over much of the teaching hour. For example, adding students to the class and rearranging their positions should be performed during the teacher's spare time or, even better, a secretary should be responsible for such activities.
- The edit button was used to reveal the delete and add buttons, while the edit screen of an item (e.g., a classroom, a student, a rule) is displayed when clicking on its name, independently of the "edit" button, which affects the consistency and the expected behavior of the visual interface.
- One out of three evaluators thought that the "classroom" metaphor (doors, books, desks, board) that is used throughout the design would be cumbersome for a teacher that uses the system on a daily basis and wants to concentrate on important and time-critical tasks.
- All evaluators pointed out that the lists displaying the rules and interventions should be accompanied by a rich filtering mechanism to assist the educators in finding whatever they want quickly.

The major findings of this experiment were mostly related to the complexity of the most frequently used screens, and secondly to the metaphors used in the design, suggesting their refinement in order to simplify the interaction paradigm used to execute time-critical or common functions expected to occur on a daily basis. Subsequently, an improved vertical high fidelity interactive prototype [117] was created integrating the feedback received and was re-assessed by five (5) UX experts via heuristic evaluation [108] in order to test the overall usability and address any problems before conducting a full-scale user-based evaluation with the target audience (i.e. educators).

The problems identified through the experiment where ranked according to their severity by the evaluators. The severity ratings range from zero ("not a

usability problem") to four ("Usability catastrophe") [118] and are used to indicate how serious each problem is and how important is to fix it. Next, the development team ranked each problem with an ease-of-fix ranking ranging from zero ("would be extremely easy to fix") to three ("would be difficult to fix") to designate the amount of effort needed to address it. This process revealed 16 usability issues out of which 2 were ranked as cosmetic problems only, 7 were identified as minor usability problems, and the remaining 7 where ranked as major issues, hence the most important to fix. All 16 issues have been prioritized in the list below, with the most severe and hardest to fix problems listed first.

Priority 3

- The extra information that is provided in the list view should also be available in the seating chart view (*ease-of-fix 1*)
- There should be a summary log for each class, containing diagrams that display how many interventions have been done during a course, and the success rate of interventions (*ease-of-fix 1*)
- It was not clear that the pie chart of attention had filters (*ease-of-fix 0*)
- The percentages of the pie chart should be immediately visible without having to hover over them (*ease-of-fix 0*)
- The focus of the main screen should be the students, everything else is of secondary importance. The pie chart and buttons in the upper part of the screen is of secondary importance and should be located elsewhere (*ease-of-fix 0*)
- There should be a way to see in which mode I am viewing the class: while the course is taking place, or not? (*ease-of-fix 0*)
- Instead of Enable/Disable Interventions the button should read Enable/Disable Auto-Interventions (*ease-of-fix 0*)

Priority 2

- There should not be paging in the log for the same day, for each day there should be infinite scrolling (*ease-of-fix 1*)
- Instead of the label “need revision” the label “uncertain” should be used (*ease-of-fix 0*)
- In the seating chart layout, there should also be a sign for where the teacher’s desk is for orientation purposes (*ease-of-fix 0*)
- Since the system is a real-time one, the hour should be visible somewhere on the interface (*ease-of-fix 0*)
- The messages displaying the status of a student should be clearer (*ease-of-fix 0*)
- It is not clear that the orange color represents the state that the educator must revise the system’s decision (*ease-of-fix 0*)
- It is not clear that the STOP hand icon stops an active intervention (*ease-of-fix 0*)

Priority 1

- Upcoming Activities are of secondary importance and should maybe be a drawer toolbar (*ease-of-fix 1*)
- The title “Log” should be changed to “Reviewer” (*ease-of-fix 0*)

According to that list, fixing the identified issues requires minimum effort on behalf of the developers.

NotifEye: A Smart Watch Application for Hosting Interventions

NotifEye is a smartwatch application able to provide subtle interventions to both students or educators. Employing such wearable devices to act as intervention hosts seemed natural, since in addition to indicating time they: (i) are increasingly available to the market, (ii) are familiar to adults and children, (iii) support notifications and reminders, and (iv) are appropriate for private interventions.

"NotifEye for educators" (Figure 55) can be used to provide informative interventions regarding important incidents that occur during a course. In more detail, the application is able to display LECTOR messages , while at the same time the watch vibrates to alert the user. For example, when the entire classroom displays signs of inattention, NotifEye is instructed to deliver the short yet meaningful message "CLASSROOM TIRED", accompanied with an exclamation mark icon. The use of self-explanatory icons that require little effort to see and understand was imperative for an application running on a wearable small-screen device whose target audience must not be distracted from its main task (i.e., being a teacher).

Figure 55: Snapshots from NotifEye for educators

Furthermore, apart from delivering notifications, the educator's smart watch is used as an input device through which she can communicate useful information to LECTOR. Specifically, when a class-wide intervention is about to start, NotifEye displays a message asking for approval; in case the educator rejects it, LECTOR is notified so as to increase the cancelation percentage of the selected intervention accordingly.

Similar functionality is offered through "NotifEye for students", which can be used to deliver encouraging or inspiring messages to unmotivated individuals. Moreover, the students have the opportunity to provide input directly to LECTOR in case they find themselves mind-wandering or tired. The latter is really important during the adjustment period where the system has to be "calibrated" to the needs of each individual student.

Future work includes extending NotifEye to make use of smart watches' ability of monitoring physical activity and physiological measures in order to identify cues of inattention [119] .

CognitOS: A Student-Centric Working Environment for an Attention-Aware Intelligent Classroom

CognitOS [95] is a sophisticated web-based working environment that hosts educational applications, which are also utilized as a channel to present LECTOR interventions . For example, a mini-quiz application can be launched either explicitly by a student who selects a specific exercise on her book, or automatically when LECTOR intervenes to display a fun quiz to keep her motivated during a reading assignment. CognitOS is deployed permanently on the technologically augmented desks residing in the intelligent classroom, each of which features a 27-inch multitouch-enabled All-in-One PC and integrates various sensors (e.g., eye-tracker, camera, microphone, etc.), and on demand on the students' personal smartphones and tablets.

