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

AmI-Solertis

Online Platform to Support Intelligent Behaviors in Ambient Intelligence Environments

by

Asterios Leonidis

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

AmI-Solertis: Online Platform to Support Intelligent Behaviors in Ambient Intelligence Environments

PhD Dissertation Presented

by **Asterios Leonidis**

in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

APPROVED BY:

Author: Asterios Leonidis

Supervisor: Constantine Stephanidis, Professor, University of Crete

---

Committee Member: Dimitrios Plexousakis, Professor, University of Crete

---

Committee Member: Anthony Savidis, Professor, University of Crete

---

Committee Member: Konstantinos Magoutis, Assistant Professor, University of Ioannnina

---

Committee Member: Dimitrios Tzovaras Researcher A', CERTH-ITI

Committee Member: Eleni Eythymioy, Researcher A', ILSP/ATHENA RC

Committee Member: Margherita Antona, Researcher B', FORTH-ICS

Department Chairman: Angelos Bilas, Professor, University of Crete

Heraklion, November 2017

to my wife Maria and 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 Dimitris Plexousakis 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 my colleagues Dimitris Arabatzis, Nikos Anifantis and Nikos Louloudakis for their contribution in the design and implementation of the AmI-Solertis core, to Achilleas Tsiolkas for his contribution in the development of the Scaffolding tools and the Home Automation Plugin, to Vaggelis Kalligiannakis for his contribution in the development of the Social components of AmI-Solertis and to M.Sc. Giorgos Nikitakis for his contribution in the development of the Datastore component.

In addition, I would like to express my deepest gratitude to my lovely wife Maria Korozi who as a colleague assisted me throughout the design and development process of this work and as my spouse equipped me with all the strength and courage I needed to pursue my goals. Yet, the biggest appreciation goes to my daughter Anastasia who is the person that inspires me to aim high and keep going. I also want to thank my parents Dimitris and Anastasia, and my parents in law Costas and Dimitra, for their unconditional support and love, especially during the final months of my dissertation.

Finally, I need to thank the Department of Computer Science (CSD) of the University of Crete, and the Human Computer Interaction (HCI) Laboratory of the Institute of Computer Science (ICS) of the Foundation for Research and Technology - Hellas (FORTH) for equipping me with the means and creating a highly motivational working environment to achieve my research goals.

Abstract

Ambient Intelligence (AmI) envisages that technology and information flow around the physical environment, and objects are enhanced with computer technology to communicate, share information and collaborate with other technological devices in an intelligent fashion, thus forming a ubiquitous and pervasive computing landscape, where implicit interaction and continuous cooperation is becoming the norm of computer supported activities. AmI constitutes an emerging market that is forecasted to exhibit exponential growth in the forthcoming years. Yet, AmI systems can only maximize their efficiency, extensibility and adaptation to the needs of their users, if they are programmable.

This thesis proposes the AmI-Solertis framework which empowers users to create behavior scenarios in AmI by reviewing and modifying the "high-level business logic" of an intelligent environment through an advanced programming platform and an accompanying chat-bot agent. In addition to the framework, the AmI-Solertis Studio offers a complete suite of tools allowing management, programming, testing and monitoring of all the individual artifacts (i.e., services, hardware modules, software components, etc.) of the overall AmI Environment.

From an engineering perspective, the AmI-Solertis framework: (i) introduces a unified **Hybrid Communication protocol** which supports synchronous, asynchronous and event-based communication, (ii) **unifies the definition and introduction** of new devices, services and software components, (iii) facilitates the **integration and usage** of heterogenous services in a standardized -yet agnostic- manner, (iv) delivers a **scripting mechanism** that can dynamically adapt the execution flow and govern the entire AmI environment'As behavior, and (v) offers a **standard library of tools** (i.e., Analytics and History, Fault Tolerance, Storage Management, Common Utilities) that developer can use.

From a user perspective, the AmI-Solertis Studio constitutes a web-based Integrated Development Environment (IDE) and Control Center that can be used as a creative tool for designing

user experiences in intelligent environments. In particular, the studio: (i) supports the **entire development life-cycle** of an AmI system, (ii) **empower users** to explore and adapt software to their personal needs -through **user-friendly scripting environment**- or develop new innovative applications, (iii) simplifies service **discovery, definition, and management**, (iv) **scaffolds** typical designs, (v) offers **multiple visual representations**, (vi) provides **testing** facilities, (vii) facilitate **collaboration** between users, (viii) assists **real-time management** of the AmI environment, and (ix) delivers an **AmI-Solertis virtual agent in the form of a chat-bot**, that can communicate with the end-users via a natural language textual interface in order to help them accomplish numerous orchestration-related tasks.

Keywords: Ambient Intelligence, Ubiquitous Computing, User Programming, Programming Framework, Computer systems organization, Human-centered computing, Software engineering.

Περίληψη

Η Διάχυτη Νοημοσύνη (ΔΝ) οραματίζεται περιβάλλοντα στα οποία η τεχνολογία και οι πληροφορίες 'ρέουν' συνεχώς, και τα φυσικά αντικείμενα είναι επαυξημένα με υπολογιστική τεχνολογία ώστε να μπορούν να επικοινωνούν, να ανταλλάσσουν δεδομένα και να συνεργάζονται με έναν 'ευφυή' τρόπο, δημιουργώντας έτσι ένα διευρυμένο υπολογιστικό τοπίο, στο οποίο η έμμεση αλληλεπίδραση και η διαρκής συνεργασία μεταξύ του περιβάλλοντος και των χρηστών επιτρέπουν την υπολογιστική υποστήριξη καθημερινών δραστηριοτήτων. Η Διάχυτη Νοημοσύνη αποτελεί μια ραγδαία αναπτυσσόμενη οικονομία η οποία προβλέπεται να αυξηθεί εκθετικά στα επόμενα χρόνια. Ωστόσο, τα διάφορα συστήματα ΔΝ θα πρέπει να είναι εύκολα προγραμματίσιμα, ώστε να μεγιστοποιήσουν την αποτελεσματικότητα τους, την επεκτασιμότητα τους και την προσαρμοστικότητα τους στις ανάγκες των τελικών χρηστών.

Προς αυτή την κατεύθυνση, η παρούσα διατριβή προτείνει την τεχνολογική πλατφόρμα AmI-Solertis, η οποία επιτρέπει στους χρήστες της να καθορίσουν την 'ευφυή' συμπεριφορά του περιβάλλοντος ΔΝ, δημιουργώντας μικρό-προγράμματα (AmI scripts) μέσω μιας διαδικτυακής προγραμματιστικής πλατφόρμας (AmI Solertis Studio) και ενός συνοδευτικού ψηφιακού βοηθού (chat-bot agent). Επιπρόσθετα, μια πλήρης σουίτα εργαλείων διευκολύνει την διαχείριση, τον προγραμματισμό, τον έλεγχο και την παρακολούθηση σε πραγματικό χρόνο των επιμέρους υποδομών του συνολικού περιβάλλοντος ΔΝ (π.χ., υπηρεσίες, συσκευές, λογισμικό, κλπ.).

Από τεχνολογικής σκοπιάς, η πλατφόρμα AmI-Solertis: (i) προτείνει ένα **υβριδικό πρωτόκολλο επικοινωνίας** το οποίο υποστηρίζει τόσο σύγχρονη όσο και ασύγχρονη αλληλεπίδραση (π.χ., events) μεταξύ των διαφόρων συστημάτων, (ii) **ενοποιεί τον ορισμό και την εισαγωγή** νεών συσκευών, υπηρεσιών και λογισμικού ανεξαρτήτως του τύπου τους, (iii) διευκολύνει την **ενσωμάτωση και χρήση** ετερογενών υπηρεσιών

μέσω ενός προτυποποιημένου -και ταυτόχρονα τεχνολογικά ανεξάρτητου (agnostic)- μηχανισμού, (iv) παρέχει ένα **σύστημα μικρό-προγραμματισμού (scripting)** που επιτρέπει την δυναμική προσαρμογή και έλεγχο της συμπεριφοράς του συνολικού περιβάλλοντος ΔΝ, και (v) προσφέρει μια **πρότυπη προγραμματιστική βιβλιοθήκη εργαλείων** προς τους τελικούς προγραμματιστές για την διευκόλυνση τους (π.χ., Δεδομένα και ιστορικό χρήσης, Αντιμετώπιση 'βλαβών' σε πραγματικό χρόνο, Κατανεμημένος χώρος αποθήκευσης δεδομένων).

Από τη σκοπιά των χρηστών του, το AmI-Solertis Studio αποτελεί ένα Διαδικτυακό Περιβάλλον Προγραμματισμού (Integrated Development Environment (IDE)) και ένα Κέντρο Διαχείρισης το οποίο μπορεί να χρησιμοποιηθεί ως εργαλείο 'δημιουργίας και σχεδιασμού' εμπειριών χρήσης σε περιβάλλοντα ΔΝ. Συγκεκριμένα, το περιβάλλον: (i) υποστηρίζει τους προγραμματιστές **καθ΄ όλη την διάρκεια ανάπτυξης** ενός μικρο-προγράμματος ΔΝ, (ii) **επιτρέπει στους χρήστες** να εξερευνήσουν και να προσαρμόσουν μικρό-προγράμματα ΔΝ στις δικές τους προσωπικές ανάγκες ή να αναπτύξουν νέες καινοτόμες εφαρμογές μέσω ενός **φιλικού προς τους χρήστης περιβάλλοντος προγραμματισμού**, (iii) απλοποιεί **τον εντοπισμό**, **τον ορισμό και την διαχείριση** υπηρεσιών ΔΝ, (iv) **δημιουργεί πρότυπα** project skeletons κατάλληλα για περιβάλλοντα ΔΝ, (v) παρέχει **πολλαπλές εναλλακτικές αναπαραστάσεις**, (vi) προσφέρει **βασικές υπηρεσίες testing**, (vii) διευκολύνει **την συνεργασία** μεταξύ των χρηστών, (viii) υποβοηθά **σε πραγματικό χρόνο την διαχείριση** ενός περιβάλλοντος ΔΝ, και (ix) εισάγει τον **ψηφιακό βοηθό του AmI-Solertis στην μορφή ενός chat-bot**, που χρησιμοποιεί φυσική γλώσσα για να ανταλλάσσει μηνύματα με τους τελικούς χρήστες ώστε να τους βοηθήσει να εκτελέσουν διάφορες βασικές εργασίες διαχείρισης ή προγραμματισμού.

Λέξεις Κλειδιά: Διάχυτη Νοημοσύνη, Προγραμματισμός Χρήστη, Αλληλεπίδραση Ανθρώπου-Υπολογιστή, Προγραμματιστική Υποδομή, Οργάνωση Προγραμματιστικών Συστημάτων, Τεχνολογία Λογισμικού.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 General Objective

The rapid evolution of technology, along with the miniaturization of microprocessors enabled computing power to be embedded in everyday objects and spaces and increased people's access to multiple computers (which does not necessarily just mean owning both a Personal Computer (PC) and a laptop). On top of continuous technological advancement, the expectations of the users have been growing due to their experiences with computers, transforming them into to the most-used tool on a daily basis [82]. Concomitantly with this difference in the way society perceives technology, millions of innovative applications and services continuously emerge on almost a daily basis [78].

This modern way of living, where technology and information are flowing around the physical environment, led to the emergence of the Ambient Intelligence (AmI) paradigm, where physical objects are enhanced with computer technology to communicate, share information and collaborate with other technological devices in an intelligent fashion. AmI is a prominent dimension of Information and Communication Technology (ICT) that transforms traditional environments to technologically enhanced "intelligent" ones, giving rise to a ubiquitous and pervasive computing landscape, where implicit interaction and continuous cooperation is becoming the norm of computer supported activities [237], forming a highly emerging market [232] that is forecasted to exhibit exponential growth in the forthcoming years [79, 139, 256].

In order to maximize the efficiency, extensibility and adaptation to the needs of their users,

AmI systems need to be programmable. In fact, their programmers are not expected to be only professional developers, but also inexperienced end-users who can either modify the behavior of the system based on their current needs or extend its intelligence even further to address their future necessities. To that end, this thesis proposes the AmI-Solertis framework and authoring studio, which empowers users to create behaviors scenarios by reviewing and modifying the "high-level business logic" of an AmI environment through an advanced programming platform and an accompanying chat-bot agent. In addition, the AmI-Solertis system offers a complete suite of tools allowing management, programming, testing and monitoring of all the individual artifacts (i.e., services, hardware modules, software components, etc.) of the overall AmI Environment.

1.2 Motivation and Research Questions

AmI environments are powered by many different technologies that coexist and cooperate in order to enhance the surrounding environment with the appropriate computational means, so as to make it sensitive the presence of people and objects, and able to proactively and intelligently react to human needs. In this context, horizontal integration between application layer protocols is needed, as the respective AmI components and applications have to communicate with each other in order to collaborate. Therefore, in order to aid developers in defining the intelligent behavior of AmI environments, a holistic approach that incorporates both an appropriate communication middleware and a unified programming platform is required. The research questions that this thesis aims at tackling are the following:

- Since AmI environments integrate diverse technologies and solutions, can a method unify the definition, introduction and management of new services that control devices and software components?

- With respect to the communication middleware, literature [33, 173, 259] converges that the basic functional components should facilitate: (i) Interoperation, (ii) Context detection, (iii) Device discovery and management, (iv) Security and privacy, and (v) Data volume Management. Can a communication middleware be real-time, scalable, cross-platform, rely on well-established technologies (e.g., REST, JSON), and support synchronous, asynchronous

and event-based interaction between services in the context of AmI environments?

- An AmI environment should build upon advances in sensors and sensor networks, actuators, pervasive computing, and artificial intelligence [82], and requires the technological layer to be distributed [317], embedded and hidden in the surroundings [48] so as to create a pervasive environment [393]. It should be sensitive, adaptive and responsive to the presence of people and objects [48], augment the activities through smart non-explicit assistance and preserve security, privacy and trustworthiness. Moreover, it should utilize contextual information when needed to behave intelligently [7], by customizing its requirements or forecasting behaviors [308]. Finally, an AmI environment should transparently interact with the user either passively by observing and trying to interpret the actions and the intentions, or actively by learning the preferences and adapting the behavior accordingly to improve the quality of life [317]. Currently, various tools exist to empower the creation of services and applications for AmI environments. Can a tool assist developers in building software that aims to satisfy these requirements? Can it support the entire development life-cycle of an AmI system (i.e., analysis, design, implementation, testing, deployment, maintenance and disposal)?

- Since End-User Development (EUD) is an emerging paradigm and over the next few years, the goal of AmI systems and services will evolve from just making them easy to use, to making them easy to develop by end users, it will be imperative to empower users to explore the initial system's functionality, adapt software to their personal needs and develop new innovative applications. Can that be accomplished in a user-friendly manner?

1.3 The Approach

The AmI-Solertis system is built using a micro-service architecture style, hence its business capabilities are implemented and offered through a collection of loosely coupled services. By applying such an architecture style, the core system is flexible and scalable to be used as a backbone [266] across a wide range of ubiquitous systems [74] and intelligent environments with diverse objectives (e.g., compose a new service using two existing ones, define the behavior of a smart hotel

room [220], control a smart home, build an intelligent management system for a smart city [40]).

The core system is composed of autonomous components that interoperate over the network using a variety of communication protocols and channels. Every component exposes its functionality either via a public REST API in the form of stateless operations through which it can to receive incoming calls, while it can communicate its intention to the AmI ecosystem by emitting appropriate events via the AmI-Solertis Event Federator. To further enhance interoperability and discoverability, the OpenAPI Specification (OAS) [5] is used to formally describe every API. AmI-Solertis encapsulates the complexity of configuring and performing remote calls in automatically generated proxies that eliminate the difficulties of distributed programming by (i) masking remote operations into local methods, and (ii) enabling consumers to register their interest to events coming from a remote component without specifying any details about the underlying topology and infrastructure. In addition to code minimization, proxies also empower AmI-Solertis to dynamically adapt and adjust the invocation process in order to address emerging requirements such as re-routing a call to a replicated host to achieve load balancing, immediately terminating a call if the remote endpoint is unavailable, intercepting a call and logging relevant QoS-related metrics, replacing a target endpoint with another that offers semantically similar functionality.

AmI-Solertis enables fast, easy and error-free integration of external AmI artifacts (i.e., services) independently of their kind (i.e., back-end, front-end or mixed services that follow the SaaS paradigm). Moreover, it supports the creation of AmI scripts that define the behavior of the technological facilities (i.e., business logic) towards creating pervasive, intelligent and personalized environment experiences by combining multiple components. In addition to AmI components management (i.e., AmI artifacts or scripts), AmI-Solertis also assists developers when building their AmI scripts or integrating their AmI artifacts in the ecosystem. Particularly, it automatically: (i) *generates* the necessary *auxiliary files* (e.g., service formal specification, configuration file template, service-specific methods and definition of types) (ii) *prepares the executable* files, (iii) creates the *installation packages*, (iv) *deploys* it on the designated AmI-Solertis compliant hosts, (v) orchestrates the *execution process*, and (vi) facilitates *real-time monitoring and control*.

The AmI-Solertis framework consists of various light-weight service-based components that constitute the following core modules:

- **Universal AmI Artifacts and Scripts Repository**: copes with artifacts and script management and enforces the common intercommunication scheme. It includes: (a) the components that inspect the installed services and extract their Applications Programming Interface (API) specification in OAS scheme, (b) the components that generate the appropriate software bridges, based on a OAS specification, to harmonize the programming approach independently of the service type; (c) the metadata storage mechanisms; (d) the service discovery tools and (e) the installation wizard that installs new services.

- **Packaging and Deployment Manager**: automates the overall packaging and deployment process for any AmI-Solertis compliant artifact or script. It consults the Universal Repository in order to assist the developer to properly configure and install the desired components in one or more physical or virtual hosts.

- **Execution and Health Monitoring**: enables remote administration of AmI-Solertis components of any kind and monitors the overall health status of the system; if necessary, it takes corrective actions to ensure a proper Quality of Service (QoS). Interventions refer to the modification of the execution rules to compensate for under-performing components via custom automatically-generated circuit-breakers, or missing services via on-the-fly composition of semantically equivalents (at a proxy level).

- **Analytics and History Logs**: record data and key events that happened within the environment. It can be used by AmI-Solertis compliant services or applications to store their own content. Mainly though, these data are analyzed by the core in order to evaluate the performance of the various artifacts and generate insights about potential optimizations.

- **Artificial Intelligence**: provides access to well-known machine learning services [290]. Behavior scripts authored via the web-IDE can use them to create scenarios that employ artificial intelligence principles to enhance the overall user experience in the intelligent space, whereas the AmI-Solertis core employs them to facilitate the internal decision making processes.

- **Universal Datastore**: offers a universal mechanism to store both document-based and binary data (i.e., blobs [335]) regarding the AmI ecosystem and simplify their exchange with 3^{rd}-party services and systems [349].

- **AmI-Solertis Studio**: consists of the various graphical components through which end-users can interact with the system in order to explore the available artifacts and program the behavior of an Intelligent Environment. In addition to the editing and project management tools, it includes a testing suite that can validate the behavior of a script and auxiliary facilities that can be used to define optimizations or fault-tolerance strategies.

- **Social Features**: AmI-Solertis users can collaborate to share their experiences and best practices. To leverage the collective experience of its user community, the system facilitates the discovery of new and noteworthy content by promoting featured, popular and trending services based on their impact in the community. In addition, via user profiles the overall experience is personalized [195] by moderating the displayed content (e.g., recommendations, context-sensitive lists of recently viewed items for quick reference, etc.).

1.4 Contributions of this Dissertation

The key contributions of this thesis fall into the domains[1] of: Computer systems organization, Human-centered computing and Software engineering. From a broader engineering perspective, AmI-Solertis delivers an advanced technological framework that:

- Reflecting on the benefits of asynchronous and event-based communication, introduces a unified **Hybrid Communication protocol**. In particular, the proposed communication middleware: (i) is real-time, scalable and cross-platform, (ii) relies on well-established technologies (i.e., REST, Pub/Sub) (iii) supports synchronous, asynchronous and event-based communication, (iv) facilitates resource discovery, management and update, (v) enables service composition and self-exposition, (vi) simplifies service updates and deployment, (vii) streamlines the introduction of existing external services, (viii) empowers users to create versatile AmI applications targeted to their context, (ix) supports self-* properties (e.g.,

[1] https://dl.acm.org/ccs/ccs.cfm

self-adaptive), (x) is developer-friendly, and finally (xi) allows the introduction of new components that extend its functionality.

- **Unifies** the definition and introduction of new services that control devices and applications.

- Facilitates the **integration and usage** of heterogenous services in a standardized -yet agnostic- manner.

- Delivers a **scripting mechanism** that can dynamically adapt the execution flow and govern the entire AmI environment'As behavior.

- Offers a **standard library with tools** (i.e., Analytics and History, Fault Tolerance, Storage Management, Common Utilities) to any developer who wants to: (i) create a new AmI artifact or script, (ii) implement new core functionality for the AmI-Solertis framework, or (iii) build external tools that interoperate with AmI-Solertis to introduce new features.

From a user perspective, AmI-Solertis adopts solutions from the broader domains of Ambient Intelligent and agent-based computing as the means to implement the AmI-Solertis Studio, which constitutes a web-based Integrated Development Environment (IDE) and Control Center that can be used as a creative tool for designing user experiences in Intelligent Environments. In particular, AmI-Solertis Studio:

- Supports the **entire development life-cycle** of an AmI system (i.e., analysis, design, implementation, testing, deployment, maintenance and disposal).

- **Empower users** to explore the initial system's functionality, adapt software to their personal needs or develop new innovative applications.

- Establishes a **user-friendly scripting environment** through which end-users with varying degrees of expertise and development skills can easily create or modify the high-level behavior of a Intelligent Environments.

- Simplifies service **discovery, definition, and management**.

- **Scaffolds** projects (i.e., file structure, boiler-plate code) that can be used as guides on how to build external AmI artifacts.
- Offers **multiple visual representations**.
- Provides **testing** facilities.
- Facilitates **collaboration** between users and expertise sharing.
- Assists **real-time management** of the AmI environment (i.e, installation and deployment of new components, monitoring of existing infrastructure).
- Delivers **AmI-Solertis virtual agent in the form of a chat-bot**, that can communicate with the end-users via a natural language textual interface in order to help them accomplish numerous orchestration-related tasks.

1.5 Thesis Outline

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

- Chapter 2 will present the concept of Ambient Intelligence (AmI) and the contributing technologies that promote its realization. It will report related work towards building an appropriate communication framework and middleware, and will conclude by presenting and discussing the relevant solutions that enable End-User Development (EUD) in order to define the intelligent behavior of AmI environments.
- Chapter 3 will introduce a variety of use cases that motivate this work and outline the key functional and non-functional requirements for the AmI-Solertis framework.
- Chapter 4 will describe in details the microservice-based architecture of AmI-Solertis, its core components and its holistic approach that incorporates both an appropriate communication middleware and a unified programming platform towards aiding developers in managing the intelligent behavior of AmI environments.
- AmI-Solertis in addition to its core components, relies on a number of auxiliary tools and libraries that are necessary for its proper operation. Chapter 5 will introduce the AmI-Solertis

Standard Library that is available to any developer that wants to: (i) create a new AmI artifact or script, (ii) implement new core functionality for the AmI-Solertis framework, or (iii) build external tools that interoperate with AmI-Solertis to introduce new features.

- Chapter 6 will present in details the various frontend facilities of the AmI-Solertis Studio. It will provide an overview of its functionality, and then will report the major components through which developers interact with the AmI-Solertis framework so as to create, deploy and monitor software that controls the behavior of an AmI environment.

- Chapter 7 will report evaluation findings that highlight the benefits of AmI-Solertis when used to define the intelligent behavior of AmI environments.

- Chapter 8 will summarize this work and outline the short- and long-term future directions of AmI-Solertis, and offer some closing reflections about the promising potentials of Ambient Intelligence.

- Appendix A reports the publications resulted from this thesis.

- Appendix B reports the acronyms used in this thesis.

Chapter 2

Background and Related Work

This chapter will firstly present the Ambient Intelligence (AmI) concept along with the contributing technologies that promote its realization. Next, it will report and discuss the approaches, methodologies and techniques that have been introduced over the years towards building an appropriate communication framework and middleware that facilitates interoperability among the heterogenous, distributed applications and artifacts residing in such environments. Finally, the chapter will conclude by presenting and discussing the relevant solutions that enable End-User Development (EUD) in order to define (i.e., program) the intelligent behavior of AmI environments.

2.1 Ambient Intelligence

2.1.1 AmI Definition

Computer Science, despite being a relative new scientific domain compared to traditional sciences such as Mathematics or Physics, has gone through rapid and yet important transformations during the first decades of its existence, and continues to grow at an increasing rate [277]. The driving force behind that progress in the field of ICT is a combination of the long-term trend in microelectronics (as Gordon Moore already roughly stated back in 1965 [260] and still holds true with astonishing accuracy and consistency [300]), where the power of microprocessors doubles about every 18 months, and a similar high increase in cost-efficiency for other auxiliary technological parameters such as storage capacity and communications bandwidth [48].

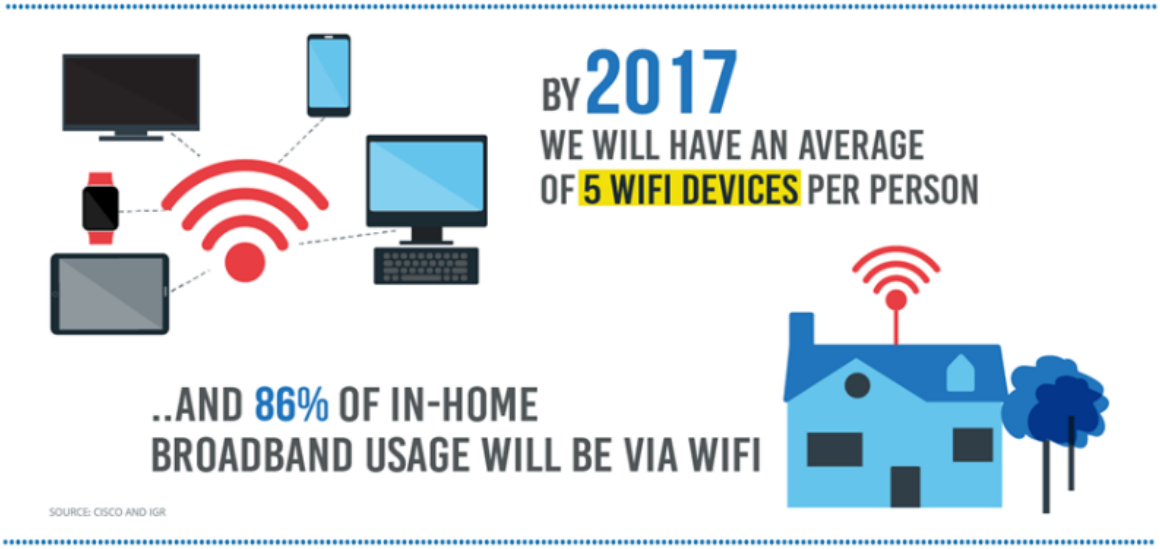

Figure 2.1: A shift in people-computing power ratio.

Initially, computers were very expensive [91], large in terms of physical dimensions [322] and most importantly difficult to understand and use. Each computer was a rare and precious resource. A single computer would typically be used by many individuals who were exclusively responsible for its operation (e.g., scientists, accountants, etc.) [66]. In the next evolutionary step during the 80s [263], the introduction of the PC changed the ratio to one user per computer, eliminating the need for turn-taking. As industry further progressed and costs dropped, every user gained access to at least one computer, whereas the type of computational resources that we have at our disposal today is dramatically more varied than a few decades ago. Today, access to multiple computers does not necessarily just mean owning both a PC and a laptop (Figure 2.1).

That rapid evolution of technology, along with the miniaturization of microprocessors and the fact that computing power could be embedded in familiar objects, gave birth to the field of AmI [82]. On top of such continuous technological advancement, the expectations of the users have been growing due to their experiences with computers, and people's fear of using them has decreased, transforming computing from a corporate tool available to a minority of users into the most-used tool on a daily basis [82]. Concomitantly with this difference in the way society perceives technology, millions of innovative applications and services [1, 2, 23] continuously emerge on almost a daily basis [78]. Microsystems, nanotechnology, electronic labels, wireless commu-

nications, web services, cloud computing are just examples which confirm that the technological basis for a new technological world has been created, where everyday objects and physical environments are in some respects "smart" and with which we can communicate under certain circumstances [48].

Currently, AmI lacks a well-established and universally accepted formal definition. In [82] it is defined as an emerging discipline that brings intelligence to our everyday environments and makes them sensitive to people. In [45] AmI is described as the incorporation of machine intelligence into people's everyday lives and existing environments that holds great potential for engendering drastic social transformations. The authors of [308] state that AmI is a new world where computing devices are spread everywhere (ubiquity), allowing the human being to interact in physical world environments in an intelligent and unobtrusive way. On the other hand, [317] adopts the definition given by the European Union [108] which identifies AmI as a paradigm that equips environments with advanced technology and makes them ergonomic in the sense that those living spaces will be capable of aiding us with daily chores and professional duties. Additionally, [329] highlights that the essence of AmI is the creation of environments saturated with computing and communication yet gracefully integrated with human users. Finally, [83] defines an AmI environment as one that is able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment.

Despite obvious differences, these definitions are built around similar concepts. First and foremost, an AmI environment is build upon advances in sensors and sensor networks, actuators, pervasive computing, and artificial intelligence [82], and requires the technological layer to be distributed [317], embedded and hidden in the surroundings [48], so as to create a pervasive environment [393]. Secondly, that environment should be sensitive, adaptive and responsive to the presence of people and objects [48], augment the activities through smart non-explicit assistance and preserve security, privacy and trustworthiness. Moreover, it should utilize contextual information when needed to behave intelligently [7], by customizing its requirements or forecasting behaviors [308]. Finally, an AmI environment should transparently interact with the user either passively by observing and trying to interpret their actions and intentions, or actively by learning their preferences and adapting its behavior accordingly to improve the quality of life [317].

AmI is described a multidisciplinary field where a wide range of scientific and technological

areas and human-directed sciences converge [45] towards a common vision of the future with enormous opportunities. Because these contributing fields have experienced tremendous growth in the last few years, AmI research has strengthened and expanded [82]. As AmI research is maturing, the resulting technologies promise to revolutionize daily human life by making people's surroundings flexible, adaptive and naturally controllable, resulting in enhanced efficiency, increased productivity and overall greater well-being [7].

2.1.2 AmI vs. UbiComp

In the literature the terms Ambient Intelligence, Ubiquitous and Pervasive Computing in many cases are mistakenly considered as synonyms. As [329] specifically states, the terms Ubiquitous Computing (UbiComp) and Pervasive Computing can be used interchangeably since their meaning is almost the same. In general, the term 'ubiquitous' means omnipresent: appearing or existing everywhere. It is a concept in Computer Science wherein computing can occur using any device and system, in any location and co-location, and in any design format, enabling the human user to interact with such diverse forms of computers, as laptops, smart cards and devices, tablets, and terminals in everyday objects [45]; it is a way to describe computers that "fit the human environment instead of forcing humans to enter their" [406]. As [308] states, Ubiquitous Computing is best considered as the underlying framework (e.g., the embedded systems, networks and displays, etc.) which is invisible and omnipresent, allowing to 'plug-and-play' devices and tools, whereas Pervasive Computing is related to all the physical artifacts that people use in their daily lives (e.g., mobile phone, hand-held computer or smart jacket, etc.).

AmI, on the other hand, as the Information and Communication Technologies Advisory Group (ISTAG) claims [109], emerged in parallel with UbiComp but is different from it, in that it is concerned more with the use of the technology than with basic technology. What characterizes this difference particularly is the focus (users in their environment versus next-generation computing technology) and the orientation (user-pull versus technology push) of technology. At the core of the AmI vision three technologies co-exist, namely Ubiquitous Computing, Ubiquitous Communication, and Intelligent User Interfaces, whereas AmI can be described as the merger of two important visions, namely 'Ubiquitous Computing' and 'Social User Interfaces' [45]. Moreover,

as [82] reports, AmI draws from advances in artificial intelligence, to deliver systems that are even more sensitive, responsive, adaptive, and ubiquitous, thus constructing "a digital environment that proactively, but sensibly, supports people in their daily lives".

2.1.3 Typical Components of an AmI Environment

AmI constitutes a new paradigm for future electronics which consider the presence of persons, provide personalized interaction and adaptation with respect to the user preferences, yet operate autonomously [7]. Nevertheless, to accomplish the AmI vision, a interplay between hardware and software components (Figure 2.2) is mandatory. In general, four main layers compile the infrastructure of an AmI Environment [82], namely: (i) Sensing, (ii) Networking, (iii) Perception and Reasoning, and (iv) Acting, which are combined in a bottom-up approach [63] that extends the predominant Sense-Plan-Act [61] robot control methodology. Specifically, the external layers sense and act upon the environment, whereas the middle layers constitute the processing core that decides what actions should be taken in order for the environment to exhibit the desired behavior based on the current context of use.

Sensing

Ambient Intelligence is designed for real-world, physical environments, thus the use of sensors is vital. Without physical components that allow an intelligent agent to sense and act upon the environment, AmI has no no practical use [82]. Different types of sensors can be employed in an AmI environment to acquire context. In general, the term 'sensor' is used to refer to tangible sensor hardware devices. However, sensors are often referred to any data source that provides relevant context; therefore, they can be divided into three categories [295]: Physical, Virtual and Logical.

Physical sensors. They constitute the most commonly used type of sensors, which generate data by themselves and enable AmI to retrieve low-level context. Such sensors are typically quite small, therefore they can be easily integrated into almost any AmI environment using a wired or wireless connection, and can monitor and collect various physical data (e.g., Table 2.1). As an alternative to stationary sensors, persons and items can wear a sensor (i.e., smart watch, pendant,

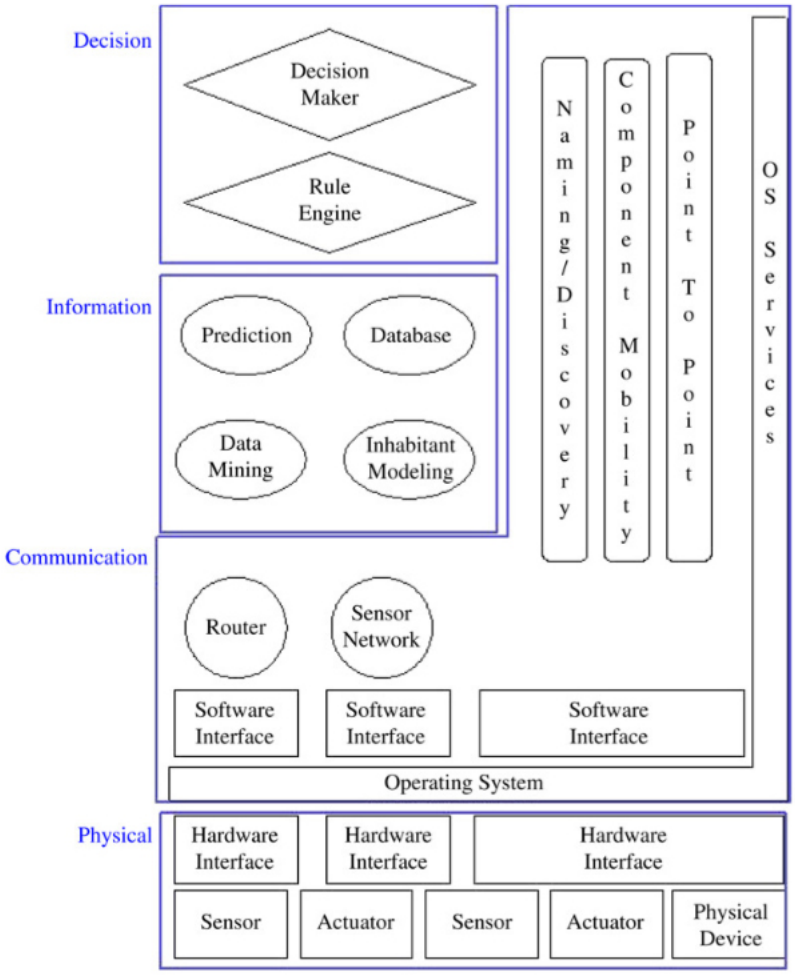

Figure 2.2: The components of an AmI environment as reported in [83].

ring, Bluetooth Low-Energy (BLE) beacons [375]) that track environmental properties on the move. The data provided by physical sensors are less meaningful, trivial, and vulnerable to small changes with respect to other sensors' categories.; hence, AmI needs to understand the physical world using imperfect, conflicting and imprecise data.

Virtual sensors. These sensors do not necessarily generate sensor data by themselves. Virtual sensors retrieve data from many sources and publish it as sensor data (e.g., calendar details, contact number directory, status of ambient lights, etc.). These sensors do not have a physical presence. They commonly use web services technology to send and receive data.

Table 2.1: Common types of physical sensors.

Properties	**Measurand**	**Common Uses**
Physical	Pressure, temperature, humidity, flow, luminance	Security, location, tracking, health safety, energy efficiency
Motion	Position, velocity, angular velocity, acceleration	Security, locator, tracking, falls detection
Contact	Strain, force, torque, slip, vibration	Floors, bed, sofas, scales
Presence	Tactile/contact, proximity, distance/range, motion	Security, locator, tracking, falls detection
Biochemical	Biochemical agents	Security and health, monitoring, pool maintenance, sprinkler efficiency
Physiological	Heart rate, EDA	Secutiry and health
Identification	Personal features, RFID or personal	Used to identify people and objects

Logical sensors. They are software-based components that combine physical and virtual sensors in order to produce more meaningful information; such process is called sensor fusion and generally improves detection efficiency [7]. Illustrative examples of logical sensors (also called software sensors) are: a fall detection sensor that fuses accelerometer data with body posture information from camera sub-system for confirmation, and a weather station that uses thousands of physical sensors to collect weather information and combines them with information from virtual sensors such as maps, calendars, and historic data in order to expose a single weather forecasting service. Input sensors of different kinds can capture and propagate to logical sensors both implicit and explicit interaction, in order to empower sophisticated AI algorithms to use that contextual information and to deduce who is the user that expressed a command and then detect the intention of that particular user, thus enhancing natural interaction.

Sensing-as-a-service: Due to the popularity of cloud computing [287], consuming resources as a service [415] such as Platform-as-a-Service (PaaS), Infrastructure-as-a- Service (IaaS), Software-as-a-Service (SaaS), has become mainstream. The Everything-as-a-service [34] model is highly ef-

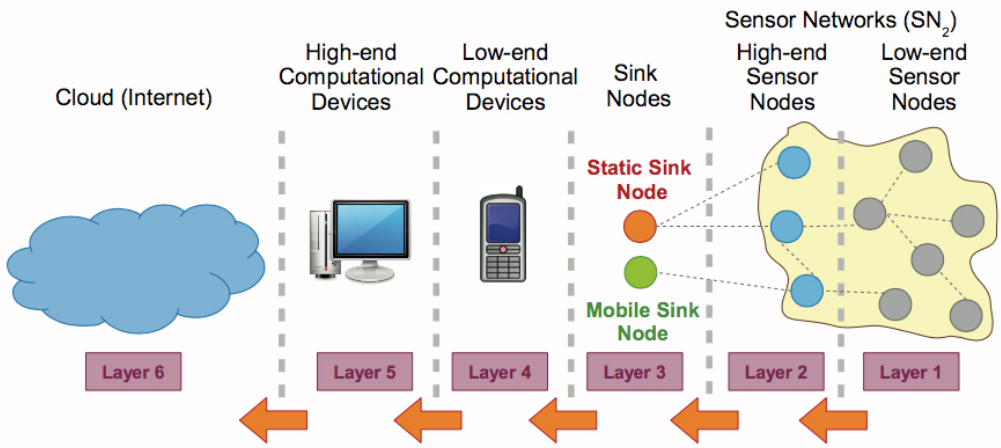

Figure 2.3: Layered structure of a sensor network as illustrated in [295].

ficient, scalable, and easy to use. AmI demands significant amounts of infrastructure (Figure 2.3) to make its vision a reality, therefore an everything-as-a-service model can be used as the mold to implement tailored sensing-as-a-service [296] models.

Communicating

An AmI environment is anticipated to include a rich decentralized and heterogeneous infrastructure of sensors that will observe and actuators that will affect the physical surroundings, provided by multitude of devices (as seen in Figure 2.4). All of these artifacts are intelligent but they only work alone and locally, as despite their networking capabilities, they do not inherently 'network' to extend their functionality [362]. Like distributed computing, they share their capabilities and raw measurements [325] using many alternative protocols [144] such: RFID, Zigbee (ZigBee), Bluetooth, Z-Wave (Z-Wave), LoRaWAN (LoRaWAN), WPAN (WPAN), WSN (WSN), WiFi (Wifi), WiMax (WiMax), 4G, 5G, etc. To ensure seamless access to remote information resources and communication with fault tolerance, high availability, and security, like distributed, mobile and pervasive computing, AmI requires a middleware "shell" to interface between the networking kernel and the applications running on pervasive devices.

A middleware architecture aims at providing complete transparency of the underlying technology and their surrounding environment. Generally, it consists mostly of firmware and software bundles executing in either client-server or peer-to-peer mode [325] and offers: (i) identity-related

Services that bring real world objects to the virtual world, (ii) information Aggregation Services that collect and summarize raw sensory measurements that need to further be processed [15], and (iii) on-demand action triggering across remote end-points. Therefore, the communication layer empowers the provision of context-aware, situation-aware and pervasive computing that delivers services and information to people relevant to what they want, where and when they want them, in the way that they prefer to consume them.

Figure 2.4: Communication layer is the cornerstone of the IoT reference architecture [151].

Context-awareness and Reasoning

In the AmI vision, collection and distribution of sensor data constitutes just the tip of the iceberg. An AmI environment is expected to not just observe, but try to interpret the situation and actively learn [317] in order to support people in their daily-used environments, as [295] states by quoting

Dey's definition of the term context-awareness: "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task". In addition, understanding context makes it easier to perform Machine-to-Machine (M2M) communication. Therefore it is clear that Intelligence, in the form of Artificial Intelligence, is a prerequisite towards Ambient Intelligence [308].

The Oxford dictionary defines Intelligence as "the ability to acquire and apply knowledge and skills". Specifically, in the context of AmI, intelligence can be divided into two discrete categories: Context-awareness and Reasoning. Context-awareness, or Perception, is an intrinsic characteristic of intelligent environments and refers to raw data collection and their transformation into high-level information (knowledge). Implementing perception introduces significant complexity: location monitoring, uncertainty modeling, real-time information processing, and merging data from multiple and possibly disagreeing sensors. The information that defines context awareness must be accurate; otherwise, it can confuse or intrude on the user experience [325]. Reasoning couples sensing with acting and provides links between intelligent algorithms and the real world where they operate in terms of intelligent interaction and communication. In order to make such algorithms responsive, adaptive, and beneficial to users, a number of different reasoning tasks must take place. These include user modeling, activity prediction and recognition, decision making, and spatial-temporal reasoning [82].