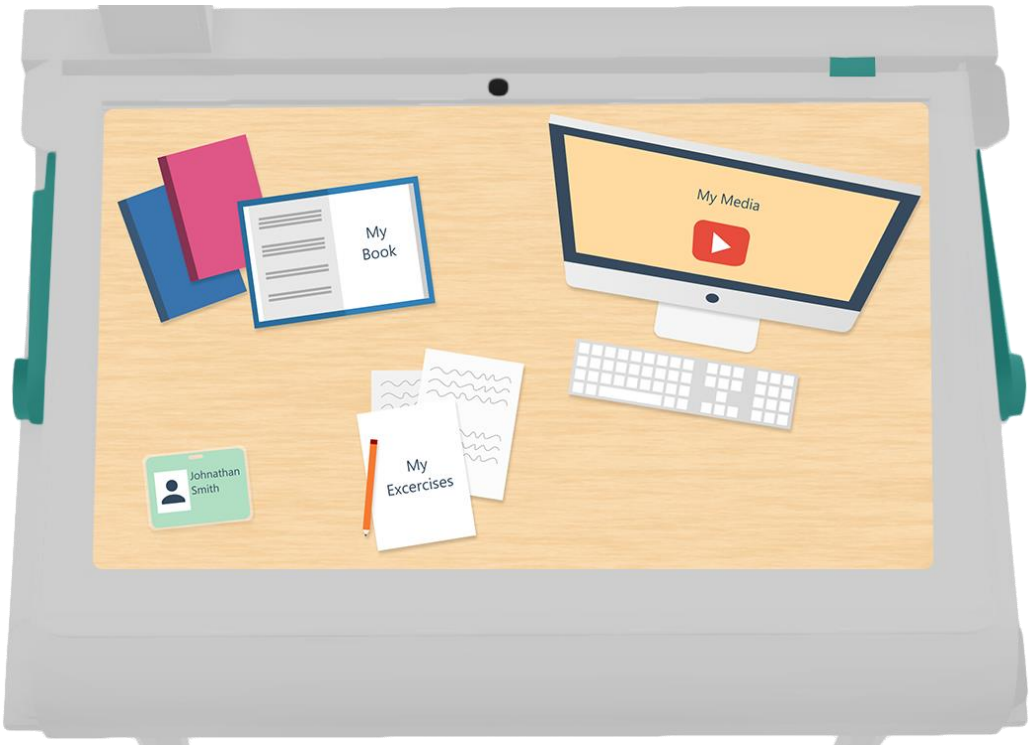

Figure 56: The desktop of CognitOS running on the augmented school desk.

Figure 56, presents the main working area of CognitOS (i.e., the desktop). It follows the metaphor of an actual desk containing virtual student items (e.g.,

books, pencils, etc.) that can be used to launch the respective applications. In more detail, the desktop contains:

- A **pile of books** that offers a shortcut to the student's collection of books. The topmost book is always related to the current course and looks open; the student can use it to quickly launch the book application with the respective content.
- A **pile of notebook pages** that acts as a shortcut to student's collection of completed or pending assignments. The first page filters the assignments' list and displays only those related to the current course.
- The **personal card** that displays the student's name and provides access to the profile application with the detailed academic record of that student.
- A **computer monitor** that can be used to launch the multimedia player for presenting such content (i.e., pictures and videos).

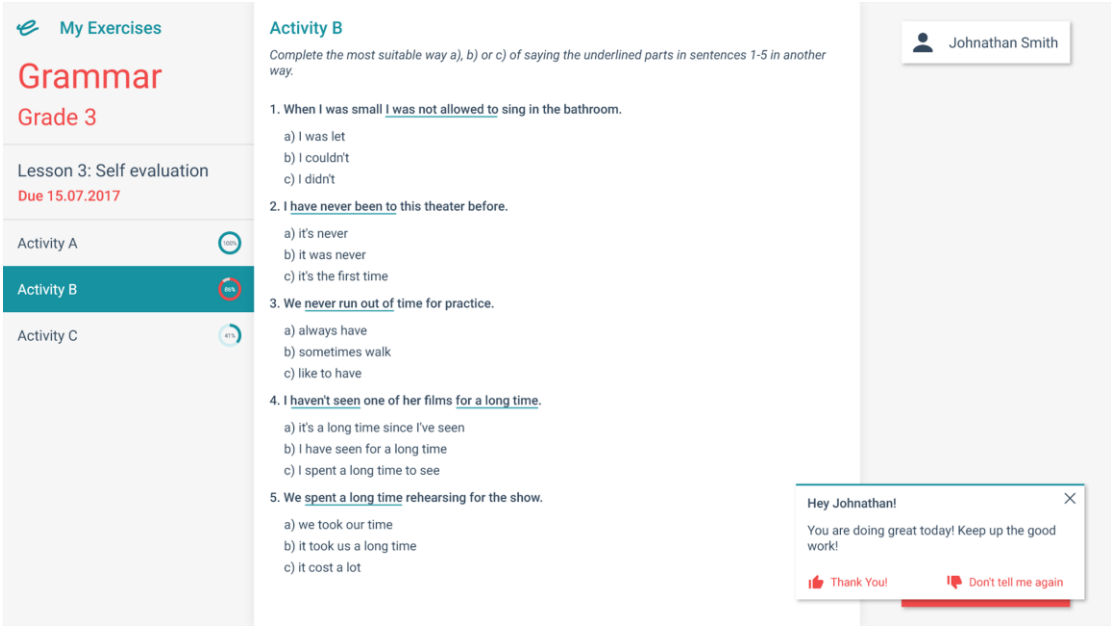

Figure 57: A notification appears trying to encourage a student during exercise solving

A digital educational working environment should allow students to launch multiple applications simultaneously (e.g., digital books, dictionary, personal documents, etc.); therefore, it requires a mechanism that decides the placement of each newly launched application. To this end, a sophisticated algorithm was introduced ensuring that (i) if an application displays additional

information related to another application they will always be launched next to each other, (ii) the application with which the user had interacted last will remain on top and (iii) secondary applications (e.g., calendar, calculator, etc.) will occupy less screen real-estate if more important ones are already present. Nevertheless, in addition to automatic layout, CognitOS permits the rearrangement of any launched applications so that each student can customize the working environment according to his/her personal preference.

However, apart from common application management, CognitOS has the ability to present four (4) types of interventions: notices, augmentations, alterations and restrictions. As soon as LECTOR plans a specific intervention, CognitOS receives a command via AmI-Solertis to launch the appropriate application(s). In more detail, an advanced notification mechanism is featured for delivering appropriate messages (i.e., notice) to the students who seem unmotivated (Figure 57), troubled or disengaged from the task at hand. Furthermore, CognitOS features a collection of educational applications that can be launched on demand with specific content, so as to present motivating material. Each application is available in full- and mini-view; the latter is employed to present auxiliary content alongside with other material (i.e., augmentation), while full-view applications aim to monopolize the student's interest (i.e., alteration). Finally, they can also get locked (i.e., restriction), denying access to the students, when either the teacher or LECTOR deems them irrelevant to the current activity.