Context is about evolving, structured and shared information spaces that are designed for a particular purpose, to amplify human activities with new services that can adapt to the circumstances [85]. Context can be further partitioned into primary and secondary [295]. Primary context includes any information retrieved without using existing knowledge and without performing any kind of sensor data fusion operations (e.g., Global Positioning System (GPS) sensor readings as location information). Secondary context refers to any information that can be computed using such primary context, by using sensor data fusion operations or data retrieval operations such as web service calls (e.g., identify the distance between two sensors by applying sensor data fusion operations on two raw GPS sensor values).

Context services form a fabric structured into multiple levels of abstraction, as illustrated in Figure 2.5, that aim to provide a structured unified view of the world. The sensing layer generates numeric observables. To determine meaning from numeric observables, the system must perform

transformations. The perception layer is independent of the sensing technology and provides symbolic observables at the appropriate level of abstraction. The situation and context identification layer identifies the current situation and context from observables, and detects conditions for moving between situations and contexts, and finally, the exploitation layer acts as an adapter between the application and the infrastructure.

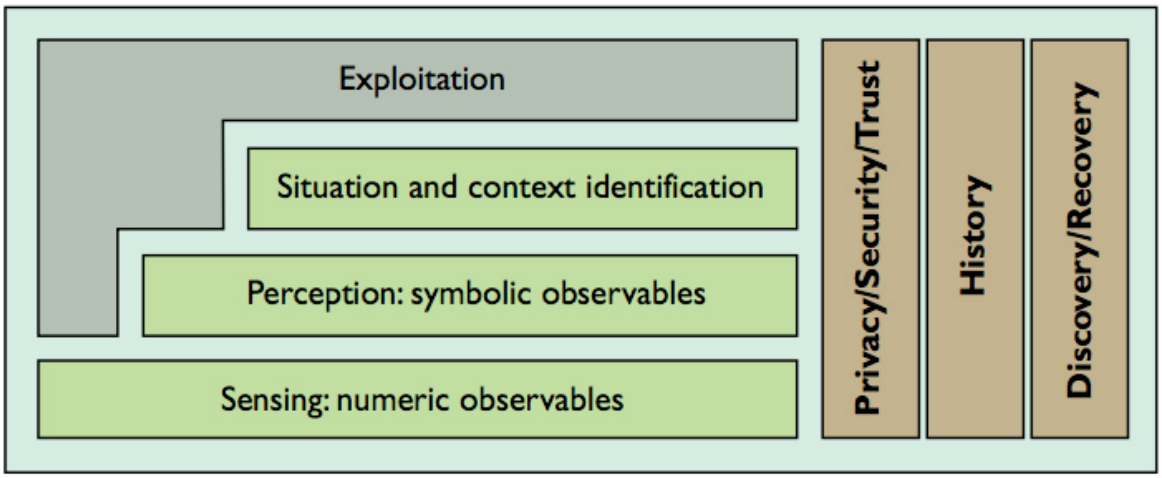

Figure 2.5: Levels of abstraction for a general-purpose infrastructure for context-aware computing [85].

In [308] it is highlighted that the Encyclopedia of Artificial Intelligence (AI) refers to the concept of Ambient Intelligence and notices that, in the past, the AI community centred the attention on the hardware (40's and 50's), on the computer (60's), on the network (70's and 80's) and on the Web (90's till now). Despite the fact that AI has successfully applied well-established techniques (e.g., Planning, Rule-based reasoning, etc.) towards the realization of automated, yet isolated, AmI applications, still work is needed to fully automate the entire environment [83]. Reasoning in AmI refers to the various decision making mechanism incorporated in the environment in order to make it exhibit different outputs according to the identified situation. The domain of AI already contributes various technologies and approaches that assist Context-awareness and Reasoning [308], which are outlined in Table 2.2.

In AmI environments, context-awareness is no longer confined to a desktop, web, or mobile applications. Given that it constitutes the heart and brain of the overall environment, it is essen-

tially distributed in the surroundings in the form of a service, available to be consumed by every interested party, realising the Context-as-a-Service (CXaaS) concept [178], which distributes it via: (i) a query interface and (ii) Publish-Subscribe protocol. The former empowers agents to acquire contextual information on-demand, and (ii) the latter provides greater scalability and a dynamic topology, as senders of contextual information (i.e., publishers), do not program the messages to be sent directly to specific receivers (i.e., subscribers) but instead broadcast them to the environment without knowing which subscribers will handle them.

Finally, in AmI, context is a highly dynamic and diverse collection of information that drives the overall intelligent behavior especially regarding user interaction. With respect to that, [295] identifies context-awareness as:

Personalization, that allows the users to set their preferences, likes, and expectations to the system manually. For example, users may set the preferred temperature in a smart home environment where the heating system of the home can maintain the specified temperature across all rooms.

Passive, where the system constantly monitors the environment and offers the appropriate options to the users so they can take actions. For example, when a user enters a super market, the mobile phone alerts the user with a list of discounted products to be considered.

Active, when the system continuously and autonomously monitors the situation and acts autonomously. For example, if the smoke detectors and temperature sensors detect a fire in a room in a smart home environment, the system will automatically notify the fire brigade as well as the owner of the house via appropriate methods such as phone calls.

Acting and Interacting

The vision of AmI dictates that ultimately the environment should be sensitive, adaptive, and responsive to human needs so as to assist its users in their daily tasks and make their lives simpler. Far more than mobile computing, AmI fundamentally changes the nature of computing, allowing most objects people encounter in daily life to be “aware” of their interaction in both the physical and virtual world [325]. Therefore, an important aspect of AmI has to do with seamless and natural cooperation between the human and the environment towards a common objective, a concept known as Human-Computer Confluence [119]. For confluence to be achieved, the environment

Table 2.2: AI Technologies contributing to Context-awareness and Reasoning.

Domains	**Technologies**
Knowledge representation	Ontologies, Semantic web, Information Retrieval.
Machine learning	Neural networks for classification and other techniques (inductive learning, case-based reasoning, decision trees, Behavior trees).
Computational Intelligence	Pattern recognition and Optimization-oriented methods like: Neural Networks, hidden Markov models, Genetic Algorithms, Ant Colonies, Particle Swarm Intelligence, Taboo Search, Simulated Annealing, Fuzzy Logic.
Planning	Plans can be established before the plan execution (off-line) or during the execution (on-line). Deliberative, Reactive or Hybrid.
Incompleteness and Uncertainty	Bayesian, Fuzzy Logic, Dempster-Shafer Theory.
Speech and NLP	Spellcheck, Part of Speech Tagging, Sentiment Analysis, Synonyms, Entity Recognition.
Computer vision	Gesture Detection, Human Tracking, Fall Detection.
Robotics	Autonomous stationary or mobile robotic agents.
Multi-agent systems	AI community started with a new area, DAI, combining AI with Distributed Computing.

should appropriately react to any kind of user-driven interaction or proactively use its intelligence to infer situations and user needs from the recorded activities, much as a butler observes activities unfold with the expectation of helping when (and only if) needed [82]. This is the idea of an "Intelligent Social User Interface" [379]. To that end, Human Computer Interaction (HCI) [104] in such environment gets reformed by replacing traditional explicit interaction technologies that are difficult or unnatural for residents, with more human-life communication capabilities and implicit actions [95 cook2009]. Various mature and well-established technologies, including motion tracking, gesture recognition [96], facial expression recognition [97] and emotion recognition [98], speech processing [99], and even whistle processing [100] facilitate natural interactions with AmI environments. Additionally, diverse interface mechanisms are combined to form multi-modal [12, 14, 67]

and distributed interfaces [317], without the need of devices foreign to the environment or unfamiliar to the target user group (e.g., elderly) [83]. Finally, advanced HCI concepts such as Intelligent User Interface (IUI) [246, 359] and User Interface Adaptation [331, 350, 367] not only enable users to indicate their preferences and needs, but also prevent AmI technologies from becoming "ubiquitous clutter" [338].

2.1.4 Relation with other Emerging Technologies and Domains

In addition to the aspects of context-aware computing, disappearing computers, pervasive and ubiquitous computing, and AI, AmI incorporates technologies from other relevant emerging domains such as: Cloud and Fog Computing, Internet of Things (IoT) and Cyber-Physical System (CPS). Their contributions empower AmI to reach its potentials, including the realization of scenarios from science fiction movies, like: doors opening when someone with the appropriate access permissions approaches or computers able to identify the interlocutor without their name being explicitly mentioned.

Cloud Computing

NIST formally defines **Cloud Computing** [250] as: *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models*" (depicted in Table 2.3) , which can be simply described as a computing environment where computing needs by one party can be outsourced to another [180].

The main goal of Cloud computing is to maximize the utilization of distributed resources, achieve higher throughput and solve large scale computational problems. With respect to AmI, its main advantages that constitute the decisive features for its application are: (i) on the one hand the ability to access data, applications and services over the internet regardless of the user's device or localization, and (ii) on the other hand the fact that performance can be monitored and scaled according to the actual computational needs (i.e., elasticity [164]). In addition to its

intrinsic architecture-oriented benefits, Cloud computing also significantly contributes from an engineering perspective by introducing key tools and practices that address various aspects of an AmI system such as: automated deployment tools [86], programming Software Development Kits (SDKs) that facilitate distributed and asynchronous interoperability, as well as relevant service management practices that simplify maintenance, reduce operation costs and ensure uninterrupted service.

Table 2.3: The NIST Cloud Computing Model

Essential Characteristics	On-demand service Broad network access Resource pooling Rapid elasticity Measured service (control and optimize)
Service Models	SaaS PaaS IaaS
Deployment Models	Public Community Private Hybrid

Internet of Things

Internet of Things (IoT) is the concept of many objects, smart devices, machines, consumers, patients and services being increasingly able to be connected to solve problems in new and more effective ways. The vision behind the concept is that increased connectivity will facilitate automation, visibility and access to services, which subsequently will enable companies and governmental organizations to tailor products and services (Table 2.6) to individual needs and ensure they are delivered accurately and effectively [151]. Smart objects along with their functionality constitute domain specific applications (vertical markets) targeting a wide spectrum of spaces, such as consumer/domestic, commercial, industrial, agricultural, medical, transportation and so on, while ubiquitous computing and analytical services form application domain independent services (horizontal markets) [15].

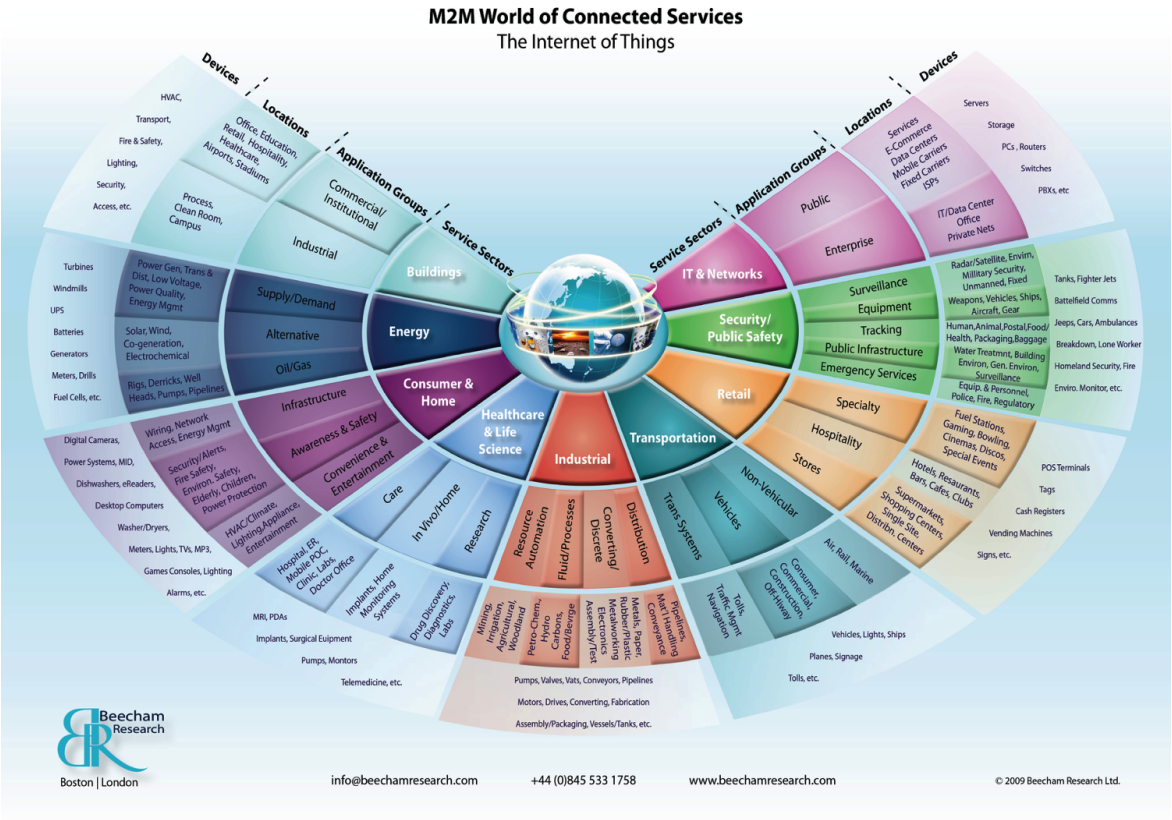

Figure 2.6: IoT and its Application Domains.

An important remark about IoT is that it is a vision, not a technology, which rapidly becomes a reality, driven by the growing number of physical objects connected to the internet, which can see, hear, think and perform functions by having them “talk” to each-other, to share information and to coordinate decisions [15]. This is based on the convergence of:

- everyday use of digital connectivity by industry and consumers, leading to a natural expectation that all things will be “connected”
- the upwards trend in capability and the downwards trend in cost of microcontroller and communications technologies
- the proliferation of cloud-based data gathering, processing and dissemination platforms.

The integration of multiple technologies in IoT applications requires a high-level of interop-

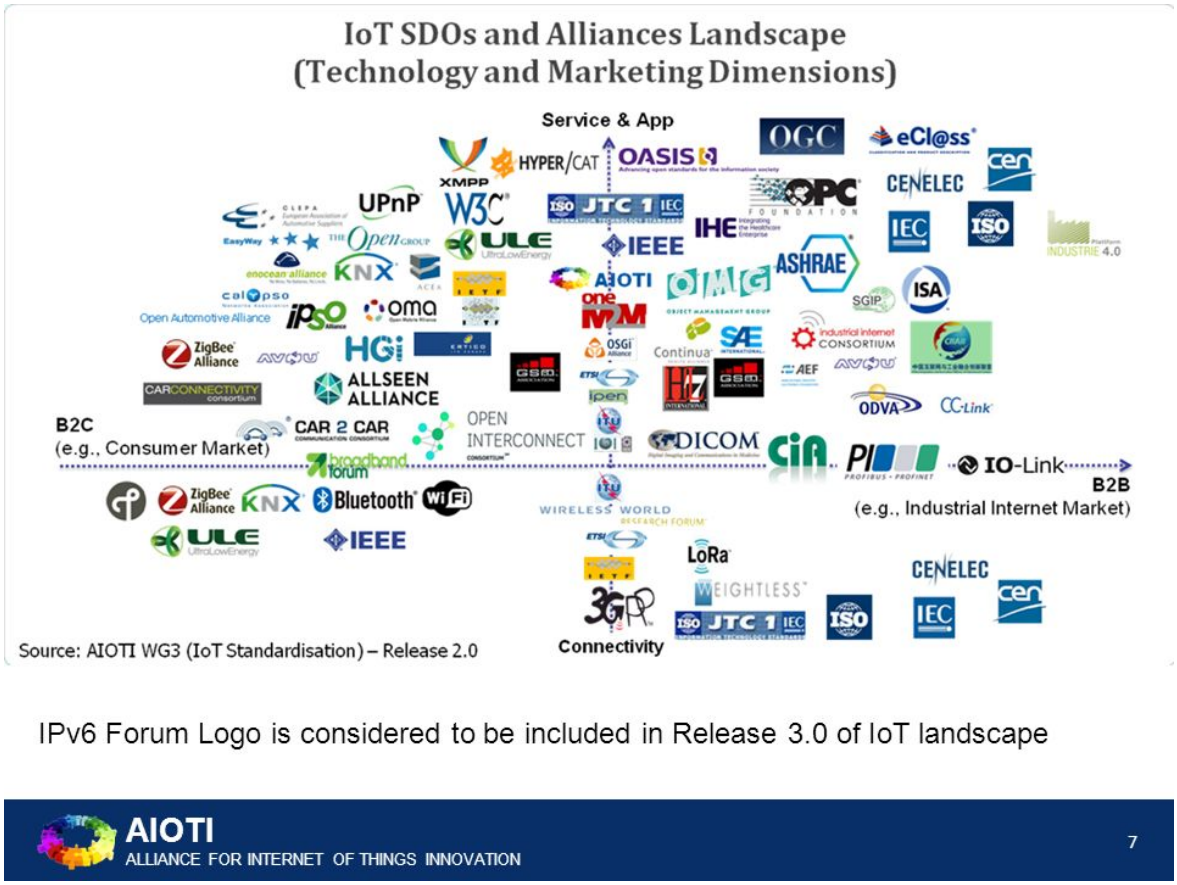

Figure 2.7: Members of the AIOTI.

erability between the various components. This is a critical factor to enable IoT applications to develop and to proliferate, and arises from the fact that IoT is a concept rather than a specific technology and different organizations tend to define it from their perspective. However, worldwide initiatives aim to eliminate those stumbling blocks via standardization (Figure 2.7).

Cyber-Physical Systems

Computing and communication capabilities are being embedded in all types of objects and structures in the physical environment. Applications with enormous societal impact and economic benefit will be created by harnessing these capabilities across both space and time [307]. Such systems that bridge the cyber-world of computing and communications with the physical world are referred to as **Cyber-Physical Systems (CPSs)**. A CPS is a smart system that includes engineered

interacting networks of physical and computational components [150] that can interact with humans through many new modalities [28]. The operations of a CPS are monitored, coordinated, controlled and integrated by a computing and communication core.

This intimate coupling between the cyber and physical will be manifested from the nano-world to large-scale wide-area systems of systems [307], as depicted in Figures 2.8 and 2.9, whereas the ability to interact is a key enabler for future technology developments [28]. To that end, embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa [216]. In addition to CPS, there are many terms (e.g., Industrial Internet, IoT, M2M, Smart Cities) that describe similar or related systems and concepts, while their significant overlap permits the usage of the terms CPS and IoT interchangeably [150].

Applications of CPS are claimed in [216] to have the potential to dwarf the 20^{th} century IT revolution, by providing new means to improve quality of life and enabling technological advances in critical areas [150], such as those listed in the following Table 2.4:

Table 2.4: Indicative applications domains for a CPS

Domestic Environments	Energy conservation, Environmental control, Smart net-zero energy structures [337]
Health	Personalized health care, Medical devices and systems (e.g., prostheses that allow brain signals to control physical objects [28]), Assisted living
Transportation	Avionics, Traffic control, Advanced autonomous automotive systems, Intelligent highways, Aerospace systems
Automation	Process control, Instrumentation, Critical infrastructure control (e.g., electric power, water resources, communications), Smart energy systems
Industry	Smart manufacturing (Industry 4.0), Distributed robotics (telepresence, telemedicine)
Security	Safety, Defense systems, Homeland security, Emergency response

Via a ubiquitous infrastructure consisting of a variety of global and localized networks, users, sensors, devices, systems and applications may seamlessly interact with each other and even the

physical world in unprecedented ways [337]. Moreover, in such an environment, CPS can be further developed using Big Data and leveraging the interconnectivity of machines to reach the goal of intelligent, resilient and self-adaptable machines. [217]. Nevertheless, for a CPS to be successful, it should not only interact with the physical world, but operate dependably, safely, securely, and efficiently and in real-time [307].

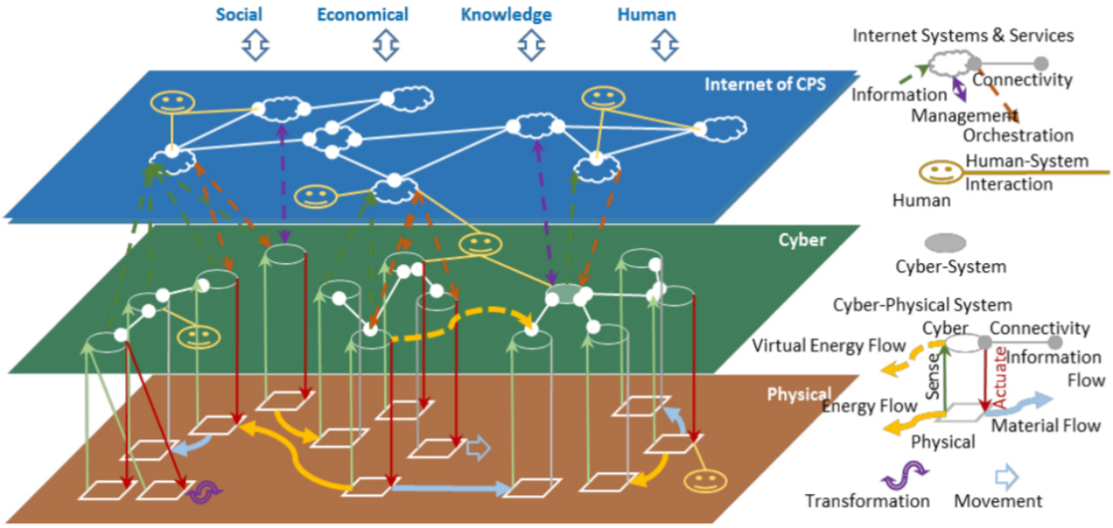

Figure 2.8: A CPS View: Systems of Systems.

Fog Computing

The integration of IoT with Cloud computing, termed as Cloud of Things (CoT), can help achieve the goals envisioned by the Internet of Everything (IoE) [257] and Future Internet [407]. However, due to the surge demand on ubiquitous high-quality mobile services for every object that will be online and the consequent explosive growth of network traffic [231], the centralized IoT-Cloud computing integration is not straight-forward [8], and does not scale to accommodate the advanced requirements of such environments [92]. Therefore, despite its broad utilization, some applications and services still cannot benefit from that popular computing paradigm.

The main problem resides to the inherent necessity of cloud-based solutions to send data over the network and pay the imperative round-trip cost. Even-though data centers of clouds are located near the core network, many applications and services will suffer unacceptable latency (e.g.,

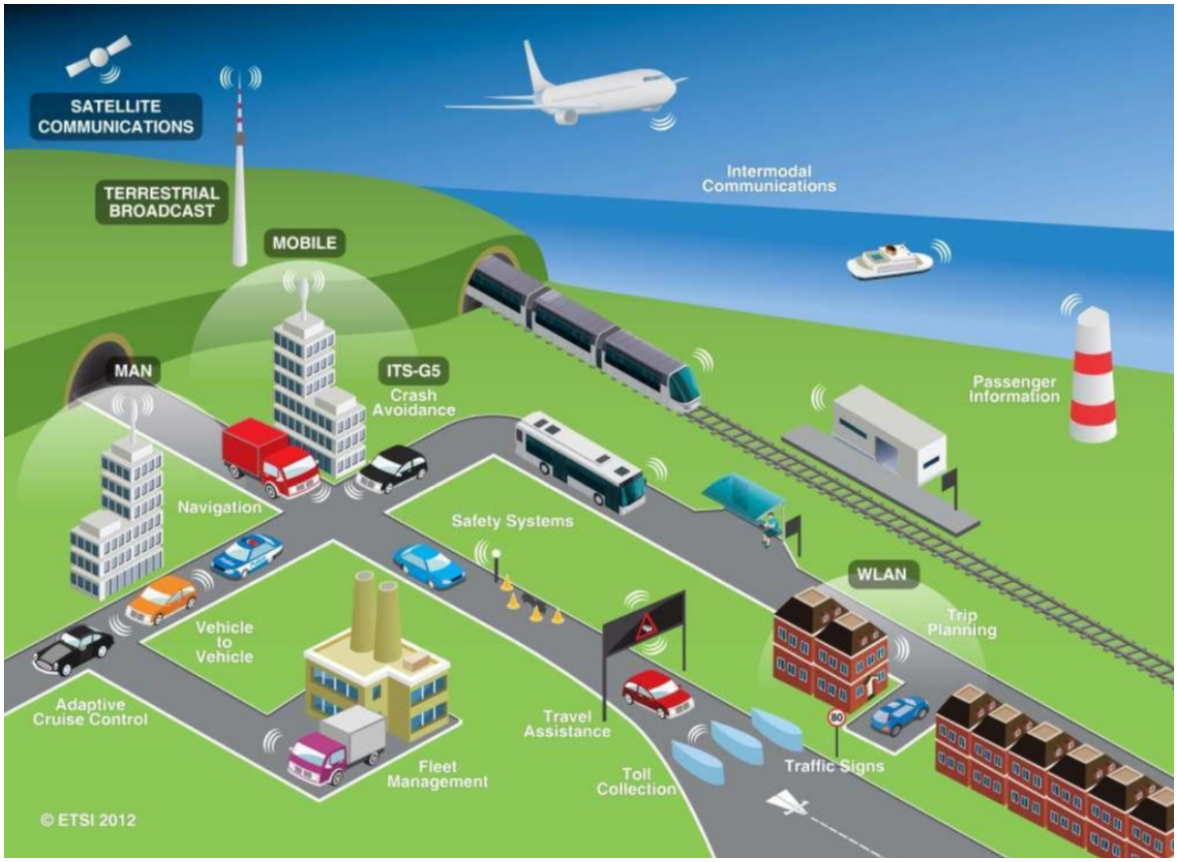

Figure 2.9: Vision of a Transportation CPS as seen in [150].

Health monitoring and emergency response services [93], Real-time gaming, Augmented Reality, Real-time streaming), when data are transmitted from/to end devices to/from the cloud data center through multiple gateways [404]. Besides this, there are also problems unsolved in IoT applications that usually require mobility support, geo-distribution and location-awareness [404, 405]. Because unnecessary communication not only burdens the core network, but also the data center in the cloud, any data should be preprocessed and trimmed before sending to the cloud [8]. Consequently, Fog computing has emerged as a potential solution to the above problem [353].

Fog Computing was first introduced by CISCO [77] and is defined as a distributed computing paradigm for managing a highly distributed and virtualized environment of elastic [405] resources and location-based services, that fundamentally extends the services provided by the cloud to the edge of the network [93]. The main idea of Fog computing is to bring networking resources near the underlying networks [8] by placing light-weight cloud-like facility at the proximity of mobile

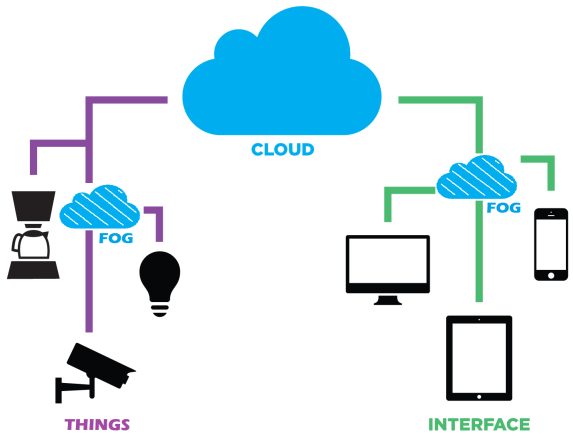

Figure 2.10: Fog Computing extends the cloud to the edge of the network.

users so as to serve mobile users with a direct short-fat connection as compared to the long-thin mobile cloud connection [231], which enables real time applications [353] and acts as a bridge between smart devices and large-scale cloud computing and storage services [15]. Fog Computing supports mobility, computing resources, communication protocols, interface heterogeneity, cloud integration, and distributed data analytics services between sensors and cloud data centers, thus decreasing the latency and network congestion and enabling a new breed of applications and services [53]. In essence, Fog Computing enables to deal with IoT data locally by utilizing clients or edge devices near users to carry out a substantial amount of storage, communication, control, configuration, and management.

The Fog Computing approach benefits from edge devices' close proximity to sensors, while leveraging the on-demand scalability of cloud resources [93]. It is particularly suitable for IoT tasks and queries, as with the increasing number of smart devices, most of the requests pertain to the surroundings of the device, thus: minimizing latency [53, 93, 231, 354, 404], reducing network traffic between cloud and mobile users [93, 231], reducing bandwidth cost due to connections' locality [231], increasing data rate [231], providing customized and location-aware services [53, 75, 231, 354], improving QoS and overall performance for streaming and real time applications [15, 354], improving efficiency using pools of local resources [75, 404], promoting scalability [53, 93, 404], wide-spread geographical distribution and mobility [53], enabling agility to empoever rapid innovation and affordable scaling [75].

The above characteristics make the Fog the appropriate platform for a number of critical IoT services and applications, such as: Connected vehicles, Smart Grid, Smart Cities, Wireless Sensors and Actuators Networks (WSANs) [53], roads, highways and tracks [8], medical centers, farms [93], Real-time Mobile Big data analytics [354,405], advertising [354] and entertainmen (including Real-time AR applications [405]. As a closing remark, it should be emphasized that Cloud and Fog computing are not a binary choice. On the contrary, they form a mutually beneficial inter-dependent continuum, beneficial as certain functions are naturally more advantageous to carry out in Fog while others in Cloud; inter-dependent because coordination among devices in a Fog may rely on the Cloud, and finally a continuum as the wearable devices, and mobile phones may be viewed as the Cloud [75].

2.1.5 Economical, Societal and Ethical implications of AmI

Ambient Intelligence (AmI) is still considered as an emerging technological domain (Figure 2.11) with huge economical potentials. In 2013 [152] predicted that the next revolution will be the interconnection between objects to create a smart environment. They reported that only in 2011, the number of interconnected devices on the planet (estimated close to billion) overtook the actual number of people and that number was expected to reach 24 billion devices by 2020, which according to the Groupe Speciale Mobile Association (GSMA), amounted to $1.3 trillion revenue opportunities for mobile network operators alone, spanning vertical segments such as health, automotive, utilities and consumer electronics. The same year (2013) Gartner Research predicted that endpoints of the IoT will grow at a 32% Compound Annual Growth Rate (CAGR) from 2013 through 2020, reaching an installed base of 21 billion units, with almost two-thirds of them being consumer applications. Spending on networked consumer and business endpoints will displace non-networked, growing at a 22% CAGR to $3 trillion. [256].

A couple of years later (2015) these numbers are growing faster than originally expected (as the relevant surveys confirm). McKinsey reported that, based on a bottom-up analysis for its applications, the IoT is estimated to have a total potential economic impact of $3.9 trillion to $11.1 trillion a year by 2025. At the top end, that level of value -including the consumer surplus- would be equivalent to about 11 percent of the world economy (exhibit) [237]. In 2016, IHS forecasted

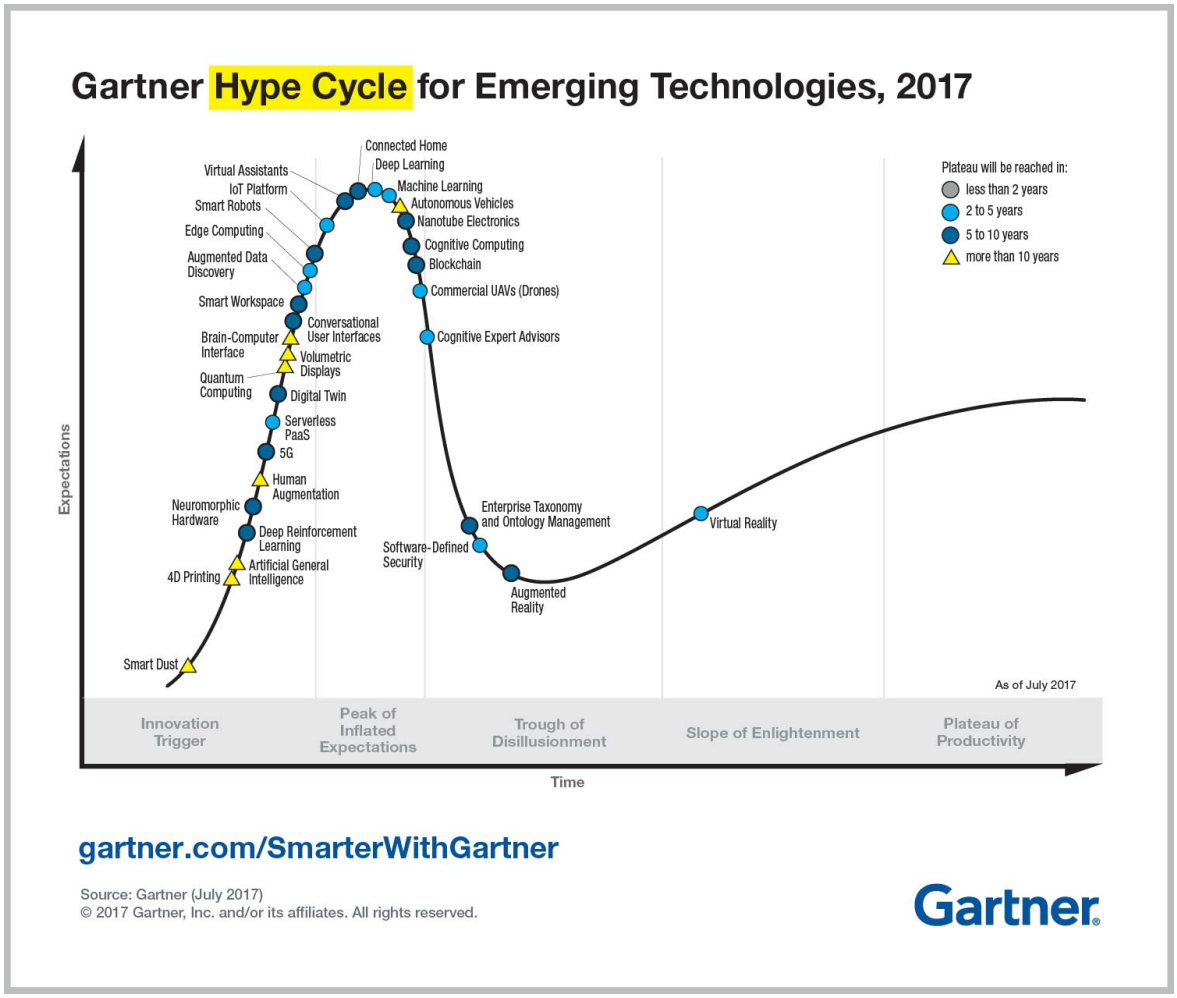

Figure 2.11: The 2017 Hype Cycle of Emerging Technologies [139].

the massive growth of the IoT market, which is expected to grow from an installed base of 15.4 billion devices in 2015 to 30.7 billion devices in 2020 and 75.4 billion in 2025 [232], as reported below in Figure 2.12. Finally, Forbes [79] in its yearly series of Internet of Things (IoT) and Industrial Internet of Things (IIoT) forecasts, reflected a growing focus on IoT by quoting Bain's prediction that by 2020 annual revenues could exceed $470 billion for the IoT vendors selling the hardware, software and comprehensive solutions, and General Electric's anticipation that the investment in the IIoT is expected to top $60 trillion during the next 15 years.

As [15] argues, all these statistics point to a potentially significant and fast-pace growth of the IoT and the related industries and services in the near future. This progression provides a

Figure 2.12: Forecast of the number of IoT-enabled devices installed worldwide [232].

unique opportunity for traditional equipment and appliance manufacturers to transform their products into “smart things”. Spreading the IoT and related services globally requires Internet Service Providers (ISPs) to provision their networks to provide QoS for a mix of M2M, Person-to-machine (P2M) and Peer-to-peer (P2P) traffic flows. These smart objects along with their supposed tasks constitute domain specific applications (vertical markets) while ubiquitous computing and analytical services form application domain independent services (horizontal markets). Nevertheless, there is an imperative need for better horizontal integration among such services to accomplish the vision of AmI and deliver new services in a large number of real-life scenarios [144]. Tables 2.5 and 2.6 aggregate and classify many application domains that are already or will be benefited by AmI services and resources, adapting the categories reported in [295].

Evidently, AmI has the potential to be interweaved in various aspects of people’s daily activities (including their professional, personal and social life), a fact that raises many concerns regardings its social and ethical implications. As throughout history, the development of regulatory, social, and ethical standards tends to lag considerably behind the rapid proliferation of pioneering technological inventions, therefore most of them have not been properly addressed yet. One of the most important issues that refer to personal privacy and social acceptance of the ambient solu-

tions [48].

In particular, privacy is a topic that has received much critique as AmI is foreseen to create an invisible and comprehensive surveillance network, covering an unprecedented share of our public and private life, and affect every facet of privacy; namely privacy: (i) as **empowerment** to control which information will be published and distributed, (ii) as a **utility** to protect every individual's right to personal life, (iii) as a **regulating agent** that balances on the powers of a decision-making elite by limiting information gathering, and (iv) as the **dignity** that maintains the equilibrium of information availability between people.. Moreover, it is often argued that continuous surveillance can potentially violate the four border crossings [242] that form the basis for perceived privacy violation: (i) **natural**, (ii) **social**, (iii) **spatial or temporal**, and (iv) **ephemeral**. Whether such crossings will ultimately occur, depends to a large extent on the type of searching capabilities that AmI systems might offer, including the power of searching and combining information bits.

With respect to social challenges, AmI has to address the issues that arise from the artificial introduction of a technologically induced dependence that cannot be easily escaped and leads to a number of fundamental social challenges, where basic attributes of the world subtly change: (i) AmI has to ensure **reliability** in terms of remaining manageable and controllable, retaining the ability to predict (and, to a certain extent, verify) that the system is behaving correctly and overcoming service interruptions and device failures; (ii) Given that the need for human intervention in complex and highly dynamic environments will be minimized, new concepts for **delegating control** are necessary, including the need for manual override when the environment does not perform as expected or its behavior is not the intended one; (iii) AmI systems should ensure **social compatibility** by adjusting their behavior to retain their operational transparency and ease-of use. Moreover, they should respect and promote knowledge sustainability and information inertia that permits humans to use acquired knowledge and prior experiences to cope with future situations and tasks, and since the AmI landscape targets the entire society, fairness and universal access should be guaranteed; (iv) Finally, the decisive factor that will determine whether the AmI vision will be successful or not, is its widespread public **acceptance** which rests on issues of an almost philosophical nature, such as the fundamental nature of smart objects or our changing relationship with our environment [392]..

Table 2.5: Application domains of AmI

Domain	Objectives	Cases
Domestic and Home	(i) Artifacts and items enriched with sensors to transparently gather information about their use and act independently without human intervention, (ii) Intelligent resources utilization, and (iii) Support for daily activities and routines	[7, 15, 82, 83, 134, 295, 308, 408]
AAL	(i) Promotion of independent living and socialization, (ii) Real-time health monitoring, (iii) Tele-assistance, (iv) Wearables, (v) At-home automated assistance for people with mental and physical challenges, (vi) Reduced cost of hospitalization, and (vii) Sharing of caregiving human resources	[7, 96, 192, 308, 317, 337]
Education	(i) Learning Content Personalization, (ii) Alternative pedagogies, (iii) Behavior student monitoring, (iv) Learning in the wild, and (v) Universally accessible ubiquitous digital libraries of knowledge	[11, 82, 83, 221, 336]
Workplace	(i) Ad-hoc formation of dynamic teams during meetings, (ii) Virtual organizations, (iii) Physically distributed and mobile workers, (iv) Desk sharing, (v) Human resources management, (vi) Office organization, and (vii) Decision making	[82, 295, 308, 357, 383]
Culture and Tourism	(i) High-degree of customization, (ii) Immersive tours, (iii) Post-visit reflections, (iv) Arts training, and (v) Interactive exhibits	[124, 149, 286, 308]
Sports	(i) Tele-assistance, (ii) Wearables for real-time monitoring of physiological parameters, (iii) Personalized training and coaching, and (iv) Real-time decision-making support (e.g., Hawk-Eye System)	[29, 140, 161, 308]
Cities	(i) Smart grid applications in production, transmission and distribution (ii) Large-scale 3D printing, (iii) Waste management, (iv) Traffic congestion management, (v) Smart parking, (vi) City energy consumption monitoring and control, (vii) Artistic re-enchantment of the urban landscape with informational artifacts, (viii) Efficient energy routing and distribution, (ix) Smart lighting, (x) Smart mobility, and (xi) Smart water management	[15, 87, 214, 225, 306, 339, 348, 352]
Transportation	(i) Autonomous Vehicles, (ii) Proactive driving assistance (incl. non-line-of-sight communication), (iii) Personal safety and asset security in public transport, (iv) Location-aware services in aviation, (v) Real-time monitoring of flight-related parameters, (vi) Aircraft design and manufacturing	[15, 52, 106, 138, 144, 225, 283, 308, 312, 382]
Environment	(i) Air quality monitoring, (ii) Water supply monitoring, (iii) Atmospheric or soil conditions monitoring (iv) Noise pollution (v) Fire prevention and early warning for possible fire disaster, (vi) Early of physical disasters (e.g., earthquakes, tsunami), and (vii) Net zero energy buildings	[90, 97, 306, 337, 339, 384, 394]

Table 2.6: Application domains of AmI

Category	Objectives	Cases
Security and emergencies	(i) Real-time and accurate situational awareness about emergency scenes, (ii) Accurate and real-time accountability of resources, (iii) Remote monitoring, tracing and tracking of response personnel, (iv) Efficient resource allocation and deliverery to the disaster scene, (v) Optimized coordination between the various participating response teams (e.g., fire brigades, police forces, ambulance services, local or national public sectors, and humanitarian aid organizations) (vi) Extensive real-time information on disaster development and the remaining resources of each participating organization, (vii) Fine-grain orchestration of mission critical processes (e.g., tower crane groups), (viii) Improved construction safety management efficiency, and (ix) Real-time detection, monitoring and early warning of safety risks	[82, 103, 107, 144, 225, 400, 413]
eHealth	(i) Reduced cost of healthcare and wellbeing services, (ii) Improved efficacy of hospital information systems, (iii) Remote provision of health care, (iv) Improved quality and efficiency of patient care and medical, nursing, administrative and management tasks and needs, and (v) Wearables	[7, 10, 15, 82, 83, 144, 295, 308, 315, 324]
Retail	(i) Immersive customer experiences that use AR to facilitate embodiment, (ii) Customer behavior analysis, (iii) Indoor human and object localization and tracking, (iv) Interactice digital signage, (v) Smart shopping windows, hangers, shelves and showcases (vi) Wearables for employees for efficient customer response, (vii) Enhanced product information including personalized content and users' product ratings, (viii) Personalized location-sensitive promotions and advertising, (ix) Conversational virtual agents, (x) Physical brochures augmentations, (xi) Virtual product catalogues, (xii) MAR to virtually integrated products in physical environments, (xiii) Enlarging limited space of physical stores using VR and AR , and (xiv) Real-time personalized billing systems	[16, 47, 48, 72, 73, 102, 165, 177, 303]
Logistics	(i) Farm-to-plate products tracking, (ii) Real-time monitoring of the move of physical objects from an origin to a destination across the entire supply chain including manufacturing, shipping, distribution, (iii) Fleet tracking and performance monitoring, and (iv) Optimized shipping and routing	[48, 90, 144, 295]
Industrial control	(i) Connecting factories to the smart grid, (ii) Sharing the production facility as a service, (iii) Allowing more agility and flexibility within the production systems, (iv) Improved safety in hazardous occupations (e.g., underground mining), and (v) Monitor vital signs of industrial motors (e.g., temperature, vibrations), (vi) Enhancement of Lean Manufacturing using technology	[7, 15, 59, 90, 175, 295, 339]
Agriculture	(i) Food chain monitoring, (ii) Minimized ecological footprint, (iii) Reduced cost due to resources optimization, (iv) Increase food security, and (v) Precision Agriculture	[51, 248, 394, 410]
Animal farming	(i) Breeding monitoring, (ii) Animal identification, (iii) Real-time detection and movement tracking of animals, (iv) Early detection of outbreaks of contagious diseases, and (v) Health monitoring, (vi) Certification of origin (e.g., region, country), and (vii) Direct marketing	[295, 361, 388]

2.1.6 Challenges and Open issues

As with any emerging technology that strives for its establishment, many challenges and open issues regarding Ambient Intelligence (AmI) need to be properly addressed spanning across various conceptual and practical topics. First and foremost, one of the initial visions of AmI was to empower people to become maximally productive; however, in the recent years that vision has been revised as it was observed that people require a balanced approach where technology supports them instead of driving them [6] and AmI is referred to as "meaningful digital solutions that contribute to a prosperous and sustainable development of mankind". Additionally, [45] stresses that the initial concept has not been fulfilled yet, mainly because the proposed prototypes were technology-centric, rather than user-centric, and that techno-visions only look at what is technologically feasible and have a one-dimensional account of how social change occurs. Therefore, the original vision gets updated towards a people-centric design named **Synergetic Prosperity**, where people become the drivers and AmI technology is just the vehicle. Putting emphasis on the user in AmI innovation research plays a key role in the development of related applications and services, therefore prototyping tools that enable the quick generation of AmI Scenarios that users could experience or even let them create on their own, are absolutely necessary to maximize the acceptance of AmI.