Chapter 7
Conclusions and Future Work

Conclusions

The primary goal of an AmI environment is to help and support the people living in it; towards this objective, it should be able to identify user needs and act accordingly. Many research approaches and commercial tools have focused on realizing this concept, which follow the fitting paradigm of the trigger-action model; however, the majority of them poses several limitations as described in Chapter 3. Specifically, the domain of education would particularly benefit from an AmI environment able to monitor students during their educational activities and intervene when deemed necessary to help, support or motivate them so as to enhance the learning process. However, despite the fact that the Intelligent Classroom has gained much attention from researchers over the past decade, a solution that offers a generic, scalable, fast and easy way to connect triggers with actions in the classroom context is not currently available.

Aiming to bridge this gap, this thesis has presented a framework and an authoring tool that support both developers and educators in defining the **behaviors** (triggers) that initiate context-aware **interventions** (actions). This framework equips the Intelligent Classroom with attention-aware facilities that monitor the learners' attention levels and intervene when necessary to (i) provide motivating activities to distracted, unmotivated or tired individuals or

(ii) suggest alternative learning methodologies to educators that would be beneficial for the entire classroom.

In more detail, the LECTOR framework offers a mechanism for identifying student **behaviors** that require remedial actions and intervening when the students need help or support. This mechanism relies on "if-then" rules - created either by developers or by educators- to dictate the reaction of the classroom environment to student-oriented stimuli. In order to ensure scalability and simplify rules' management, a three (3) step process for connecting a behavior with an intervention has been introduced. In particular, the first step requires the user to define a **behavior**, next the conditions under which the behavior becomes a **trigger** have to be described, and during the last step connections between a trigger and appropriate **interventions** are created. This decomposition permits a behavior to be associated with multiple triggers, and a trigger with multiple interventions that alternate depending on the context of use, hence ensure scalability and reuse. Additionally, the fact that a recipe connecting N behaviors with M interventions is composed by three independent rules rather than a monolithic one, enables end-users to easily manipulate certain parts of the recipe without affecting the others, thus simplifying rule management and minimizing ramification. Furthermore, in contrast to the artifact-oriented approach offered by the majority of tools that enable end-users in creating simple recipes, LECTOR's rule structure supports the creation of **user-oriented** intervention scenarios, which harmonize with the human-centered nature of AmI environments.

Developers and educators can easily and rapidly create the rules that describe: (i) behaviors, (ii) triggers, and (iii) interventions through a sophisticated user-friendly authoring tool, named LECTORstudio. The findings of the user-based evaluation corroborate the fact that LECTORstudio permits not only computer experts but also non-technical users to create their own scenarios and customize LECTOR's decision-making process according to their needs. Despite the limitations in expressiveness of the programs stemming from the

use of “if-then” rules, this work provides appropriate tooling and conceptual models that create an ideal environment for users to create their own integrations.

LECTORstudio also enables developers to integrate the building blocks (i.e., actors, artifacts, physical context, virtual context, interventions) necessary for programming the classroom environment. This functionality permits LECTOR to be scalable, easily customizable, and open to new additions (e.g., sensors, applications, actors, environments). According to the user-based evaluation findings, the general opinion of the evaluators was that it constitutes an intuitive and useful tool that they would definitely use to establish the conditions that trigger remedial actions and the respective intervention strategies, since it minimizes the amount of work required on their behalf.

Lastly, in order to further support the targeted end-users of this environment (i.e., students and educators), this work has also introduced three (3) additional tools: LECTORviewer, NotifEye and CognitOS. The former two (namely LECTORviewer and NotifEye) aim to support educators in having an overview of the students’ attention levels and providing their input regarding ambiguous behaviors or scheduled interventions that aim to re-engage distracted, tired or unmotivated students. CognitOS on the other hand, is a sophisticated web-based working environment for students that hosts a variety of educational applications, which comprise the communication channels through which LECTOR presents the interventions.

The potential of this framework and the encouraging results of its deployment in the in-vitro Intelligent Classroom of ICS-FORTH raised the question of whether it can be used to support other intelligent environments and domains as well. The majority of current state-of-the-art approaches allow non-technical users to express their preferences through simple “recipes” and programs that feature rule-based conditions. LECTOR and its accompanying authoring tools streamline such processes, therefore its generalization to support other domains constitutes a logical extension. Additionally, the

framework not only follows well-established state-of-the-art practices, but in some cases it advances the way through which both developers and non-technical users connect human- or artifact- initiated triggers to specific context-aware actions as well. Particularly, LECTOR:

- Supports the creation of **user-oriented** behavior-intervention (trigger-action) scenarios in contrast to the artifact-oriented recipes.
- Enables the definition of behaviors that combine multiple contextual information.
- Supports the connection of N behaviors with M interventions (where N ≠ M).
- Permits the definition of multimodal and ubiquitous interventions.
- Provides a mechanism for assessing the efficacy of interventions.

The above functionality addresses some of the gaps identified in the related work discussed in Chapter 3, and introduces some features that would be beneficial for a rule-based programming environment. In order to support such claim, both LECTOR and LECTORstudio have been deployed in the in-vitro Intelligent Home [1] of ICS-FORTH, where their potential in such environments is being examined. Specifically, given the ambient facilities available through the Intelligent Home, LECTOR aims to realize the scenarios described in Appendix D.

Currently, LECTOR is also employed to support CaLmi [120], a pervasive system that aims to reduce the stress of the inhabitants of an Intelligent Home. In more detail, LECTORstudio is being used to create rules that (i) guide LECTOR in identifying users that require support due to increased stress levels, and (ii) that define the interventions (i.e., relaxation techniques) that will be initiated depending on the situation. CaLmi employs a wristband that collects user psychophysiological signals (i.e., EDA, HRV, ST and accelerometer), while it utilizes various contextual data (user agenda, bank

[1] See http://ami.ics.forth.gr

account transitions and current conditions) in order to better understand the user's daily activities and their impact on stress. As soon as a STRESS trigger is identified, LECTOR's IM component is able to select an appropriate stress relief technique and create a relaxing experience (Figure 58).

Figure 58: CaLmi exposes the user to images and videos of natural environments so as to help her relax.