Various challenges exist from an engineering perspective as well. [362] argues that communication will change from Human-to-Human (H2H), to Human-to-Technology (H2T) and finally Technology-to-Technology (T2T) will dominate. Then, the following question raises: "why do two objects have to 'talk' to each other? why does the fridge have to talk with the toothbrush?"; in fact, the respective IoT applications have to communicate with each other in order to collaborate. That exact observation outlines the need for better **horizontal integration between application layer protocols** [15]. Towards the unified environmental programming [6], the following aspects have to be ensured:

- End-to-end Interoperability based on Service-Oriented Architecture (SOA) to exploit integration with Internet and empower interfacing with wide range of technologies, devices, platforms and networks [7, 15, 337, 362].
- Heterogeneity by requesting each component to provide a system composition interface

that specifies not only its input and output but also relevant QoS properties and constraints [7, 337].

- Dynamics and Scalability by enabling the integration of new devices, services and functions for customers without negatively affecting the quality of existing services [7, 15].

- Software principles, computer architectures, middleware technology, real-time system abstractions and event triggers that meet the development needs of the next generation of AmI systems and in particular CPS, including their reuse through application-oriented designing approaches [28, 215, 227, 337].

- Enhanced context management mechanisms including: automatic configuration and discovery of sources, approaches to facilitate acquisition, modelling, reasoning, and distribution, sharing solutions and provision of Sensing-as-a-Service (S2aaS) and Context-as-a-Service (CXaaS) models [295].

- Security, privacy and trust via regulatory safeguards that dictate how "smart things" are used, in our everyday lives, when should they be switched off, what should they be permitted to hear, see, and feel and to whom they are allowed tell about it [6, 15, 48, 225, 337].

- QoS, reliability and predictability should be improved and guaranteed in the dimensions that matter (e.g., in real-life situations rather than isolated test scenarios) [6, 216].

- Support for the emerging paradigm of Social Internet-of-Things (SIoT) that envisions scenarios where: the users will be capable of interacting with IoT in the same manner they use the social network services and comment on the messages with natural language and "smart objects" will be presented as "beings" in social networks that could establish their own relationships [144, 314, 315].

2.2 Communication Frameworks and middleware

The combination of the Internet and emerging technologies, such as near-field communications, real-time localization and embedded sensors, transform everyday objects into smart objects that can understand and react to their environment [200]. Subsequently, the key challenges towards implementing AmI environments shifts from the technology perspective on how to build the necessary hardware, to the software perspective on how to build decentralized interoperable systems of (i) autonomous physical or digital objects augmented with sensing, processing, and network capabilities, and (ii) ubiquitous services that can efficiently and effectively interoperate in order to facilitate the realization of the AmI vision.

The authors of [116] in 2005 classified 29 well-known systems which reflect fundamentally different approaches to building ubiquitous computing environments under the following main categories of Augmented Reality, Intelligent Environments and Distributed Mobile Systems. With respect to their underlying technologies, Transmission Control Protocol / Internet Protocol (TCP/IP), Hypertext Transfer Protocol (HTTP) and Common Object Request Broker Architecture (CORBA) are amongst the most prominent technologies that are used to empower the creation of such environments; however, consensus has not been reached so far, as each different approach has its own specific goals and strengths. RFID Chef [213] highlighted the importance of event-based communication, as it was one of the first attempts to rely on contextual events for detecting state changes in the real world and triggering virtual counterparts to simulate state change.

The authors of [194] introduced the concept of reflective middleware model as a principled and efficient way of dealing with highly dynamic environments that supports the development of flexible and adaptive systems and applications. In the reflective model, middleware is implemented as a collection of components that can be configured and reconfigured by the application. The middleware interface remains unchanged and may support applications developed for traditional middleware, but most importantly it enables the creation of next-generation applications that can adapt to changes in the environment and be customized to fit into heterogenous devices. In addition, system and application code may inspect the internal configuration of the middleware and, if needed, reconfigure it to adapt to changes in the environment through meta-interfaces. In this manner, it is possible to select networking protocols, security policies, encoding

algorithms, and various other mechanisms to optimize system performance for specific and often unpredictable contexts and situations.

In [368] the need for creating hierarchies of services is outlined, by suggesting a coherent classification of IoT services based on their relationship to a physical entity and their life-cycle. In particular, with respect to the former, they define: (i) Low-level services that make the capabilities of the devices or the resources accessible to entity services or integrated services, (ii) a Resource service which is capable to make or provide the actions a resource is capable to execute, (iii) Entity services that constitute the heart of IoT systems and are compositions of low-level services, and (iv) an Integrated service that work with Entity services and compose them with non IoT services. Regarding service life-cycle, they classify a service as: (i) Deployable: its description exists in a service repository, but an appropriate runtime environment is not yet assigned, thus a service locator is not available in the service registry, (ii) Deployed: a service is already in the field, but not yet ready to be used as further steps are necessary to make it operational (e.g., technical, economical) and (iii) Operational: as the name suggests, the service is already deployed (if applicable) and ready to be used.

With respect to middleware technologies, currently many common practices and standards exist [15] in terms of: (i) application protocols (e.g, Constrained Application Protocol (CoAP) over REST, Message Queue Telemetry Transport (MQTT)), (ii) service Discovery mechanisms (e.g, multicast Domain Name System (mDNS), DNS Service Discovery (DNS-SD)), (iii) infrastructure Protocols, (iv) security solutions, and (v) interoperability protocols (IEEE 1905.1).

The ultimate middleware technology will have to satisfy all the following functional and non-functional requirements [316, 319]. From an architectural perspective it should be: (i) abstract in terms of programming (over heterogeneous input and output hardware devices, hardware and software interfaces, data streams and data types, physicality,the development process), (ii) interoperable, (iii) service-based, (iv) adaptive, (v) context-aware, (vi) autonomous, and (vii) distributed.

From a functional perspective it should facilitate: (i) resource discovery, (ii) resource, (iii) data, (iv) event, and (v) code management, and should satisfy the following non-functional requirements: (i) scalability, (ii) real-time, (iii) reliability, (iv) availability, (v) security and privacy, (vi) ease of deployment, and (vii) popularity.

In general, most middleware solutions can be classified under the following types [179] (which are similar to the programming abstractions of [316])

- **Transactions:** are contracts that guarantee a consistent system state transition and are used in various distributed application domains (e.g., applications centered on databases, telecommunications and safety critical systems). Technically, a transaction is a unit of coordination among sub- systems, which in its most general form respects the ACID (Atomicity, Consistency, Isolation, and Durability).

- **Tuplespace-based middleware:** in concrete terms, a tuplespace is globally shared among components, and can be accessed for inserting, reading, or withdrawing tuples. The important characteristic of the tuplespace model is that it introduces complete decoupling among components, in terms of both space and time.

- **Message-Oriented Middleware (MOM):** can be seen as particular instance of tuplespace-based middleware where tuples are implemented as messages and the space is implemented through distributed message-queues. In this case, parties communicate with each other by publishing, selecting and reading queued messages. MOM can be further classified in two categories according to the message selection mechanism that is provided: (i) Queue-based middleware, where messages are selected by means of queue membership, and (ii) Publish/-Subscribe middleware, where messages are selected by means of predicates. MOM provides functionality to publish, select and deliver messages with properties such as persistence, replication, real-time performance as well as scalability and security.

- **Remote Procedure Call (RPC):** as a pioneering coordination paradigm for distributed computing, Remote Procedure Call (RPC) is a protocol that allows a component to invoke procedures executed on remote hosts without explicitly coding the details for these interactions. In this context, a RPC middleware offers services for (i) generating client/server stub, (ii) marshalling/unmarshalling data (e.g., input parameters and re- turn values), (iii) establishing synchronous communication, as well as for (iv) ensuring non-functional properties.

- **Object and component oriented middleware:** just like procedural programming evolved towards object-oriented (OO) and further component-based (CB) programming, OO and

CB middleware represent the natural evolutions of RPC-based middleware. Specifically, OO and CB middleware provide the proper abstractions for exploiting the respective programming paradigms in a fully distributed environment. In particular, OO (CB) middleware offers tools that allow engineers to (i) generate stubs from the object (component) interface specification, (ii) obtain the reference of the remote object (component) to interact with, (iii) establish synchronous communication, and (iv) invoke requested methods (operations) by marshalling and unmarshalling exchanged data.

- **Service-oriented middleware:** a further step in the evolution of component-based programming is towards the Service Oriented Architecture (SOA) paradigm that supports the development of distributed software systems in terms of loosely coupled networked services. In SOA, networked resources are made available as autonomous software services that can be accessed without knowledge of their underlying technologies. Key feature of SOA is that services are independent entities, with well defined interfaces, which can be invoked in a standard way, without requiring the client to have knowledge about how the service actually performs its tasks.

Early 2000 - 2012

Since the inception of the Ubiquitous computing and AmI vision, researches had put various technological approaches under the microscope in order to identify the most suitable option. Before 2012, a clear trends exists towards the incorporation of well-established solutions coming from the domain of distributed computing and agent-oriented domains (e.g., Java Agent Development Environment (JADE), CORBA, Web Service Definition Language (WSDL)). This section presents key contributions to middleware technologies based on such a trend.

In [299], a web-based middleware is introduced for smart spaces that relies on technologies used in Internet services so as to simplify the creation of ubiquitous services. The proposed smart space network is based on a decentralized, local IP-based network which includes devices and services provided by the smart space owner. It consists of four layers, namely: networking layer, base platform and communication layer, platform services layer (that run on different run-times), and the Smart space application layer. A Zero Configuration (Zeroconf) mDNS mechanism is used

to facilitate device discovery, whereas custom technology-specific proxy components are used to connect non-IP devices to the network. Finally, HTTP and Web Services are used as the primary means for integrating software across devices in the smart space. Despite being a full-fledged solution, compared to AmI-Solertis it does not report to the developers the available functionality since it lacks any formal API specification, and does not support event-based communication.

An agent based middleware framework based on the JADE platform was introduced in [346], which can ease the implementation of sophisticated context-aware services in appropriately configured in-door environments (called "smart rooms") that contain video and acoustic sensors which provide elementary context cues. The introduced agent framework provides functionality for service access control, personalization, context modeling, as well as of dynamic use, control and management of sensors and actuating devices, and has been augmented with fault tolerance capabilities ensuring that failures are timely detected and restored (e.g., migrate an agent to a different execution environment upon detection of problems with its availability). The fact that it is based on the Java platform limits the potentials domains of use, but mainly the API of various sensors and devices cannot easily change as it requires redistribution of the agent library and recompilation of the consuming app.

The AMUN architecture [374] which consists of the following four components: the Transport Interface, the Event Dispatcher, the Service Interface and Service Proxy, and the Autonomic Manager. The Transport interface uses Juxtapose (JXTA) [146] to decouple the middleware from the underlying communication infrastructure, whereas the Event Dispatcher is responsible for message delivery across the various services, approaches alternative to traditional RPC. The Service interface is implemented by every service that participates in AMUN and is used to receive the messages that trigger the invocation of its functions, and finally the Autonomic Manager controls the AMUN nodes to realize their self-x mechanisms (i.e., self-configuration, self-optimization, and self-healing). The AMUN architecture is an interesting concept, however its development has been suspended since 2011. Moreover, every service that wishes to integrate in AMUN has to obey certain protocols with limited support (i.e., JXTA), which excludes the integration of services offered by platforms where the required libraries are unavailable.

The Ubiquitous Mobile Agent System (UbiMAS) [27] relies on a peer-to-peer communication mechanism (i.e., JXTA) to achieve a decentralized application structure. Messages are sent over

so-called uni-directional (i.e., only send or receive), bidirectional (i.e., send and receive) or propagate (i.e., multi-cast) pipes. Because of the event-based mechanism, every service must register itself to the Service Communication Layer to receive messages. Finally, a Pipe Abstraction Layer is used to standardize the services' access to the common communication layer, which notifies the interested services (i.e., subscribers whenever a message arrives and is ready for consumption. UbiMAS as well relies on the JXTA which is not a cross-platform library and no longer maintained.

The ICrafter system [298] aims to aggregate service-specific user interfaces under a unified environment, thus facilitating their use. To address service heterogeneity, ICrafter requires for very service to describe its offered functionality using the Service Description Language (SDL) XML-based language. In terms of service discovery and remote invocation, the EventHeap system is used, which maintains a list of the available services and offers an global event-based communication system where the interested parties subscribe to events matching their desired specified pattern. Compared to AmI-Solertis, the ICrafter only consolidates interfaces to consume remote services. Every service is required to be strictly defined, the framework only offers event-based communication (RPC is not supported), does not support aggregation at a service level to expose new components, nor empowers definition of dynamic behaviors.

In [402] the Reconfigurable Context-Sensitive Middleware (RCSM) proposed in [403] is expanded to simplify the development of situation-aware application software, enhance reusability and and achieve runtime reconfigurability. A declarative Situation-Aware Interface Definition Language has been developed to specify the Situation Awareness (SA) requirement during development, whereas runtime support is provided through the customized RCSM Object Request Broker. To develop SA application software, application developers need to first identify and formally encode their SA requirements (e.g., what contexts need to be collected, what are the situations of interest, what actions should be triggered in certain situations, how the triggered actions should be scheduled), generate the application skeleton and implement the desired business logic. Compared to AmI-Solertis, such approach does not simplify API definition and is not cross-platform (it only supports Windows CE.

Gaia [321] is a distributed middleware infrastructure (built on top of CORBA [261]) that coordinates software entities and heterogeneous networked devices contained in a physical space. It exports services to query and utilize existing resources, to access and use current context, and

provides a framework to develop user-centric, resource-aware, multi-device, context-sensitive, and mobile applications. Gaia components are distributed objects and require communication middleware to support remote interaction. Its five basic services are the Event Manager Service, Presence Service, Context Service, Space Repository Service, and Context File System. Some of the services are built on top of existing middleware services (e.g. Event Manager), while others are extensions to the communication middleware. Moreover, Gaia is extended via ontologies [309] to ensure that different agents in the environment have the same semantic understanding of different context information. This allows better semantic interoperability between different agents, as well as between different ubiquitous computing environments. Gaia is a quite powerful platform that offers similar functionality to AmI-Solertis; however, its main drawbacks are the hard requirement to recompile every consumer of a remote API in case of an update, whereas its architecture limits its scaling capabilities.

The authors of [128], by comparing various middleware solutions for smart environments(including Robot Operating System (ROS) [304] iRoom [62], Gaia, Ambient Agoras [358], Voyager-2Wear [211], etc.) outline the need for a framework to, automatically rather than manually, manage a huge number of cooperative services or smart objects and suggests that Cloud Computing and Agent-oriented methodologies could be exploited as basis for that objective. In [145], the Sunflower service execution platform is described. It relies on web services technology to provide information and functionalities of physical objects to business processes by supporting the composition of Simple Object Access Protocol (SOAP) services via Web Service Business Process Execution Language (WS-BPEL) workflows. Appropriate REST/SOAP proxies are used to permit REST compatibility, since WS-BPEL relies on SOAP web services. In relation to AmI-Solertis, it also uses agent-oriented technologies that can scale well, however it does not support event-based communication which constitutes a key attribute in AmI environments.

The Context-Aware Middleware for Ubiquitous computing Systems (CAMUS) middleware [267] which envisions a solution that not only focuses on providing context composition at the software level, but also facilitates dynamic features retrieval at the hardware level by masking the inherent heterogeneity of environment sensors. Different reasoning mechanisms are incorporated in CAMUS as pluggable services. Ontology based formal context modeling using Web Ontology Language (OWL) is described. With a systematic approach, CAMUS is proved to be a flexible and

reusable middleware framework. Context Aggregators register with the registration service to provide information about the context they can deliver, whereas interested applications and agents query the registration service to find services of their interests and continuously communicate the ontology repository to retrieve the latest values. Putting CAMUS and AmI-Solertis side by side, it becomes apparent that the latter offers minimal support for service aggregation and dynamic behavior definition, whereas it does not inherently support asynchronous events. However, it relies on reasoning to assist developers in using the appropriate services, based on their needs.

WComp middleware model [370] federates three main paradigms: event-based Web services, a lightweight component-based approach to design dynamic composite services, and an adaptation approach. Event-based web services are distinguished into basic and composite services. The lightweight component-based paradigm is based on an internal lightweight components to assembly event-based Web services and to design the interface of a new higher-level composite service. Finally, adaptation is based on an extended model of Aspect-Oriented Programming (AOP) for adaptation advices and a weaving process with logical merging that is achieved through dynamically modifying the internal assembly of WComp components. WComp supports Java-based services only, delivers limited support on service aggregation, whereas asynchronous communication is not inherently supported.

The Essex Intelligent Dormitory (iDorm) [157] is based on embedded agents for realising ambient environmental intelligence within a ubiquitous computing environment, in a test-bed environment that combines a number of heterogeneous computational artifacts and networks. The various components are interconnected with the Lonworks platform (Echelon's proprietary network) and the 1-wire protocol for building automation, and are exposed via an HTTP interface. In addition, a fuzzy logic-based Incremental Synchronous Learning (ISL) is used as a life-long learning mode that learns the users' behavior and adapts to their needs, and controls the iDorm according to the user's preferences in a personalized and non-intrusive way. iDorm constitutes a prototype implementation of a module that introduces intelligent behavior in an environment using fuzzy logic. From an interoperability perspective, it relies on specific commercial solutions to achieve collaboration among its supported artifacts.

Mobile Collaboration Architecture (MoCA) [323] is extended in [89] with context information service to support hardware and software heterogeneity, context evolution, and deployment at

devices with different resource profiles. Appropriate bindings are implemented for the Java language to provide a uniform view of context types and data, so that the system can retrieve context information published by different sources and at different locations using a single primitive. The system also permits the inclusion of new context-aware services without adapting (and redeploying) the middleware to support the new context types. Comparing AmI-Solertis with MoCA, the latter only supports the Java programming language and does not easily support dynamic adaptation at run-time.

[201] integrated RFID tags to objects (e.g., jack hammer) to make them smart, activity-aware, policy-aware and process-aware. These object could record information about work activities and their own use, and communicate with each other in a peer-to-peer fashion so as to access their rule sets and can make collective assessments (e.g., does the use of the hammer violates its licensing rules). Nevertheless, as the authors state ad-hoc combination of high-level application models (e.g., workflows) is still not supported. Despite enabling interoperability across ubiquitous artifacts, this system constitutes a single platform rather that a generic middleware for AmI environments.

ubiSOAP [68] is a service-oriented middleware, which leverages wireless networking capacities to effectively enable the ubiquitous networking of services. ubiSOAP specifically defines a layered communication middleware that underlies standard SOAP-based middleware, hence supporting legacy Web Services while exploiting ubiquitous connectivity. Its architecture exploits a two-layer design, where the network-agnostic connectivity deals with network heterogeneity and provides multiradio networking, and ubiSOAP communication implements a multi-network overlay achieving both point-to-point and group transports among nodes in ubiquitous networks. Every service has to be formally specify its methods and QoS attributes using a custom SDL. Compared to AmI-Solertis, ubiSOAP introduces its own propriety custom custom language, it does not support events, whereas in terms of performance the SOAP protocols add a significant overhead even to simple calls.

The authors of [347] propose an application layer solution for interoperability that utilize device semantics provided by existing specifications (i.e., Bluetooth, Universal Plug and Play (uPnP)) and dynamically wrapping into semantic services. The Web Ontology Language - Semantic (OWL-S) enable users to create and execute complex tasks involving multiple heterogeneous devices in an

abstract format, rather than directly dealing with devices of different standards. Interoperability is achieved via adapters and execution modules that encapsulate the peculiarities of the different standards (e.g., parameter handling). In terms of smart behavior definition, this work only supports service orchestration that enables interoperation as a proof-of-concept, rather than permitting developers to create complex scenarios.

In [203] a middleware architecture is introduced that follows the paradigm of Service Oriented Architecture in order to empower the interoperability of Smart Objects (SOs) that integrate RFID. Every SO implements its functionality in its object-oriented language environment and exposes it using WSDL, which gets semantically described in OWL and classified as a specific abstract device. The "Service Management" layer collects the definitions of the SOs, classifies them under a category named "Abstract Devices" and enables object discovery, status monitoring, mapping of available services to objects and service configuration. The "Service Composition" layer provides the corresponding functionality required for the composition of plain or complex services by joining and combining services exposed by the Service Management layer. Service composition is realized in terms of workflows of business processes using appropriate languages (e.g. WS-BPEL, WSDL). Compared to AmI-Solertis, this middleware also follows the SOA paradigm, but it requires every service to formally define its functionality in WSDL which adds a significant overhead both at design time as the developers have to build and document the service, and at run-time due to the inherent nature of WSDL-based services.

A ubiquitous middleware is presented in [183] which, amongst others, integrates various kinds of sensors and sensor networks using web service technologies. A Common Device Interface encapsulates several adapters that facilitate the retrieval of sensing data from heterogeneous devices. A central gateway collects all incoming data and forwards them at specified time intervals, to minimize latency, to any interested party that has registered a listener to receive them. Moreover, a Context-aware Computing Layer processes the data obtained through sensor network and provides intelligent context using domain-specific ontologies described in OWL. In relation to AmI-Solertis, this middleware relies on ontologies to generate information mashup only, rather than enabling interoperability among services.

The DoAmI platform [21] was developed to serve as foundation for connecting services through a service-oriented middleware architecture on top of CORBA. All application services are required

to be compliant with a base service model, therefore they implement several administration methods for accessing the configuration containers, as well as for setting a configuration, binding remote services to local variables and starting and stopping services. However, in order to allow the programmer to concentrate on implementing the service's functionality, a predefined super class minimizes the platform-specific code that needs to be written. In comparison with AmI-Solertis, the DoAmI platform relies on CORBA, thus excluding any platforms that do not support CORBA.

In [172] an adaptive middleware design for context-aware applications that abstracts the applications from the sensors that provide context is proposed. Raw sensor data are passed to Context Providers, which are either software or hardware components that aggregate and interpret data to produce some higher-level context. The new data are then propagated to Context Services that unify heterogenous context providers and eventually generate the final context that is forward to the application level. The performance of these intermediate levels is monitored in Quality of Context (QoC) units that can be used to facilitate runtime adaptation when necessary. The layered distribution is an interesting concept, however this middleware mainly aims to aggregate contextual information rather than facilitating interoperability across heterogenous services.

2012 - Today

The literature review has revealed two distinct trends on how middleware systems have been implemented so far. In particular, before 2012 the majority of the approaches relied on pure software-oriented techniques (e.g., CORBA, RPC, JADE). After 2012 though, a clear shift towards web-oriented technologies (e.g., REST, WSDL) is identified, mainly due to their increasing use by major market players (e.g., Amazon, Google, Apple, Samsung, Bosch, LG) who inevitably shape the overall landscape.

In [210], a service-oriented discovery framework that facilitates cooperation between smart objects is reported. A metadata model, accessible through a REST interface, has been defined to describe features, services, and operations of network-enabled Smart objects. The framework includes a central service registry and a client-side library which allows the applications to interact effectively with the registry using objects and local methods, making the remote methods invocation completely transparent. Every compliant RESTful service is built using the Java API for

RESTful Web Services (JAX-RS) specification and the JSON format is used for metadata exchange. Nevertheless, in relation to AmI-Solertis it does not simplify the process of building and distributing the exposed API of a service.

An agent-oriented event-based framework for the development of cooperating smart objects is proposed in [127]. The framework implementation relies on the JADE middleware that provides an effective infrastructure for agent management and communication. In particular, cooperating smart objects are implemented as JADE agents which directly cooperate through ACL messages [129], and indirectly via topic-based publish/subscribe. In comparison with AmI-Solertis, it only supports Java, while the behavior definition is performed through inference rules that implicitly requires additional services to execute the actual logic.

The Dynamic Integration Middleware (DIM) framework ensures seamless interoperability and communication between heterogeneous components in a global CPS network [284]. By employing the Internet Protocol version 6 (IPv6), along with local and global communications, and performing protocol conversion, enables the communication of devices using different protocols and networks. In DIM, a device that wants to find an interoperable device in the network sends a search message to the multicast group, instead of sending it to all devices, while appropriate protocol conversion for diverse protocols are employed in different wired and wireless networks. DIM consists of the Actuator Management Module, Sensor Management Module, Controller Management Module, Message Module, and Network Routing Module. The modules have library functions that support interoperability between heterogeneous devices in a CPS network, and provide APIs required to build DIM products. Compared to AmI-Solertis, the DIM framework relies on abstract generic modules that do not specify their formal API, thus limiting the way developers can explore and use the available facilities.

BioBot [115] is based on several bioinspired heuristic mechanisms, and serves as a wrapper for a set of resources (abstracting data, functionality, and services) that facilitates the propagation of queries through the network in an autonomous manner. BioSpace is a layer that acts as a middleware between the BioBots and the underlying operating system, while different middleware instances running on different servers are interconnected in order to facilitate a unified environment and a singular spatial universe. Inspired by the natural world, where biological organisms aspire to maximize the energy gain and minimize the energy loss, BioBots which fail to maximize

their energy and reach a critical low are dying (i.e., removed from the network), while successful agents are rewarded with the ability to reproduce. BioBots are built using the Jadex Active Components (JAC) framework that not only enables Belief-Desire-Intention (BDI) agent implementation [311], but also supports Plain Old Java Object (POJO) programming for any type of agent (e.g., model-based, reflex agents) and asynchronous agent interaction. Finally, apart from any traditional services offered through an Automated Deployment Services (ADS) developed with JAC, the framework is capable of exposing any service as either a WSDL or RESTful service. BioBot only targets Java-based services whose API should be explicitly specified in WSDL, whereas it offers limited support for service aggregation in order to dynamically define the behavior logic, as any logic must be integrated within the smart object itself.

The VIRTUS Middleware [81] relies on the open Extensible Messaging and Presence Protocol (XMPP) protocol and Open Service Gateway Initiative (OSGi) to provide secure event-driven communications within an IoT scenario. Through XMPP Extensions (XEPs) the framework delivers a secure publish/subscribe middleware; in particular, it uses XEP-0050 (Ad-Hoc Command) to provide workflow capabilities for any structured interaction between two XMPP entities, and XEP-0060 (Publish-Subscribe) to allow the subscription to receive notifications from a particular information node, thus providing a scalable and real-time alternative to constant and expensive polling for updates. XMPP is also used in [131] to solve the problem of interoperability by using semantically annotated messages as transmission format that unify all heterogeneous sensor data, and a semantic query language for handling later registered or removed devices without changing ongoing processes. Due to the semantic models, all participating CPS devices and their messages are annotated in a unified way, thus offering a platform- and programming language- independent approach towards providing sensor data or sending instructions to actuators. Both these approaches rely on the XMPP protocol, which however does not provide any method towards API specification, thus forcing the participating parties to exchange additional proprietary resources to agree on the data types to be used, while it does not support synchronous communication.

By viewing smart things as terminals, instead of services, [171] proposes a resource-oriented middleware framework designed specially for heterogeneous IoTs. The middleware is considered as an IoT browser over a unified platform where every node is identified in the Sensor Model Language (SensorML) format and is assigned an URI, whereas the Sensor Web Enablement (SWE)

standard from the Open Geospatial Consortium [57] is employed to make all types of sensors, transducers and sensor data repositories discoverable, accessible and useable via the Web in real-time. As the authors state, their system expect that dynamic network discovery, query, selection, and on-demand provisioning of web services will be supported, as their own work only permits the integration of sensors as input sources.

The IoT@Work project [154] focused on a holistic approach towards a Plug&Play IoT solution for manufacturing through the development of self-configuration mechanisms, whereby devices are auto-configured and ready to cooperate with each other as soon as they are plugged into the factory network, self-adapting to changes in response to demands, faults, etc. Among other functional components, it envisages a specific middleware (the Event Notification Service (ENS)) based on RabbitMQ that provides a publish/subscribe event based communication model to support both one-to-many and many-to-many communication patterns, as well as dynamic coupling of devices, processes and services. The ENS functionalities can be summarized as follows: (i) asynchronous fire-and-forget communication pattern, (ii) distinct namespaces of events, (iii) event filtering capabilities, (iv) horizontal scalability and reconfiguration flexibility, (v) access control features, (vi) events semantic, and (vii) complete decoupling of publishers and subscribers. The ENS, therefore, brings the data to the interested parties, instead of bringing the parties to the data. This reversed approach in data provision has significantly impacted both the system's security and the control of what data are provided to whom. In relation to AmI-Solertis, the ENS implements an asynchronous message-oriented server that relies on external semantic ontologies to define the event data. It constitutes a scalable solution, but offers limited API documentation to the developers, as it mainly aims at collecting and forwarding the various events to external reasoning engines that implement the business logic. Thus, it does not focus on facilitating the interoperation between AmI components.

The authors of [401] combine CoAP [56] with the Ubiquitous ID (uID) architecture [202] that plays a crucial role for keeping the semantic knowledge and data required for practical complex IoT services. It provides a software framework for embedded appliance nodes, designed to reduce the burden of embedded appliance manufacturers, by providing an intuitive, consistent, and easy-to-use API in order to build RESTful services in addition to the low-level communication API. The proposed lightweight protocol targets resource-oriented applications run on constrained net-

works, resembles HTTP and empowers programmers to focus on data processing. Comparing AmI-Solertis and this framework, the latter focuses on mainly on embedded systems, but it does not facilitate smart behavior definition, since it does not provide additional information regarding the data types to be exchanged nor it supports events.

Mobile Sensor Data Processing Engine (MOSDEN) [292] is a plug-in-based IoT middleware for mobile devices that allows the collection and processing of sensor data on resource constrained devices, without programming efforts. Its architecture supports the Sensing as a service model by extending the Global Sensor Network (GSN) middleware [9] and proposing new strategies that make it more scalable and user friendly. MOSDEN is a true zero programming middleware where users do not need to write program code or any other specifications using declarative languages. On the contrary, via a plugin architecture, developers can create plugins allowing MOSDEN to communicate with for their sensor hardware, collect information about each sensor, and send it to a cloud-based IoT middleware. Along with MOSDEN, the CADDOT model is proposed [293,297] as a potential solution to the challenging task of sensor configuration. By using the aforementioned plugins, the sensor discovery and configuration process is automated. In terms of programming, MOSDEN is mostly an attempt to improve scalability through wrappers that collect information form heterogenous sensors, rather than enabling developers to build scripts that define the behavior of an AmI environment.

The authors of [160] acknowledge that event-based technology has played an important role in the middleware space to enable scalable software architecture based on its loosely coupled interaction model. Nevertheless, event-based systems assume a high level of semantic agreement between event producers and consumers, which is challenging for largely heterogeneous environments such as smart cities. To address these concerns, they extend an event-based architecture by incorporating semantics in the exchanged events. As a result, there is no need for rigid a-priori agreements, since subscribers can signify to the matcher to search either for the term used or any term semantically similar to it, by associating appropriate thematic tags during initialization. This feature is inherently supported in the AmI-Solertis framework via its ability to automatically generate documentation for any integrated AmI component.

The work reported in [50] aims to empower integration of heterogenous IoT devices and services by modifying a GSN to forward its data to Firebase [147] instead of simply storing them in

a private SQL database, through which an extensible custom Data Interpreter draws conclusions regarding the current status of the sensors and instruct the system to act accordingly. By interfacing with the mediator, developers can create mobile and web applications that are independent of the actual IoT infrastructure, as they only rely on well-defined stored information. In comparison with AmI-Solertis, this system only collects the various heterogenous data of an AmI environment in a database and executes reasoning to infer certain facts (e.g., if a device is on or off).

The DIMMER middleware [204] (based on the LinkSmart OpenSource Middleware [209]) aims to deliver a Smart City IoT platform by applying the microservice architecture style. DIMMER applications supports search and discovery of IoT devices through both lightweight queries over HTTP by mobile applications and SPARQL queries by Semantic Web clients. In addition, the middleware provides a Historical Datastore service for storage of sensor data, a Message Broker for Publish/Subscribe communication of sensor data, and a Service Catalog for service discovery. Using the Application Gateway pattern of microservice architecture [266], DIMMER implements an IoT Data Gateway service that provides convenient high-level APIs for different kinds of consumers by fanning out requests to the platform services and returning them in the most convenient form to the corresponding consumers. Putting AmI-Solertis and DIMMER side by side, the latter only focuses on aggregating data from various sensors and distribute them to any interested parties via a central message broker. Thus, its scope is far more limited than AmI-Solertis that aims to deliver a communication protocol that will both support heterogenous components interoperability and the creation of scripts that define the intelligent behavior of an AmI environment.

Cooperative Middleware Platform as a Service (COMPaaS) [233] is a web-based middleware platform to enable an easy and loosely coupled cooperation between applications and IoT devices, and also to facilitate the development of IoT applications. The architecture of COMPaaS is composed of three main cooperating systems: API, Middleware, and Logical Device. API is the system that provides methods to be used by applications that desire to use middleware services. Middleware is the system responsible for abstracting the interactions between applications and devices and hides all the complexity involved in these activities. Logical Device is the system responsible for hiding all the complexity of computing devices and abstracts their functionality. These systems are based on the Subscribe/Notify communication pattern. Nevertheless, COMPaaS still does not facilitate developers to search devices available in the middleware (based on their context charac-

teristics), and provide to users an easy and transparent method to use them. Subsequently, the COntext BAsed Search ENgine (COBASEN) [19] has been implemented to complement COMPaaS. It is a software framework composed of a Context Module and a Search Engine to address the research challenge regarding the discovery and interaction with IoT devices when large number of devices with overlapping and sometimes redundant functionality are available in IoT middleware systems. The Context Module is responsible for gathering the device context and related data from the middleware and send it to the Search Engine, and the Search Engine is responsible for indexing the device information and answer queries using its internal index. COMPaaS, in comparison with AmI-Solertis, mainly focuses on discovering data and devices that generate data and making them available in the respective search engine, rather than facilitating services reuse.

The Web Ecosystem of Physical Devices (EcoDiF) [98] IoT platform integrates heterogeneous devices to provide real-time data control, visualization, processing, and storage from the perspective of Systems-of-Systems. In EcoDiF, devices, information, users and applications are integrated to create an IoT ecosystem in which new ideas and products can be developed easily. The platform consists of the following modules: (i) Devices Connection, (ii) Visualization and Management, (iii) Collaboration, (iv) Storage, (v) Data Manipulation, and (vi) Applications. In particular, with respect to middleware technologies, the Devices Connection Module facilitates the connection of physical devices to EcoDiF by requiring the manufacturers to configure their devices according to the EcoDiF's specific API to enable the integration with the platform, whereas any exchanged data should be annotated using the Extended Environments Markup Language (EEML) to permit their manipulation. The Visualization and Management Module provides a Web interface that enables users to manage the connected devices, and finally the Applications Module provides an environment for programming and executing applications that can make use of the available data (feeds) in order to generate new information in the form of Web Mashups [153]. EcoDiF only focuses on generating mashups by requiring any compatible services to annotate their data using EEML, with no provision for service interoperability.

In [378] an architectural design of a smart-home context-aware middleware that supports cooperation among application developers is proposed, implemented as a Java package. The system is minimal as it focuses on context gathering, aggregation and distribution. Context providers reside on any computational device or server and either gather raw data, process and model it or

infer complicated context types using input provided by several context sources and providers. The Registration service is implemented on the server and is invoked by context providers to register their contextual capacities. Finally, the Context retrieval service enables any context-aware application to gain access to the available context providers. Compared to AmI-Solertis, it ony supports interoperation across Java applications that use its respective library.

[167] middleware architectures such as Web-of-Things, or service oriented paradigms are likely candidates for CPS. WebMed aims at facilitating the service-oriented architecture for physical devices. The middleware contains high-level, logical representations of physical devices, computing elements and software services that are not necessarily linked to physical devices. WebMed caters for three different types of users: administrators, service developers and end-users (i.e., application users). WebMed consists of five components: (i) Device adapter (named WebMed node), (ii) Web service enabler, (iii) Service repository, (iv) Engine, and (v) Application development interface. A WebMed node acts as an adapter and device aggregator that "standardizes" the heterogeneous devices' hardware, data structures, communication protocols and device control issues. It is responsible for consolidating underlying devices' data into a common model and controlling devices through the proprietary device drivers and APIs. The Web Service Enabler provides a mechanism for the data and functionality of the physical devices to be accessible as Web services, that is it is responsible for service- enabling the devices. Service repository stores web service for interacting with physical devices, and any other services that manipulate the stored data. The WebMed Engine is the core element, providing a runtime environment for all Web services and operations in the middleware using HTTP as the transportation protocol. Finally, the WebMed application component provides high-level management of interaction and composition of Web service components in the middleware. It serves as the user interface for developers and end users to invoke a Web service, to create a mashup application and composite services, to monitor a physical device, or to subscribe for alerting service of a Web service. WebMed also mainly focuses on mashups and does not provide details on how service integration can be done. It only supports service orchestration rather than definition of intelligent behaviors.

2.3 Programming AmI Environments

2.3.1 An Emerging Paradigm

It is evident from the literature that End-User Development (EUD) is an emerging paradigm and over the next few years, the goal of interactive systems and services will evolve from just making systems easy to use (even though that goal has not yet been completely achieved) to making systems that are easy to develop by end users [223]. By now, most people have become familiar with the basic functionality and interfaces of computers, but they are not able to manage any programming language, therefore, they cannot develop new applications (i.e., program creation) or modify current ones (i.e., customization) according to their needs [333].

EUD, from an economical perspective as well, leads to more efficient appropriation processes, by empowering users to adapt software to their specific needs. It motivates users to explore the initial and potential system functionality that typically becomes a catalyst to develop innovative work practices. Obviously, there are costs involved with EUD, as the initial cost of software development is typically more expensive when realizing an EUD environment; however, EUD subsequently lowers the costs of adaptation and encourages the exploitation of opportunities for organizational appropriation. Given the ratio between costs in software and its organizational appropriation, EUD has a considerable potential to enhance the productivity of IT investments [398].

2.3.2 Functional Requirements of an EUD Tool

Flexible software architectures are a prerequisite for enabling adaptivity. Approaches range from simple parameters, rules, and constraints to changeable descriptions of system behavior (metadata) and component-based architectures. A key property of an EUD-friendly architecture is to allow for substantive changes during run-time, without having to stop and restart or rebuild the system [223]. According to [31,64,95,318] some of the principles that an EUD environment should follow are:

- Scaffold typical designs
- Provide multiple views with incremental disclosure

- Make syntactic errors hard or impossible
- Support a spectrum of expertise in computational thinking by offering different layers of computational abstractions and using objects as language elements (e.g., in a VPL environment shapes, colors, animations can differentiate components)
- Support incremental development and sharing of unfinished or evolving projects
- Allow the use of multimodal system input (e.g., using body and objects) and Immersion to help them experience the results of their programming activity by directly manipulating and interacting with their artifacts.
- Help people to create useful components
- Help users to finish projects by subtle coaching without harassment
- Not teach how to program, but provide an ecosystem to support people in creating ideas, solutions.
- Facilitate decomposable test units towards investigating the correctness of the programs end-users create (e.g., systematic "white box" testing, automatic test generation, assertions in a form of postconditions that also serve as preconditions, fault localization).
- Build community tools and facilitate collaboration between users with various roles (e.g., support interaction with software professionals)
- Integrate development tool with knowledge services that support users them in the creative identification of the most appropriate components to be integrated.

2.3.3 Models and Concepts

The authors of [245] consider Mental models, Ontologies, and Application Software mechanisms (i.e., editing tools in the most commonly used form that support the establishment and management of applications) as EUD tools that facilitate the development of AmI applications. They argue that the acceptance of EUD tools significantly relies on whether they offer the following

elements: Automatic interface generation, Programming by example techniques, High level abstractions and Metaphors (e.g., puzzle-based programming).

From a typology of how people can potentially create and customize applications, services and objects on top of IoT, [320] elaborates that three concepts form a basis for new creation paradigms in such smart spaces, potentially leading to new Do-it-Yourself (DiY)-enabling functions in IoTservice creation environments: the Call-Out Internet of Things, the Smart Composables Internet of Things, and the Phenomena Internet of Things. Call-Out Internet of Things entails that the network gets the capability (for people) to expose and exchange call-outs in the user surroundings, as a means to provide individual users and communities with a locative, distributed communication with objects in the environment. The Smart Composables Internet of Things is defined as a specialized instance of the Call-Out Internet of Things concept, where everyday objects get augmented with crowd- or industry-produced instructions and how-to's concerning how they have been or can be produced and composed and how their parts can be reused in other combinations, also in combination with other objects. Finally, the Phenomena Internet of Things implies that the network gets the capability to capture "phenomena" in the user data, as a means to provide individual users and communities with feedback on patterns in their personal daily life, or in the broader society. Of crucial importance in this respect is which patterns are of real value to users, implying that close user involvement in the iterative identification of these phenomena is essential for maximizing the potential of adoption in user-generated or other applications. AmI-Solertis aids end-users (through its Authoring Studio) to implement the most complex version, the Phenomena Internet of Things.