From the accumulated experience it can be concluded that the LECTOR framework can be easily generalised to contexts different from the classroom, and can therefore provide a generic behaviour intervention mechanism in Ambient Intelligence environments.

Future Work

Plans for future work include additional steps to fully support the initial concept. The first step concerns the improvement of the developed frontend tools and backend infrastructure so as to promote their evolution from in-vitro prototypes to mature software products. Towards that direction, LECTORstudio and LECTORviewer should be improved according to the findings of the heuristic and user-based evaluation experiments. Furthermore, full scale user-based experiments will be conducted for all tools, including CognitOS and NotifEye, in order to acquire valuable feedback from the actual target groups of each tool.

Additionally, an idea to empower LECTORstudio –the main tool proposed by this thesis– would be to provide a graphical user interface through which developers would be able to employ visual programming to provide the

necessary code when needed. Currently, text-based programming is supported through the AmI Solertis web-editor; however, in some cases it would be beneficial to alternate to a graphical tool that facilitates coding.

Another valuable addition to LECTORstudio would be to permit developers to modify the available handlebars templates, which are responsible for translating the created rules to 'if-then' statements. That way, developers would be empowered to easily fix any identified issues, or even incorporate more elaborated reasoning mechanisms with zero changes to the framework itself.

Regarding the LECTOR framework itself, the LEARNING component should be able to expand its knowledge by discovering unknown activity patterns directly from the sensed data. That way, each dynamically discovered activity pattern would be used to define new behaviors that can be recognized and tracked. Towards that direction, a Pattern Discovery Engine should be introduced in order to analyze any unlabeled sensed data along with the Virtual Context in order to find any "out of vocabulary" recurring behaviors. However, this approach would require the active participation of end-users in order to identify whether the discovered behaviors should be considered as triggers or not.

Additionally, a long-term user-based evaluation experiment should be conducted, in order to acquire valuable feedback regarding the efficacy of LECTOR and auxiliary tools under an educational context, and their acceptance by both students and educators.

Finally, the benefits of employing such a framework in other environments will be further investigated. LECTOR is planned to be incorporated in additional in-vitro simulation spaces[2] located at the FORTH-ICS AmI Facility Building, such as the Intelligent Hotel Room and the Intelligent Greenhouse. These two entirely different environments are ideal testbeds for assessing

[2] See http://ami.ics.forth.gr

whether the framework is appropriate for general use and identifying any shortcomings that need to be addressed before evolving LECTOR into a generic tool.

Bibliography

[1] E. Aarts and R. Wichert, “Ambient intelligence,” in *Technology Guide*, Springer, 2009, pp. 244–249.

[2] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, “Practical Trigger-action Programming in the Smart Home,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2014, pp. 803–812.

[3] B. Ur *et al.*, “Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2016, pp. 3227–3231.

[4] J. Huang and M. Cakmak, “Supporting Mental Model Accuracy in Trigger-action Programming,” in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, New York, NY, USA, 2015, pp. 215–225.

[5] P. Remagnino and G. L. Foresti, “Ambient Intelligence: A New Multidisciplinary Paradigm,” *IEEE Trans. Syst. Man Cybern. - Part Syst. Hum.*, vol. 35, no. 1, pp. 1–6, Jan. 2005.

[6] D. J. Cook, J. C. Augusto, and V. R. Jakkula, “Ambient intelligence: Technologies, applications, and opportunities,” *Pervasive Mob. Comput.*, vol. 5, no. 4, pp. 277–298, 2009.

[7] A. L. Cheng, C. Georgoulas, and T. Bock, “Fall Detection and Intervention based on Wireless Sensor Network Technologies,” *Autom. Constr.*, vol. 71, no. Part 1, pp. 116–136, Nov. 2016.

[8] “Automatic Fall Detection | Philips Lifeline ®,” *Philips*. [Online]. Available: https://www.lifeline.philips.com/automatic-fall-detection.html. [Accessed: 05-Oct-2017].

[9] H. Gjoreski, J. Bizjak, and M. Gams, “Using Smartwatch as Telecare and Fall Detection Device,” in *2016 12th International Conference on Intelligent Environments (IE)*, 2016, pp. 242–245.

[10] C. C. Quinn, M. D. Shardell, M. L. Terrin, E. A. Barr, S. H. Ballew, and A. L. Gruber-Baldini, “Cluster-randomized trial of a mobile phone personalized behavioral intervention for blood glucose control,” *Diabetes Care*, vol. 34, no. 9, pp. 1934–1942, 2011.

[11] S. Gupta, P. Chang, N. Anyigbo, and A. Sabharwal, “mobileSpiro: accurate mobile spirometry for self-management of asthma,” in *Proceedings of the First ACM Workshop on Mobile Systems, Applications, and Services for Healthcare*, 2011, p. 1.

[12] C. Bexelius, M. Löf, S. Sandin, Y. T. Lagerros, E. Forsum, and J.-E. Litton, “Measures of physical activity using cell phones: validation using criterion methods,” *J. Med. Internet Res.*, vol. 12, no. 1, 2010.

[13] K. Patrick *et al.*, “A text message–based intervention for weight loss: randomized controlled trial,” *J. Med. Internet Res.*, vol. 11, no. 1, 2009.

[14] A. A. Ali, S. M. Hossain, K. Hovsepian, M. M. Rahman, K. Plarre, and S. Kumar, “mPuff: automated detection of cigarette smoking puffs from respiration measurements,” in *Information Processing in Sensor Networks (IPSN), 2012 ACM/IEEE 11th International Conference on*, 2012, pp. 269–280.

[15] K. Plarre *et al.*, “Continuous inference of psychological stress from sensory measurements collected in the natural environment,” in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on Information Processing in Sensor Networks*, 2011, pp. 97–108.

[16] J. Chen *et al.*, "FishBuddy: Promoting Student Engagement in Self-Paced Learning through Wearable Sensing," in *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2017, pp. 1–9.

[17] M. N. Burns *et al.*, "Harnessing context sensing to develop a mobile intervention for depression," *J. Med. Internet Res.*, vol. 13, no. 3, 2011.

[18] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *Int. J. Commun. Syst.*, vol. 25, no. 9, p. 1101, 2012.

[19] P. Mell and T. Grance, "The NIST definition of cloud computing," 2011.

[20] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," *J. Clean. Prod.*, vol. 140, pp. 1454–1464, 2017.

[21] "IFTTT." [Online]. Available: https://ifttt.com/. [Accessed: 05-Oct-2017].