In [364] a conceptual model for synchronous applications in ubiquitous computing environments is proposed. The BEACH framework provides the functionality for synchronous cooperation and interaction with roomware components and devices, i.e., room elements with integrated information technology. A multilayer infrastructure (i.e., task layer, generic layer, model layer, core layer) is adopted so as to make the implementation of the higher layers easier and provide a shared-object space allowing distributed access from multiple computers. Compared to AmI-Solertis, BEACH is a conceptual framework that must be implemented in a specific language, rather than a programming environment for AmI applications.

The benefits of using a framework approach to help end user developers create a variety of

adaptable applications with differing requirements are discussed in [99]. A web service and development toolkit provides a simple programming interface for interacting with an unknown database, while user trials showed that end users were able to use the system to quickly adapt and create applications. The results suggested that framework providers should design domain specific frameworks to provide a reusable and flexible structure for end user developers. AmI-Solertis encompasses this observation and realizes it via the concept of proxies that simplify the use of remote AmI artifacts and scripts.

In [163], a conceptual framework and software infrastructure that together addresses known software engineering challenges is presented. The branching model offers a novel and flexible means to insert context- and preference-dependent decision points into the normal flow of application logic. In contrast, the triggering model has been widely used previously in the programming of adaptive and context-aware applications, but is reformulated here to exploit the situation abstraction as a means of describing context changes. A case study showed that the conceptual framework and software architecture (which is organized into loosely coupled layers) are extremely successful in terms of facilitating the development of context-aware applications that are flexible, adaptable and autonomous. The use of the multi-layered context modelling techniques and the branching model mean permitted the evolution of the underlying context model without changing the source code, and both the developer and the end users can easily adapt and fine-tune the choice of communication channels simply by editing the preferences. AmI-Solertis accommodates the creation of such rules. Additionally, it also enables the integration of more advanced reasoning techniques (i.e., Machine-Learning) to further enhance decision-making capabilities beyond this traditional programming paradigm.

2.3.4 Approaches

The authors of [279, 280] describe a semi-automatic end-user programming approach that: (i) assists in the creation of easy-to-apply Semantic End-User Application Programming Interfaces for the APIs of legacy software components; and (ii) enables their usage in command-oriented and goal-oriented end-user application programming. A reference implementation is presented for the approach that provides visual programming tools and an agent-based execution environment

for smart space applications. The proposed tool offers an interesting approach, which however cannot easily scale to accommodate more complex scenarios. In addition, AmI-Solertis offers better exploration facilities that developers can use in order to find and select the components they would like to use.

A coherent way to create UbiComp applications that to consist of tangible objects, which carry the computing and networking technology required, is presented in [184]. By providing uniform abstractions and a supporting middleware, objects are treated as components of a UbiComp application, whereas the architecture is made directly visible and accessible via an Editor device (i.e., Personal Digital Assistant (PDA))that enables end-users to act as programmers. The possibility to reuse devices for several purposes -not all accounted for during their design- opens possibilities for emergent uses of ubiquitous devices, whereby the emergence results from actual use and people's creativity. The GadgetWorld architecture addresses heterogeneity of the devices as AmI-Solertis does, however in terms of editing facilities, since its "code editor" is offered only via a PDA, the functionality is -as expected- rather limited.

The IF-THIS-THEN-THAT paradigm works well when end users need to be warned or notified on a specific event, but uses a very simple language that has a quite low expressive power [38]. Therefore, an extension is proposed that gives to end-users the possibility of setting triggers that do not depend just on one event, but also on other conditions. In particular, rules can be triggered: (i) by an occurrence of time events, or (ii) periodically after as specified period, or (iii) delayed with respect to triggering event of a rule. With respect to this extension of the IF-THIS-THEN-THAT paradigm, AmI-Solertis by design supports all these alternatives.

The study [54] suggests that the type of the programming task (i.e., creation or modification), plays a important role in the success of using a programming environment. Specifically, the observation of a relevant user study relieved that participants felt more confident using the visual programming environment while executing a modifying task, rather than building a new program from scratch. Nevertheless, additional findings suggest that visual programming may be good to get users started, but at some point they will want to switch due to the various limitations of the former (e.g., poor debugging support). Therefore, the authors suggest to provide both representations at the same time. AmI-Solertis embraces those findings and offers both a visual and a textual editing environment.

In [199] the results of a study that evaluated an approach of teaching computer science principles using IoT in combination with a visual programming environment are reported. Students quickly developed skills in developing IoT applications that either took in physical input from sensors or created an effect in the physical world. The IoT teaching infrastructure resulted in students raising a very small number of issues and was key to the overall success towards teaching the foundations of the IoT. AmI-Solertis already offers a basic visual editing facility, while the support of alternative programming paradigms (i.e., programming via gaming) is part of its planned future work.

In [205] an architecture for the IoT that unifies the Web of Things idea with the requirements of constrained mote-class devices is proposed. A RESTful runtime container (named Actinium) allows for dynamic installation, update, and removal of scripts. Apps are modelled as resources themselves and can provide parameters, status information and results through RESTful interfaces. Complex applications are implemented by mashing up such resources, provided directly by "things", other apps, or classic Web services. Security is provided through traditional Internet standards, but with a paradigm change in how applications are signed. AmI-Solertis also supports dynamic installation, update, and removal of scripts, which are however wrapped in appropriate proxies that simplify their use, while permitting their run-time adaptation if necessary (e.g., dynamic replacement of a failed service). Moreover, it introduces a web-based authoring studio through which users can not only explore the available service, but also edit their scripts and manage their deployment.

In [294], the Context Awareness for Internet of Things (CA4IOT) architecture that aims at helping users by automating the task of selecting the sensors according to the problems/tasks at hand is proposed. It focuses on automated configuration of filtering, fusion and reasoning mechanisms that can be applied to the collected sensor data streams using selected sensors. The objective is to allow the users to submit their problems, then the proposed architecture understands them and produces more comprehensive and meaningful information than the raw sensor data streams generated by individual sensors. AmI-Solertis permits end-users to manually select (based on their own criteria, preferences and needs) which services they would like to use; however, by design it enables the installation of external modules that can provide high-level recommendations and suggestions.

The authors of [156] argue that IoT applications are becoming more and more popular but not interoperable with each other, thus they propose the Machine-to-Machine Measurement framework to: (i) build IoT applications, (ii) assist users in interpreting sensor measurements, and (iii) combine domains with each other. The framework is based on semantic web technologies to explicitly describe the meaning of sensor measurements in an unified way to ease the interpretation of sensor data and to combine domains. A smartphone application is used as the user interface with the developer. Depending on the requirements, the user can select a scenario and the smart device will employ semantic web technologies to deduce new information from the sensor data and propose a list of recommendations to the user. The recommendations may include actuation which leads to intelligent control of surroundings based on the selected sensor data. As already mentioned, AmI-Solertis currently offers a manual selection facility, which can be further enhanced as needed.

The T4Tags 2.0 system [42, 385] aims to enable users to design and program by themselves the trigger-based behaviors of sensors and actuators, as well as the physical configuration of the technology in the domestic space. The authors suggest that in today's increasingly complex technological landscape, it is critical to give people tools to solve their own problems. The system, supports user programmability of tokens through an adaptive Web app, and it lets users compose, edit, and share instructions that define the behavior of a programmed token. Such instructions (i.e., "recipes") let users compose certain conditions that define triggering constraints by linking different events with conditional operators. Predefined triggering events are provided, and custom events can be defined through tangible interaction with a programming-by-demonstration approach, in which the system learns new triggers by "watching" the user perform or demonstrate them (e.g., motion gestures). Predefined gestures are provided, such as "shake", "swipe", and "up-down". Finally, T4Tags 2.0 also provides crowdsourcing tools to share implemented recipes. Despite being an interesting approach, AmI-Solertis already address most of its shortcomings already identified by the authors, namely it supports other behavior definition paradigms that enable automatic and adaptive activation of behaviors depending on observed patterns in addition to the trigger-action approach, and extended support for any kind of intelligent token independently of its capabilities.

AppsGate [84] was designed to empower people with tools to augment and control their home

that also includes debugging aids for non-specialists. AppsGate was designed using a systemic and holistic approach to provide reliable operation under real-world conditions, while the goal was to support both basic, mundane activities and more creative activities, such as end-user programming. AmI-Solertis offers a similar graphical tool, which also enables the composition of advanced scripts and handles the entire development life-cycle starting from the introduction of an AmI component to its deployment and monitoring.

In [101] a prototype based on a new model for rule specification that includes a rich set of operators for coupling multiple events and defining temporal and spatial constraints on rule activation is presented. The composition model includes new operators for defining rules combining multiple events and conditions exposed by smart objects, and for defining temporal and spatial constraints on rule activation, which as the relevant studies confirmed suit the expertise of end users. AmI-Solertis, in addition to simple event-action rules, permits the creation of advanced scripts that extend the reasoning capabilities of the overall environment and deliver smarter behaviors than simple automations.

A method and a set of tools are presented in [143] that allow end users without programming experience to customize the context-dependent behavior of their Web applications through the specification of trigger-action rules. The environment is able to support end-user specification of more flexible behaviors than what can be done with existing commercial tools, and it also includes an underlying infrastructure able to detect the possible contextual changes in order to achieve the desired behavior. The resulting set of tools is able to support the dynamic creation and execution of personalized application versions more suitable for users' needs in specific contexts of use. As already mentioned, the AmI-Solertis Studio extends beyond rule editing and it manages the entire life-cycle of AmI scripts that shape the intelligent behavior of the overall AmI environment.

The findings of a longitudinal analyses regarding Trigger-Action Programming in the wild are reported in [377]. Even though the results directly highlight the continued growth of trigger-action programming in the real world and its relevance to a range of online services and physical devices, it also suggests the needs to (i) provide users with more support for discovering functionality and perhaps even managing recipe collections, (ii) integrate mechanisms to enable sensemaking of history and learning, (iii) eliminate ambiguity or temporal uncertainty, and (iv) offer debugging facilities. AmI-Solertis addresses the identified drawbacks by incorporating advanced tools (e.g.,

AmI components exploration, AmITest) in the authoring studio.

The Ambiance Platform [26] is a meta-level architecture that enables on-the-fly creation and execution of high-level user-authored programs. It leverages Artificial Intelligence methods to separate the representation of a program and its reasoning mechanisms from the actual executable code through appropriate model-to-code transformation. Among others the Ambiance platform offers a web-enabled end-user programming interface for enabling end-users to interconnect - without any assistance- distributed ambient devices that form a system of mobile agents. AmI-Solertis compared to the Ambiance platform enables easier integration and exploration of AmI components, whereas the provided editing facilities permit the definition of "smarter" behavior via more complex programs.

In [125] an extensive literature review has revealed that the majority of the tools that support End-User Development Tools for the Smart Home, namely: (i) Atooma, (ii) Bipio, (iii) GALLAG Strip, (iv) IFTTT, (v) itDuzzit, (vi) Locale, (vii) Twine, (viii) WigWag, (ix) We Wired Web, and (x) Zipato Home Management, are based on a rule-based paradigm for behavior definition, as the end users (home inhabitants) are provided with visual interfaces through which they compose events and-or conditions with actions, using structures like "if-condition(s)-then-action(s)" or "when-event(s)-then-action(s)".

In [225] it is argued that Instead of today's rigid programming paradigm - design, build, and run - intelligent ambients offer a defect-tolerant programming paradigm where system behavior monitoring is interleaved with application (re-)mapping and architecture (re-)configuring, whereas from an implementation perspective, the authors believe that energy, fault-tolerance and mobility are the three key concepts that deserve special attention when designing smart ambients. Their framework abstracts the applications from the sensors that provide context. Further applications define utility functions on the quality of context attributes that describe the context providers. Then, given multiple alternatives for providing the same type of context, the middleware applies the utility function to each alternative and choose the one with maximum utility. By allowing applications to delegate the selection of context source to the middleware, their middleware can implement autonomic properties, such as self-configuration when new context providers appear and resilience to failures of context providers [173].To that end, they have proposed an adaptive middleware framework for the provision of context information for context-aware applications.

In particular, context provision adapts to changes in the quality of information advertised by the providers, to new providers entering the network and present ones failing. AmI-Solertis acknowledges these arguments and by design extracts the business-logic code -relevant to contextual information- from the applications themselves and incorporates it on the middleware to enable scalability and resilience.

In [235] a software ecosystem is proposed that allows different-skilled users to develop location-aware services able to autonomously manage the Smart Home. These services control the environment in accordance with user-defined rules and the users' location, calculated by exploiting an indoor localization mechanism. In addition, to directly interact with smart devices, users can also define customized interfaces for mobile devices. Finally, a multi-protocol middleware allows both the services and the mobile applications to access the physical network hiding the underlying heterogeneities. With respect to EUD, the system provides simplified tools for the implementation of both location-aware services and graphical interfaces to interact with the environment through a mobile device. To do so, the user is provided with a development tool for services and interfaces. It allows to easily describe the home devices as well as the services and user interfaces that involve these devices and then define the business logic of the services that manage the Smart Home through a simple graphical interface. AmI-Solertis permits the creation of scripts that are not only location-aware, but take into consideration various contextual parameters originating from multiple heterogeneous sources (e.g., user's calendar, a specific application, history, user profile).

[234] presents a social networking platform which integrates social networking with automated deployment of applications on multi-clouds and with knowledge drawn from community-sourced information repositories. The implementation leverages two such repositories, the PaaSage repository and Chef Supermarket. The platform enriches user interactions with structured references to applications and their components, and mined information from execution data of real deployments. Mined knowledge is combined with user activity and profiles to provide personalized suggestions and hints. The main research question addressed is whether DevOps professionals perceive value in a social networking platform designed to leverage the aforementioned information repositories and thus whether they have incentives to use it, a fact that has been confirmed through three user evaluation studies, whose results confirmed that DevOps users see value in joining this online community and contributing to it.

2.4 Discussion

As the presented literature review confirms, an AmI environment is powered with many different technologies that coexist and cooperate in order to enhance the surroundings with the appropriate computational means, so to make it sensitive to human and object presence, and able to proactively and intelligently react to human needs. In fact, better horizontal integration between application layer protocols is needed, as the respective AmI components and applications have to communicate with each other in order to collaborate. Therefore, in order to aid developers in defining the intelligent behavior of AmI environments, a holistic approach that incorporates both an appropriate communication middleware and a unified programming platform is required.

With respect to the communication middleware, literature [33, 173, 259] converges that the basic functional components should facilitate: (i) Interoperation, (ii) Context detection, (iii) Device discovery and management, (iv) Security and privacy, and (v) Data volume Management. Currently HTTP and REST seem to be the most prevalent technologies that will empower communication devices, and platforms [409], whereas for supporting all types of AmI applications and for better security, privacy control and latency, middleware is slowly becoming available in the cloud as well as on the edge [268]. Even though many different approaches [120] have been proposed that address heterogeneity, interoperability, flexibility and extensibility, some open research challenges still exist [316].

In particular, in the context of this work, the proposed communication middleware should: (i) be real-time, scalable and cross-platform, (ii) rely on well-established technologies, (iii) support synchronous, asynchronous and event-based communication, (iv) facilitate resource discovery, management and update, (v) enable service composition and self-exposition, (vi) simplify service updates and deployment, (vii) streamline the introduction of existing external services, (viii) empower users to create versatile AmI applications targeted to their context, (ix) support self-* properties (e.g., self-adaptive), (x) be developer-friendly, and finally (xi) allow the introduction of new components that extend its functionality.

End-User Development (EUD) is also an emerging paradigm and over the next few years, the goal of AmI systems and services will evolve from just making them easy to use, to making them easy to develop by end users. It will be imperative to empower users to explore the initial system's

functionality, adapt software to their personal needs or develop new innovative applications. Currently, various tools exist to empower the creation of services, applications and systems for AmI environments, however none of them supports the entire development life-cycle of an AmI system (i.e., analysis, design, implementation, testing, deployment, maintenance and disposal). To that end, AmI-Solertis aims to deliver an authoring studio that will: (i) help users to create useful components, (ii) simplify service discovery, definition, and management, (iii) offer programming abstraction support, (iv) empower reprogrammability of AmI components, (v) scaffold typical designs, (vi) provide multiple visual representations, (vii) support a spectrum of expertise (e.g., make syntactic errors hard, context-sensitive hep), (viii) allow the use of multimodal system input (e.g., using body and objects), (ix) provide testing facilities, (x) facilitate collaboration between users and expertise sharing, and (xi) assist management of the AmI environment in real-time (i.e, installation and deployment of new components, monitoring of existing infrastructure).

Chapter 3

AmI-Solertis Requirements

The AmI-Solertis framework aims to empower developers to realize AmI scenarios through a scalable tooling infrastructure that supports them during the entire development cycle. In particular, AmI-Solertis supports (i) the exploration of the available intelligent facilities (e.g., technologically-augmented artifacts, computational resources, SaaS), (ii) the compilation of the desired intelligent behavior (i.e., business logic) through versatile authoring tools (e.g., source code editor, graphical editor, VR) (iii) the provision of detailed real-time monitoring of the intelligent infrastructure, and (iv) the facilitation of live logic modification at run-time (e.g., HDS, remote calls interception, replication, dynamic reconfiguration) to adapt the environment to the needs of the user and improve the overall Quality of Service (QoS).

To that end, Chapter 3 outlines the key functional and non-functional requirements [247] that the AmI-Solertis framework should satisfy, which have been collected through an iterative elicitation process during using multiple collection methods including: brainstorming, focus groups, observation, personas and scenario building.

3.1 Functional Requirements

FR- 1: Unify ubiquitous services that control devices and applications

AmI-Solertis should deliver a scalable and flexible infrastructure that would accommodate various types of heterogenous AmI artifacts [1], whose executables are set of files that include at least

[1] In the context of this work, an AmI artifact refers to an intelligent service or application that exposes part or all of its functionality in the AmI environment for further use

one file that can be executed either (i) directly by the Operating System (OS) (e.g., an executable application that can be launched by Microsoft Windows) or (ii) indirectly by launching an auxiliary runtime environment with appropriate arguments (e.g., Software-as-a-Service (SaaS) over Node.js runtime [371], a Java application over Java Runtime Environment (JRE) [148], a machine-learning toolkit over Python interpreter [327]).

Moreover, the infrastructure should be able to host any kind of service or application independently of its vendor; in particular, it should permit the integration of both custom and commercial solutions as long as their formal functionality specification complies to the AmI-Solertis guidelines and their data are available in the appropriate format. Furthermore, the overall infrastructure should provide appropriate hooks that will enable external libraries (i.e., plugins) to enhance its functionality with sophisticated facilities such as: software versioning (through a well-established version control system), semantic annotation of the integrated services to enhance interoperability, security and privacy, service indexing and quering, caching, integration of external service and module repositories (e.g., npmjs.org [332], Chocolatey [30]), etc.

Given that AmI-Solertis aims to integrate heterogenous and distributed artifacts, a unified communication mechanism should encapsulate the peculiarities of each different protocol and expose a common API for the developers to use; therefore, whenever an AmI script has to make a remote request to an external service, the code should make no distinction to synchronous calls that return immediately vs. asynchronous calls that return at some point later in the future. Finally, to support both proactive and reactive response from the environment, AmI-Solertis should facilitate event-based communication, so that artifacts could notify the environment at any time that something happened and trigger the respective handlers to respond accordingly.

FR- 2: Facilitate static and dynamic service integration, deployment and execution

AmI-Solertis should enable developers to integrate (in a static manner) new services at any time by simply uploading the respective archives that contains their executable files. Moreover, the system should be able to dynamically detect and incorporate any transient intelligent services (e.g., Mobile-Backend-as-a-Service (MBaaS) [328]) that appear in the ecosystem in order to either register their functionality during their first occurrence or mark them as active and ready to be used. To that end, appropriate services should be employed in order to facilitate the respective

advertisement and discovery process of such facilities.

Additionally, AmI-Solertis should automate the initial configuration process of AmI-Solertis enabled hosts, and simplify the deployment, configuration, and execution of any AmI artifact on them (including the infrastructure itself).In more detail, *host initialization* should be a single executable process (e.g., an installation wizard) that users will launch on any device that meets AmI-Solertis hardware and software requirement, to automatically bootstrap the essential runtime components and register it as an AmI-Solertis enabled host. Next, regarding *service deployment and configuration* in AmI-Solertis hosts, the whole procedure should be streamlined and remain transparent to the end-users via appropriate configuration managers and user-friendly interfaces.

With respect to *execution*, a user should be able to manually start, stop or restart any deployed artifact on a single host, while all of them should be instructed to run on startup and automatically recover in case of a failure. To that end, appropriate local services should permanently run on every host and manage the execution of the deployed artifacts. These local executors should notify the AmI-Solertis infrastructure about any currently running instances in order to facilitate health monitoring, logging, dynamic service discovery and resolution and centralized management. Local executors also expose their functionality in the ecosystem, thus enabling the creation of AmI scripts that could modify at runtime the overall ecosystem (i.e., specify which components should start or stop). Finally, AmI-Solertis should deliver appropriate analytics interfaces through which developers will be able to get rich information about the running instances (e.g., usage, metrics, graphs).