[22] M. W. Newman, A. Elliott, and T. F. Smith, "Providing an Integrated User Experience of Networked Media, Devices, and Services through End-User Composition," in *Pervasive Computing*, 2008, pp. 213–227.

[23] M. Walch, M. Rietzler, J. Greim, F. Schaub, B. Wiedersheim, and M. Weber, "homeBLOX: making home automation usable," in *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, 2013, pp. 295–298.

[24] L. De Russis and F. Corno, "HomeRules: A Tangible End-User Programming Interface for Smart Homes," in *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2015, pp. 2109–2114.

[25] "Zipato," *Zipato- Home Control and Automation*. [Online]. Available: https://www.zipato.com/. [Accessed: 07-Oct-2017].

[26] "Supermechanical: Twine. Listen to your world. Talk to the web." [Online]. Available: http://supermechanical.com/twine/technical.html. [Accessed: 06-Oct-2017].

[27] "Zapier," *The best apps. Better together. - Zapier*. [Online]. Available: https://zapier.com/. [Accessed: 07-Oct-2017].

[28] "behaviour Meaning in the Cambridge English Dictionary." [Online]. Available: http://dictionary.cambridge.org/dictionary/english/behaviour. [Accessed: 11-Oct-2017].

[29] "intervention Meaning in the Cambridge English Dictionary." [Online]. Available: http://dictionary.cambridge.org/dictionary/english/intervention. [Accessed: 02-Oct-2017].

[30] M. R. W. Dawson, *Minds and Machines: Connectionism and Psychological Modeling*. John Wiley & Sons, 2008.

[31] M. Siegel, "The sense-think-act paradigm revisited," in *Robotic Sensing, 2003. ROSE'03. 1st International Workshop on*, 2003, p. 5 pp.

[32] V. L. Tinio, *ICT in Education*. e-ASEAN Task Force, 2003.

[33] P. C. Abrami *et al.*, "A review of e-learning in Canada: A rough sketch of the evidence, gaps and promising directions," *Can. J. Learn. Technol. Rev. Can. L'apprentissage Technol.*, vol. 32, no. 3, 2008.

[34] A. T. Bates, *Technology, e-learning and distance education*. Routledge, 2005.

[35] N. Cross and R. Roy, *Engineering design methods*, vol. 4. Wiley New York, 1989.

[36] C. Brooks, J. Greer, E. Melis, and C. Ullrich, "Combining its and elearning technologies: Opportunities and challenges," in *Intelligent Tutoring Systems*, 2006, vol. 4053, pp. 278–287.

[37] P. Xu, G. Han, W. Li, Z. Wu, and M. Zhou, "Towards intelligent interaction in classroom," *Univers. Access Hum.-Comput. Interact. Appl. Serv.*, pp. 150–156, 2009.

[38] D. J. Cook and S. K. Das, "How smart are our environments? An updated look at the state of the art," *Pervasive Mob. Comput.*, vol. 3, no. 2, pp. 53–73, 2007.

[39] Y. Shi *et al.*, "The smart classroom: merging technologies for seamless tele-education," *IEEE Pervasive Comput.*, vol. 2, no. 2, pp. 47–55, 2003.

[40] J. R. Cooperstock, "The classroom of the future: enhancing education through augmented reality," in *Proc. HCI Inter. 2001 Conf. on Human-Computer Interaction*, 2001, vol. 1, pp. 688–692.

[41] M. Antona, A. Leonidis, G. Margetis, M. Korozi, S. Ntoa, and C. Stephanidis, "A student-centric intelligent classroom," *Ambient Intell.*, pp. 248–252, 2011.

[42] D. N. Rapp, "The value of attention aware systems in educational settings," *Comput. Hum. Behav.*, vol. 22, no. 4, pp. 603–614, Jul. 2006.

[43] R. G. Packard, "The control of 'classroom attention': A group contingency for complex behavior," *J. Appl. Behav. Anal.*, vol. 3, no. 1, pp. 13–28, 1970.

[44] D. R. Thomas, W. C. Becker, and M. Armstrong, "Production and elimination of disruptive classroom behavior by systematically varying teacher's behavior," *J. Appl. Behav. Anal.*, vol. 1, no. 1, pp. 35–45, 1968.

[45] R. A. Winett and R. C. Winkler, "Current behavior modification in the classroom: Be still, be quiet, be docile," *J. Appl. Behav. Anal.*, vol. 5, no. 4, pp. 499–504, 1972.

[46] K. Wilson and J. H. Korn, "Attention during lectures: Beyond ten minutes," *Teach. Psychol.*, vol. 34, no. 2, pp. 85–89, 2007.

[47] W. McKeachie and M. Svinicki, *McKeachie's teaching tips*. Cengage Learning, 2013.

[48] M. S. Young, S. Robinson, and P. Alberts, “Students pay attention! Combating the vigilance decrement to improve learning during lectures,” *Act. Learn. High. Educ.*, vol. 10, no. 1, pp. 41–55, 2009.

[49] D. A. Bligh, *What’s the Use of Lectures?* Intellect books, 1998.

[50] D. A. Sousa, *How the brain learns*. Corwin Press, 2016.

[51] P. J. Frederick, “The lively lecture—8 variations,” *Coll. Teach.*, vol. 34, no. 2, pp. 43–50, 1986.

[52] J. Horgan, “Lecturing for learning,” *Handb. Teach. Learn. High. Educ. Enhancing Acad. Pract. H Fry Ketteridge Marshall Abingdon RoutledgeFalmer*, pp. 75–90, 2003.

[53] J. Stuart and R. J. D. Rutherford, “Medical student concentration during lectures,” *The Lancet*, vol. 312, no. 8088, pp. 514–516, 1978.

[54] D. M. Bunce, E. A. Flens, and K. Y. Neiles, “How long can students pay attention in class? A study of student attention decline using clickers,” *J. Chem. Educ.*, vol. 87, no. 12, pp. 1438–1443, 2010.

[55] A. H. Johnstone and F. Percival, “Attention breaks in lectures.,” *Educ. Chem.*, vol. 13, no. 2, pp. 49–50, 1976.

[56] R. Stiefelhagen, J. Yang, and A. Waibel, “Estimating focus of attention based on gaze and sound,” in *Proceedings of the 2001 workshop on Perceptive user interfaces*, 2001, pp. 1–9.

[57] J. Ou, L. M. Oh, S. R. Fussell, T. Blum, and J. Yang, “Predicting visual focus of attention from intention in remote collaborative tasks,” *IEEE Trans. Multimed.*, vol. 10, no. 6, pp. 1034–1045, 2008.

[58] V. M. G. Barrios *et al.*, “AdELE: A framework for adaptive e-learning through eye tracking,” *Proc. IKNOW*, pp. 609–616, 2004.

[59] N. Thepvilojanapong *et al.*, “Evaluating people’s attention in the real world,” in *ICCAS-SICE, 2009*, 2009, pp. 3702–3709.

[60] P. Smith, M. Shah, and N. da Vitoria Lobo, “Determining driver visual attention with one camera,” *IEEE Trans. Intell. Transp. Syst.*, vol. 4, no. 4, pp. 205–218, 2003.

[61] R. Stiefelhagen, M. Finke, J. Yang, and A. Waibel, “From gaze to focus of attention,” in *Visual Information and Information Systems*, 1999, pp. 765–772.

[62] S. O. Ba and J.-M. Odobez, “Recognizing visual focus of attention from head pose in natural meetings,” *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 39, no. 1, pp. 16–33, 2009.

[63] K. Smith, S. O. Ba, J.-M. Odobez, and D. Gatica-Perez, “Tracking the visual focus of attention for a varying number of wandering people,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 7, pp. 1212–1229, 2008.

[64] R. Vertegaal, R. Slagter, G. Van der Veer, and A. Nijholt, “Eye gaze patterns in conversations: there is more to conversational agents than meets the eyes,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2001, pp. 301–308.

[65] S. Hommel and U. Handmann, “Realtime AAM based user attention estimation,” in *Intelligent Systems and Informatics (SISY), 2011 IEEE 9th International Symposium on*, 2011, pp. 201–206.

[66] D. A. Slykhuis, E. N. Wiebe, and L. A. Annetta, “Eye-tracking students’ attention to PowerPoint photographs in a science education setting,” *J. Sci. Educ. Technol.*, vol. 14, no. 5, pp. 509–520, 2005.

[67] H. Wang, M. Chignell, and M. Ishizuka, “Empathic tutoring software agents using real-time eye tracking,” in *Proceedings of the 2006 symposium on Eye tracking research & applications*, 2006, pp. 73–78.

[68] C. Merten and C. Conati, “Eye-tracking to model and adapt to user meta-cognition in intelligent learning environments,” in *Proceedings of the 11th international conference on Intelligent user interfaces*, 2006, pp. 39–46.

[69] J. L. Sibert, M. Gokturk, and R. A. Lavine, “The reading assistant: eye gaze triggered auditory prompting for reading remediation,” in *Proceedings of the 13th annual ACM symposium on User interface software and technology*, 2000, pp. 101–107.

[70] M. Prince, “Does active learning work? A review of the research,” *J. Eng. Educ.*, vol. 93, no. 3, pp. 223–231, 2004.

[71] L. A. Burke and R. Ray, “Re-setting the concentration levels of students in higher education: an exploratory study,” *Teach. High. Educ.*, vol. 13, no. 5, pp. 571–582, 2008.

[72] C. C. Bonwell and J. A. Eison, *Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports.* ERIC, 1991.

[73] A. J. Abramowitz, S. G. O’Leary, and L. A. Rosén, “Reducing off-task behavior in the classroom: A comparison of encouragement and reprimands,” *J. Abnorm. Child Psychol.*, vol. 15, no. 2, pp. 153–163, 1987.

[74] R. Hitz and A. Driscoll, “Praise in the Classroom.,” 1989.

[75] V. Santangelo, C. Ho, and C. Spence, “Capturing spatial attention with multisensory cues,” *Psychon. Bull. Rev.*, vol. 15, no. 2, pp. 398–403, 2008.

[76] A. K. Dey, T. Sohn, S. Streng, and J. Kodama, “iCAP: Interactive Prototyping of Context-Aware Applications,” in *Pervasive Computing*, 2006, pp. 254–271.

[77] M. W. Newman, “Now we’re cooking: Recipes for end-user service composition in the digital home,” 2006.

[78] D. Guinard, “Mashing Up Your Web-Enabled Home,” in *Current Trends in Web Engineering*, 2010, pp. 442–446.

[79] L. G. Massieu, “GALLAG Strip: A Mobile, Programming With Demonstration Environment for Sensor-Based Context-Aware Application Programming,” Arizona State University, 2012.

[80] “two forty four a.m.,” *Locale for Android*. [Online]. Available: https://www.twofortyfouram.com/. [Accessed: 07-Oct-2017].

[81] “Tasker for Android.” [Online]. Available: http://tasker.dinglisch.net/. [Accessed: 07-Oct-2017].

[82] “APIANT.” [Online]. Available: https://apiant.com/. [Accessed: 07-Oct-2017].

[83] R. AI, “Resonance | Predicting Human Behaviours.” [Online]. Available: https://www.resonance-ai.com/. [Accessed: 07-Oct-2017].

[84] “WigWag.” [Online]. Available: https://www.wigwag.com/. [Accessed: 06-Oct-2017].

[85] A. Rahmati, E. Fernandes, J. Jung, and A. Prakash, “IFTTT vs. Zapier: A Comparative Study of Trigger-Action Programming Frameworks,” *ArXiv Prepr. ArXiv170902788*, 2017.

[86] B. A. Nardi, *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA, USA: MIT Press, 1993.

[87] D. Fogli, R. Lanzilotti, and A. Piccinno, “End-user development tools for the smart home: a systematic literature review,” in *International Conference on Distributed, Ambient, and Pervasive Interactions*, 2016, pp. 69–79.

[88] J. M. Carroll, “Five reasons for scenario-based design,” *Interact. Comput.*, vol. 13, no. 1, pp. 43–60, 2000.

[89] A. Leonidis, D. Arampatzis, N. Louloudakis, and C. Stephanidis, “The AmI-Solertis System: Creating User Experiences in Smart Environments,” in *Proceedings of the 13th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, 2017.

[90] M. Korozi, A. Leonidis, M. Antona, and C. Stephanidis, “Lector: Towards Reengaging Students in the Educational Process inside Smart

Classrooms," presented at the 9th International Conference on Intelligent Human Computer Interaction, 2017.

[91] A. Leonidis *et al.*, "A glimpse into the ambient classroom," *Bull. IEEE Tech. Comm. Learn. Technol.*, vol. 14, no. 4, pp. 3–6, 2012.

[92] C. Stephanidis, A. A. Argyros, D. Grammenos, and X. Zabulis, "Pervasive Computing@ ICS-FORTH," in *Workshop Pervasive Computing@ Home*, 2008.

[93] C. Savvaki, A. Leonidis, G. Paparoulis, M. Antona, and C. Stephanidis, "Designing a technology–augmented school desk for the future classroom," in *International Conference on Human-Computer Interaction*, 2013, pp. 681–685.

[94] A. Leonidis, G. Margetis, M. Antona, and C. Stephanidis, "ClassMATE: enabling ambient intelligence in the classroom," *World Acad. Sci. Eng. Technol.*, vol. 66, pp. 594–598, 2010.

[95] A. Ntagianta, M. Korozi, A. Leonidis, M. Antona, and C. Stephanidis, "CognitOS: A Student-Centric Working Environment for an Attention-Aware Intelligent Classroom," in *Proceedings of the 20th International Conference on Human-Computer Interaction*, 2018, (under submission).

[96] E. Stefanidi, M. Doulgeraki, M. Korozi, A. Leonidis, and M. Antona, "Designing a Teacher-Friendly Editor for Configuring the Attention-Aware Smart Classroom," in *HCI International 2016 – Posters' Extended Abstracts*, 2016, pp. 266–270.

[97] E. Stefanidi, M. Korozi, A. Leonidis, M. Doulgeraki, and M. Antona, "Educator-Oriented Tools for Managing the Attention-Aware Intelligent Classroom," in *In the Proceedigns of The 10th International Conference on Mobile, Hybrid, and On-line Learning (eLmL 2018)*, Rome, 2018, p. (under submission).

[98] A. K. Dey, G. D. Abowd, and D. Salber, “A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications,” *Hum.-Comput. Interact.*, vol. 16, no. 2, pp. 97–166, 2001.

[99] K. Henricksen and J. Indulska, “Developing context-aware pervasive computing applications: Models and approach,” *Pervasive Mob. Comput.*, vol. 2, no. 1, pp. 37–64, 2006.

[100] X. Hong, C. Nugent, M. Mulvenna, S. McClean, B. Scotney, and S. Devlin, “Evidential fusion of sensor data for activity recognition in smart homes,” *Pervasive Mob. Comput.*, vol. 5, no. 3, pp. 236–252, 2009.

[101] D. Zhang, H. Huang, C.-F. Lai, X. Liang, Q. Zou, and M. Guo, “Survey on context-awareness in ubiquitous media,” *Multimed. Tools Appl.*, vol. 67, no. 1, pp. 179–211, 2013.

[102] K.-A. Hwang and C.-H. Yang, “Automated Inattention and Fatigue Detection System in Distance Education for Elementary School Students.,” *J. Educ. Technol. Soc.*, vol. 12, no. 2, 2009.

[103] R. Sylwester, “How emotions affect learning.,” *Educ. Leadersh.*, vol. 52, no. 2, pp. 60–65, 1994.

[104] “Handlebars.js: Minimal Templating on Steroids.” [Online]. Available: http://handlebarsjs.com/. [Accessed: 17-Oct-2017].

[105] “Mustache (template system),” *Wikipedia*. 01-Aug-2017.

[106] M. Korozi, M. Antona, A. Ntagianta, A. Leonidis, and C. Stephanidis, “LECTORSTUDIO: CREATING INATTENTION ALARMS AND INTERVENTIONS TO REENGAGE THE STUDENTS IN THE EDUCATIONAL PROCESS,” in *Proceedings of the 10th annual International Conference of Education, Research and Innovation*, 2017.

[107] N. Babich, “Wizard Design Pattern,” *UX Planet*, 07-Apr-2017. [Online]. Available: https://uxplanet.org/wizard-design-pattern-8c86e14f2a38.

[108] “Heuristic Evaluation: How-To: Article by Jakob Nielsen.” [Online]. Available: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/.

[109] “Why You Only Need to Test with 5 Users.” [Online]. Available: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

[110] “Severity Ratings for Usability Problems: Article by Jakob Nielsen.” [Online]. Available: https://www.nngroup.com/articles/how-to-rate-the-severity-of-usability-problems/.

[111] “Thinking Aloud: The #1 Usability Tool,” *Nielsen Norman Group*. [Online]. Available: https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/. [Accessed: 31-Oct-2017].

[112] A. S. for P. Affairs, “System Usability Scale (SUS),” 06-Sep-2013. [Online]. Available: /how-to-and-tools/methods/system-usability-scale.html. [Accessed: 30-Oct-2017].

[113] “MeasuringU: Measuring Usability with the System Usability Scale (SUS).” .

[114] S. Wexler, J. Shaffer, and A. Cotgreave, *The Big Book of Dashboards: Visualizing Your Data Using Real-World Business Scenarios*. John Wiley & Sons, 2017.

[115] U. Fayyad, G. G. Grinstein, and A. Wierse, Eds., *Information Visualization in Data Mining and Knowledge Discovery*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[116] R. M. Felder and R. Brent, “Understanding student differences,” *J. Eng. Educ.*, vol. 94, no. 1, pp. 57–72, 2005.

[117] J. Nielsen, *Usability engineering*. Elsevier, 1994.

[118] “Severity Ratings for Usability Problems: Article by Jakob Nielsen.” [Online]. Available: https://www.nngroup.com/articles/how-to-rate-the-severity-of-usability-problems/. [Accessed: 09-Oct-2017].

[119] B. Reeder and A. David, “Health at hand: A systematic review of smart watch uses for health and wellness,” *J. Biomed. Inform.*, vol. 63, no. Supplement C, pp. 269–276, Oct. 2016.

[120] E. Sykianaki, M. Korozi, A. Leonidis, M. Antona, and C. Stephanidis, “CaLmi: Stress Management in Intelligent Homes,” in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications*, Athens, Greece, 2018, (under submission).

Appendix A
Publications

1. E. Stefanidi, M. Doulgeraki, M. Korozi, A. Leonidis, and M. Antona, “Designing a Teacher-Friendly Editor for Configuring the Attention-Aware Smart Classroom”, in *HCI International 2016 – Posters’ Extended Abstracts*, 2016, pp. 266–270.

2. M. Korozi, M. Antona, A. Ntagianta, A. Leonidis, and C. Stephanidis, “LECTORSTUDIO: CREATING INATTENTION ALARMS AND INTERVENTIONS TO REENGAGE THE STUDENTS IN THE EDUCATIONAL PROCESS”, in Proceedings of the 10th annual International Conference of Education, Research and Innovation, 2017.

3. M. Korozi, A. Leonidis, M. Antona, and C. Stephanidis, “Lector: Towards Reengaging Students in the Educational Process inside Smart Classrooms,” presented at the 9th International Conference on Intelligent Human Computer Interaction, 2017.

4. Ntagianta, M. Korozi, A. Leonidis, M. Antona, and C. Stephanidis, “CognitOS: A Student-Centric Working Environment for an Attention-Aware Intelligent Classroom”, in *Proceedings of the 20th International* Conference on Human-Computer Interaction, 2018, (under submission).

5. E. Stefanidi, M. Korozi, A. Leonidis, M. Doulgeraki, and M. Antona, “Educator-Oriented Tools for Managing the Attention-Aware Intelligent

Classroom", In the Proceedigns of The 10th International Conference on Mobile, Hybrid, and On-line Learning (eLmL 2018), Rome, 2018.

6. E. Stefanidi, M. Korozi, A. Leonidis, and M. Antona, "Programming Intelligent Environments in Natural Language: An Extensible Interactive Approach", In the Proceedigns of The PErvasive Technologies Related to Assistive Environments (PETRA 2018) (eLmL 2018), Corfu, 2018 (under submission).

Appendix B
Acronyms

AmI Ambient Intelligence

BR Behavior Reasoner

ICT Information and Communication Technologies

IM Intervention Manager

IOT Internet of Things

IP Interventions' Pool

LC Learning Component

SAL Sensor Abstraction Layer

SUS System Usability Scale

TAP Trigger-action programming

UI User Interface

Appendix C
User-Based Evaluation Scenario Tasks

Task 1. You want to create a new behavior rule stating that:
"*Independently of the current course or activity, a user is considered to be talking when the captured sound levels from his / her microphone range between 75-85 db*".

- Each completed step of the wizard should change the color of the circle to something else to show that it is saving the information each step of the way
- A next button should be available, permitting users to navigate among the wizard steps.
- When a rule list is empty the filters should not be visible since the user might get confused
-

Task 2. You want to create a new behavior rule for identifying a user with Tachycardia.

Tachycardia is defined as "*Heart rate between 100bpm and 200 bpm signifies Tachycardia*"

However, the system does not integrate information about the heart rate physiological property. Integrate this property by translating the data received from the HearRateService, and populating an attribute named Rate.

Now try to insert the tachycardia rule.

Task 3. Based on the behavior that you modeled on step 1 (TALKING), you want to create a trigger rule:

"*Excluding the music course, in every other case a student is considered to be chatting when the teacher is talking, and he /she is talking or whispering*".

Task 4. Similarly, you want to create a trigger rule that fires when the entire classroom is affected:

"*Excluding the music course, in every other case the entire classroom is considered to be chatting when the teacher is talking, and the 30% of students are talking or whispering*".

Task 5. You want to update the actor student and add to the monitored behaviors the behavior Tachycardia that you created before.

Task 6. You want to integrate the following artifact to act as an intervention host:

Student's Watch (Type: WATCH, Service: VibrationService)

Task 7. Now, integrate the "encourage student" application that runs on the student's watches and acts as an intervention.
When employing this intervention the teacher can modify the frequency (times / hour) of the messages that the application delivers as well as the appearance duration (secs) of each message.

Task 8. Finally, create the following intervention rule:
"*If the entire classroom is chatting, send an alarm to the teacher's watch, start a multimedia presentation with content related to the courses syllabus and dim the classroom lights to 20%.*"

Appendix D
Motivating Scenarios for the Intelligent Home Use Case

Domestic Life

This scenario describes the activities of a three-member family in their intelligent home, from the time they wake up in the morning until they go to work and school. The family members are (i) John (father), who works at a big firm near his home, (ii) Mary (mother), who is unemployed, and (iii) Jimmy, who is an elementary school student.

On Monday at 7:00 the alarm in the parents' bedroom notifies them to wake up. However, five minutes later the system identifies that everyone is still asleep; given that it is a working day and it is important to get up in time, the system intervenes by turning on the radio and opening the blinds so as to let natural light into the room. Mary quickly realizes that they have overslept; she nudges John and gets out of bed immediately so as to wake up Jimmy too. John heads to the bathroom to shave and brush his teeth. At this point, the system knows that since they didn't wake up in time, John has limited time to drop Jimmy at school before going to work; hence as soon as he starts shaving the "smart mirror" displays information regarding the traffic towards Jimmy's school. This helps John to select the most appropriate route, avoiding jammed roads.

At the same time, Mary is at the kitchen preparing breakfast for Jimmy. Since today Jimmy has limited time available before leaving for school, the system suggests to Mary to make cereal, which require less preparation time compared to his regular breakfast, namely omelet. As soon as Jimmy finishes up his breakfast, father and son leave the house at around 8:00 AM.

Now that Mary is alone at the house, she finds the time to review her notes for a job interview that she has to attend at 9:30AM. Unfortunately, she starts becoming worried about the interview; the system, being aware of her schedule, suggests to take some time to do some yoga exercises in order to relieve her stress. Mary, knowing that there is enough time to exercise and to get ready for the interview she accepts that intervention.

Ambient Assisted Living

This scenario describes the activities of Mary's mother, Helen, in her intelligent home. Helen is 70 years old and leaves alone, which is quite challenging considering that she suffers from arthritis and owns a wheelchair to help her move around the house.

Yesterday, Helen visited her physician for her scheduled arthritis injections. Surprisingly, this morning she wakes up being unable to bend her knees and consequently she cannot get on her wheelchair. The system understands that Helen is awake for some time but does not get of the bed. Given that Mary is probably on the way to her interview, the system decides to inform Helen's secondary caregiver, her son George. In the meanwhile, in order to help Helen feel calm, the bedroom speakers play a message saying that George has been notified. As soon as George receives a text message explaining the situation, he immediately decides to visit his mother; indeed, Helen is still in bed and explains to him that she cannot move by herself. George calls her physician, who calms him down explaining that this is a common phenomenon after knee injections, and suggests that Helen takes a specific medication. Before

hanging up, the physician reminds George that his mother should not skip her rehabilitation exercises.

George, before returning home, helps his mother in getting in the wheelchair and informs her about the physician's advices. Helen, continues her daily activities but seems to ignore the system's suggestions for exercising. Getting off the chair and walking a short distance, is really important for an individual with Helen's condition. To this end, as soon as the system realizes that she has deviated from her rehabilitation routine, a text message is sent to her physician and her caregivers.