FR- 3: Empower the definition of intelligent behavior

AmI-Solertis should facilitate service composition both via orchestration and choreography. In general, through service composition the author of an AmI script should be able to use the available facilities in order to create new services. Service choreography is a global description of the interactions between multiple participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints; choreography employs a decentralized approach for service composition. Service orchestration represents a single centralized executable business process (the orchestrator) that coordinates the interaction among

different services from one party's perspective by invoking and combining the services. Therefore, through orchestration a single AmI script should be able to control the entire composition process of a new service, whereas through choreography a script could invoke another AmI script and so on.

In addition to alternative composition methods, AmI-Solertis should permit authors of AmI scripts to follow alternative pathways towards compiling their business logic that defines the behavior of the environment when certain stimuli of any kind occurs. AmI-Solertis should not limit the type of reasoning that could be applied in order to determine the next actions that need to be taken. As a result, reasoning could be directly embedded in the AmI script in the form of language specific conditional expressions, or it could delegate appropriate requests to external services that introduce advanced reasoning mechanisms (e.g., Machine learning systems, Rule-based reasoners, Deductive classifiers, Logic programs) in an AmI-Solertis compliant manner.

Finally, AmI-Solertis should employ appropriate techniques so that reasoning could be applied not only to determine the behavior of the environment, but also adjust and adapt the behavior of the AmI-Solertis framework itself. For that reason, the mechanisms that handle communication across remote artifacts should be easily extensible and programmatically modifiable so that AmI scripts could alter their functionality if necessary (e.g., intercept and discard calls to an under-performing service, re-route calls towards an inactive service to another equivalent).

FR- 4: Support the Software-as-a-Service cloud computing design pattern

AmI-Solertis should enable the creation of programs, called AmI scrips, which will use the available AmI artifacts in order to introduce intelligent software agents. The latter based on contextual information [301] (e.g., physical aspects, users, current activities) will govern the behavior of the technologically-enhanced environment. AmI scripts should be able to expand beyond their limited bounds as isolated programming units, towards becoming full services that expose their functionality in the ecosystem, thus implementing the SaaS cloud computing design pattern [118]. Therefore, AmI-Solertis should promote reusability and extensibility, by facilitating remote use of AmI scripts by third parties to build sophisticated domain-oriented scripts (similarly to how an SDK exposes its encapsulated functionality via public methods).

FR- 5: Deliver an online programming platform

AmI-Solertis should empower both developers (noviced or experienced) and end-users to create new AmI scripts that will define the behavior of an AmI environment. Towards that objective, appropriate graphical tools should facilitate the exploration of the currently integrated AmI facilities and the provision of detailed information about their capabilities and current use. In addition to traditional visualization approaches, immersive techniques (e.g., AR, VR) should be available to facilitate understanding of the surroundings since that might greatly affect the intelligent behavior (e.g., rely on readings of specific sensors to initiate certain actions).

With respect to source code authoring, AmI-Solertis should provide alternative solutions that can accommodate the needs of the diverse targeted user groups. In particular, a fully-featured source code editor and a visual editor should be made available so that developers can select based on their preference. In both cases, context-sensitive support and recommendations should be given if the developer asks for support; a recommendation system should suggest related artifacts, third-party systems, etc. that might interest the current user. On the same ground, social features (e.g., networks of developers of AmI scripts) could be employed to enhance collaboration and benefit from knowledge collected through crowd-sourcing [58]. Moreover, the overall programming environment including particular components (e.g., source code editor, artifacts explorer) should be exposed as service that third-party systems could easily integrate to make use of their functionality. Finally, additional auxiliary natural interfaces (e.g., a conversational agent) should be implemented to enable the end-users of the AmI environment to easily program its behavior.

FR- 6: Offer a standard library

A collection of common reusable AmI artifacts should be provided in the form of an AmI-Solertis Standard Library (ASLib) delivering ready-to-use components to the developers, thus relieving them from the burden of building them on their own. Specifically, the following useful modules should belong to the core libraries components that accompany a vanilla AmI-Solertis installation: (i) *AmI-Datastore* that implements scalable and persistent object storage and facilitates ubiquitous data exchange, (ii) *AmI-MemoryStore* that offers an easy to use document-oriented database that can hold JSON-like documents, (iii) *AmI-Context*, to store contextual information that need

to be shared across multiple authorized AmI scripts, (iv) *AmI-AI* that wraps in a AmI artifact well–known AI libraries (e.g., Naive Bayes, NLP, Anomalies detector), and (v) *AmI-Logger*, which enables environment-wide logging.

Additionally, AmI-Solertis should streamline any recurring processes that concern the creation or integration of a new AmI artifact in the ecosystem. Particularly, to ensure scalability the Components Directory Service (section 4.2) should decouple artifacts discovery and use from the underlying networking infrastructure; thus, any party that needs to use an AmI artifact or an AmI script can refer to it by name rather than using a volatile network address. AmI-Installer should configure a machine as an AmI-Solertis compliant host. AmI Configurator and Executor must encapsulate the necessary steps to download, install, configure and launch an executable in an AmI-Solertis host. Finally, AmI-Solertis Mobile Proxy and AmI-Solertis project seeds should assist and guide developers when building their programs outside of the AmI-Solertis framework.

FR- 7: Empower update of behavior rationale in a dynamic manner

The AmI-Solertis framework should be built following the microservice architectural style, as a collection of loosely coupled, autonomous, independently deployable, small, modular services, which run in unique processes and communicate through a well-defined, lightweight mechanism to serve the same business goal [266]. All communication between the services themselves should be made via network calls, to enforce separation between the services and avoid the perils of tight coupling. Therefore, the framework should be able to easily embrace different technologies either for the core components or for any external AmI artifacts that need to be integrated.

Following the aforementioned style, AmI-Solertis should ensure: (i) *scalability* by enabling new modules to be easily incorporated, (ii) *resilience* by limiting the side-effects of potential failures within the boundaries of a single component only, (iii) *ease of development* by permitting faster code deployment at a service level independently of the rest of the system, (iv) *organizational alignment* by enabling smaller teams to work on smaller codebases as then they tend to be more productive, (v) *replaceability* as developers are more comfortable with completely rewriting services when required and just killing a service when it is no longer needed, if a codebase is just a few hundred lines long and, (vi) *portability* by facilitating services migration. Finally, a similar principle should be applied at a host level as well; an AmI-Solertis host should expose its

functionality and enable remote configuration and use either directly from AmI-Solertis or indirectly from AmI scripts that use its offered functionality to manipulate its running state (i.e., PaaS).

FR- 8: Assist automation and continuous integration

AmI-Solertis should assist developers when building their AmI script or integrating their AmI artifacts in the ecosystem by automatically: (i) *generating* the necessary *auxiliary files* (e.g., service formal specification, configuration file template, service-specific methods and types definitions) (ii) in order to *prepare the executable* files, (iii) create the *installation package*, and (iv) *deploy* it on every designated host. With respect to deployment, AmI-Solertis should ensure that the Hot Deployment Service (HDS) will launch the new version of an AmI script or an AmI artifact if such an action is permitted (e.g., inactive, not in use, idle). Finally, the same level of support should be given to the developers, especially when uploading components that were not built directly in the AmI-Solertis system.

FR- 9: Enable real-time monitoring and deliver basic testing facilities

Apart from static management of AmI artifacts, AmI-Solertis should also support real-time health monitoring of active distributed instances. Therefore, it should collect and aggregate a large variety of data for all running artifact instances in order to estimate the health status of every individual host [266] and of the overall AmI ecosystem. Moreover, to accurately monitor the health of individual artifacts, AmI-Solertis should offer appropriate mechanisms through which the integrated artifacts will be able to provide extended details about their actual running status, via health-oriented functions (e.g., ping to determine whether the service remains responsive, report current workload).

Furthermore, AmI-Solertis should provide the necessary mechanisms and facilities through which developers will be able to: (i) verify the behavior of the environment when certain stimuli occurs (e.g., high-quality response, error-free), (ii) validate that this is the desired behavior that meets the customer's actual needs, and (iii) detect key aspects that will benefit from optimizations or fault-tolerance strategies. To that end, AmI-Solertis should incorporate an appropriate testing framework that will utilize the available meta-information regarding the installed artifacts and behavior definitions, in order to simplify the overall testing process for both novice and experi-

enced users. Finally, the provision of a sand-boxed environment in which testing can be executed without affecting the actual system and its artifacts, following the standard self-testing practices dictated by Continuous Integration (CI) [110], will be beneficial.

FR- 10: Empower agile development practices

A key aspect when it comes to programming of AmI environments is the ability to enable fast development of programs that dictate the behavior of the environment. In particular, people usually change their habits due to various reasons (e.g., personal, financial, work-related) [313], therefore the AmI's overall interpretation should be able to adapt to such changes. To that end, AmI-Solertis should provide an agile [241] development environment that would empower developers to easily modify the business logic of the environment and quickly re-deploy it. Moreover, the framework should rely on common programming approaches and a language with which developers are already familiar with, so as to widen the base of perspective developers who would be willing to use it; towards the same objective, alternative variants of the same language should be supported to take advantage of any variant-specific features (e.g., type safety, extensibility, automatic memory management).

Since AmI-Solertis aims to integrate third-party AmI artifacts, it should deliver to the developers the relevant tools to bootstrap them without having to set every minor detail manually (e.g, source code seeds). Furthermore, to inspire developers to create services of high-quality, it could optionally require them to properly document their code to increase its usability by others [222, 351, 412]. AmI-Solertis should also generate code that is compliant to those seeds to ensure uniformity and promote extensibility by permitting those external artifacts to eventually become converted to full-featured AmI scripts.

3.2 Non-Functional Requirements

NFR- 1: Performance Requirements.

All performance requirements must have a value which is measurable and quantitative. In particular, in AmI-Solertis: (i) Developers should be able to create and deploy a simple script within 5 minutes. (ii) Developers should be able to explore the available components within a couple of minutes. (iii) Search results should be rendered in under 10 seconds. (iv) Automatic creation of formal service specification should be complete under 30 seconds. (v) Collection at real-time of the health status of all nodes should finalize within 15 seconds. (vi) The user should be capable of using the system efficiently and effectively for the defined task after (at most) 1 hours training.

NFR- 2: Interface Requirements.

Software interface requirements include dealing with an existing software system, or any interface standards. Specifically: (i) Any formal OpenAPI Specification (OAS) [5, 94] file should be correctly parsed by AmI-Solertis, (ii) AmI-Solertis installer should execute in every computer with Microsoft Windows 10™ (iii) AmI-Solertis should easily integrate additional runtime environments (e.g., Docker [252]), and (iv) Any external services or applications should be uploaded as an archive of one of the following types: .ZIP™, .RAR™, .7z™.

NFR- 3: 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- 4: Documentation Requirements.

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

NFR- 5: Availability Requirements.

AmI-Solertis should be available 95% of the time. Even if certain components go offline, the re-

maining ones should operate as expected. If core components (such as the Components Directory Service) become unavailable, the system should be able to immediately report that and try to restore it by relaunching the faulty one or a clone of it.

NFR- 6: Compliance Requirements.

In general, AmI-Solertis should conform to well-established rules and standards regarding the supported technologies (e.g., REST [121], OAS, JSON [88]) as defined by relevant international committees and working groups (e.g., NIST, W3C), and apply any best-practices and approaches by the industry (e.g., Yeoman code generators [133] ,NPM, HATEOAS [345]).

NFR- 7: Deployment Requirements.

AmI-Solertis should minimize any deployment requirements for its core components. Additionally, it should offer the necessary facilities that will automate (to the widest extend possible) the deployment process for any externally integrated AmI artifacts, whereas deployment and execution of AmI scripts should be a fully automatic process.

NFR- 8: Interoperability Requirements.

AmI-Solertis should deliver its functionality both as a concrete system, but also as standalone components (i.e., interfaces and services) that are completely understood and can work with other products or systems, at present or future, in either implementation or access. Particularly, amongst other, the following components should be made available: (i) the API-Explorer (section 4.2) that provides access to the available AmI artifacts, (ii) the external artifact importer, (iii) the code editor, (iv) the health monitoring infrastructure, and (v) the deployment and execution services.

NFR- 9: Maintainability Requirements.

AmI-Solertis should permit its easy maintenance in the sense that: (i) faulty or worn-out components could be repaired or replaced without having to replace still working parts, (ii) efficiency, reliability, and safety should be maximized, and (iii) new requirements could be meet by introducing new micro-services, and (iv) updates should be verified and validated before their deployment.

NFR- 10: Platform Compatibility Requirements.

From a user perspective, the platform should be accessible via any modern Operating System (OS) or web browser. From an engineering perspective, AmI scripts 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- 11: Resilience Requirements.

In order to increase the the ability to provide and maintain an acceptable level of service in the face of faults and challenges to normal operation (i.e., resilience), AmI-Solertis should support distributed processing, network storage and software versioning techniques that could enable its easy replication or restoration.

NFR- 12: Reusability Requirements.

Reusability is the use of existing assets in some form within the software product development process. It constitutes one of the key principles of the AmI-Solertis, therefore assets, products and by-products of the software development life cycle including code, software components, test suites, designs and documentation should be available for further reuse.

NFR- 13: Scalability Requirements.

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. AmI-Solertis implementation model should offer such potentials for growth, meaning that if additional resources are offered, then capacity should scale proportionally.

NFR- 14: Transparency Requirements.

Transparency, as used in science, engineering, business, the humanities and in other social contexts, implies openness, communication, and accountability. Transparency is operating in such a way that it is easy for others to see what actions are performed. It has been defined simply as "the perceived quality of intentionally shared information from a sender" [334]. An AmI script is

accessible to all members of the AmI-Solertis, therefore developers are able to learn, criticize or improve potential faults and malfunctions.

Chapter 4

The AmI-Solertis Technological Framework

4.1 Overview

The AmI-Solertis system is built using a micro-service architecture style, hence its business capabilities are implemented and offered through a collection of loosely coupled services. By applying such an architecture style, the core system is flexible and scalable to be used as a backbone across a wide range of ubiquitous systems [74] and intelligent environments, where the objective could as simple as composing a new service by combining two existing ones, or as complex as building an intelligent management system for a smart house or a smart city [40].

The core system is composed of autonomous components that interoperate over the network using a variety of communication protocols and channels. Every service encapsulates the complexity of its internal mechanics and exposes a public API in the form of stateless operations, to be consumed by other services, in a uniform manner. To further enhance interoperability and discoverability, the core APIs are described in the OAS [5], while appropriate proxy libraries simplify their composition by wrapping any boilerplate (i.e., extraneous) code and enabling synchronous or asynchronous (via REST) and event-based (using Redis [69] as a message broker) communication.

Every intelligent component (e.g., device, GUI application, web service) is considered as a standalone AmI artifact operates autonomously, but at the same time exposes its functionality to the AmI-Solertis ecosystem in order to be used in conjunction with others towards creating pervasive, intelligent and personalized environment experiences. Towards introducing a universal

programming paradigm across the various target domains, AmI-Solertis will use the term *artifact* to describe either (i) headless services that offer pure data-related functionality [70] or (ii) fully-featured frontend applications that could be deployed on stationary or mobile artifacts within the AmI Environment and be remotely controlled as defined in the Software-as-a-Service (SaaS) model [376]. As a consequence, throughout AmI-Solertis everything (even the core AmI-Solertis components themselves) is considered to be some kind of a service and therefore can be used in AmI scripts that define behavior (i.e., the business logic) of the environment (Listing 4.1).

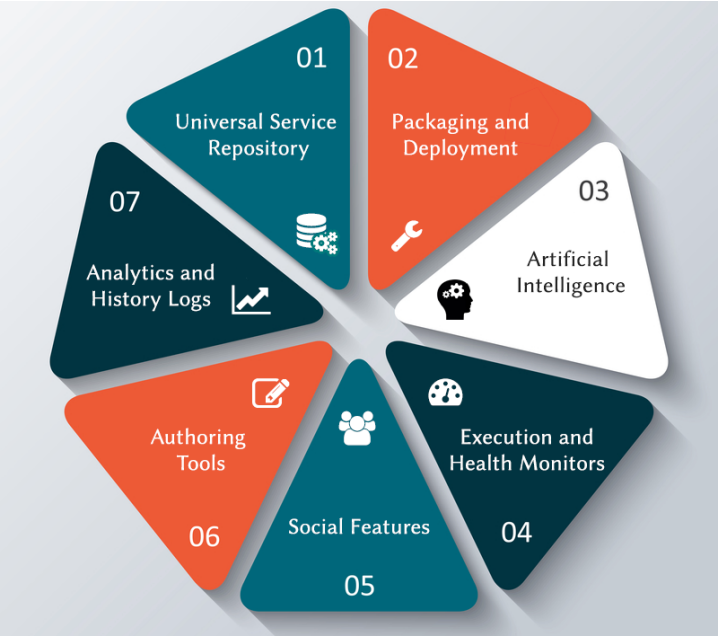

Figure 4.1: The seven basic modules that constitute the core of the AmI-Solertis system.

In practice, the AmI-Solertis system consists of various light-weight components (Figure 4.1) that are combined to formulate the core modules that belong in one of the following categories: (i) Universal AmI Artifacts and Scripts Repository, (ii) Packaging and Deployment, (iii) Execution and Health Monitoring, (iv) Analytics and History Logs, (v) Artificial Intelligence, (vi) Universal Datastore, (vii) Authoring Tools, and (viii) Social Features. Every category exposes its functionality in the form of a collection of APIs that could be consumed by other services in the ecosystem (e.g., frontend applications, behavior scripts.).

Universal AmI Artifacts and Scripts Repository: copes with artifacts and script management and enforces the common intercommunication scheme. It includes: (a) the components that inspect the installed services and extract their API specification in OAS scheme; AmI-Solertis supports various service types including: (i) custom REST-based services written in Javascript using Express [158], (ii) CORBA-based services built on top of a proprietary middleware named FORTH AmI Network Environment (FAmINE), (iii) AmI scripts scaffolded by AmI-Solertis tools„ (iv) iOS mobile applications that utilize the AmI-Solertis Mobile SDK, and (v) any other binary executables that expose their functionality over REST and their API has been manually expressed in the OAS specification; (b) the components that generate the appropriate software bridges, based on a OAS specification, to harmonize the programming approach independently of the service type; (c) the metadata storage mechanisms; (d) the service discovery tools and (e) the installation wizard that installs new services.

Packaging and Deployment: automates the overall packaging and deployment process for any AmI-Solertis compliant artifact or script. It consults the Universal Service Repository in order to assist the developer to properly configure and install the desired components in one or more physical or virtual hosts.

Execution and Health Monitoring: enables remote administration of AmI-Solertis services of any kind (e.g., core facilities, web-services, Graphical User Interface (GUI) applications, etc.) and monitors the overall health status of the system; if necessary, it takes corrective actions to ensure a proper Quality of Service (QoS). Interventions refer to the modification of the execution rules to compensate for under-performing components via custom automatically-generated circuit-breakers, or missing services via on-the-fly composition of semantically equivalents (at a proxy level).

Analytics and History Logs: contains datastores that log data of key events that happened within the environment. It can be used by AmI-Solertis compliant services or applications to store their own content. Mainly though, those data are analyzed by the core in order to evaluate the performance of the various artifacts and generate insights about potential optimizations.

Artificial Intelligence Linkage: provides access to well-known machine learning services [290] and permits the integration of solutions from the Semantic Web [46] and Cognitive Robotics [289]. Behavior scripts authored via the web-IDE can use them to create scenarios that employ artificial

intelligence principles to enhance the overall user experience in the intelligent space, whereas the AmI-Solertis core employs them to facilitate the internal decision making processes.

Universal Datastore: offers a universal mechanism to store both document-based and binary data (i.e., blobs [335]) regarding the AmI ecosystem and simplify their exchange with 3^{rd}-party services and systems [349].

Listing 4.1: An AmI script that minimizes energy consumption on an empty home

```
import * as SmartKitchen from '@ami/smarthome/smartkitchen';
import * as HomeCounter from '@ami/smarthome/homecounter';
class HomeController {
  constructor() {}
  async Init() {
    this.kitchen = await SmartKitchen.createSmartKitchen('mysmarthome');
    this.homeLocalization = await HomeCounter.createHomeCounter('
        mysmarthome');
  }
  InitializeLogic() {
    this.homeLocalization.onEmptyHouse((evt_payload) => {
      if (evt_payload.peopleCnt > 0) {
        this.minimizeEnergyConsumption();
      }
    });
  }

  minimizeEnergyConsumption() {
    this.minimizeKitchenConsumption();
    this.minimizeLivingroomConsumption();
    this.minimizeBedroomConsumption();
  }

  async minimizeKitchenConsumption() {
    // turn-off more appliances
    var cooktops = await this.kitchen.getAllCooktops();
    for (var i = cooktops.length; i <= 0; i-) {
      if (await this.kitchen.getCooktopStatus(cooktop.id).status == true
          ) {
        this.kitchen.turnOffCooktop(cooktop.id);
      }
    }
    // turn-off more appliances
  }
}
module.exports = HomeController;
```

Authoring Tools: consists of the various graphical components through which end-users can interact with the system in order to explore the available artifacts and program the behavior of a Smart Environment. In addition to the editing and project management tools, it includes a testing suite that can validate the behavior of a script and auxiliary facilities that can be used to define optimizations or fault-tolerance strategies.

Social Features: AmI-Solertis users can collaborate to share their experiences and best practices. To leverage the collective experience of its user community, the system facilitates the discovery of new and noteworthy content by promoting featured, popular and trending services based on their impact in the community. In addition, via user profiles the overall experience is personalized [195] by moderating the displayed content (e.g., recommendations, context-sensitive lists of recently viewed items for quick reference, etc.).

4.2 Universal AmI Artifacts and Scripts Repository

4.2.1 Hybrid Communication Protocol

This section will discuss the core technologies that AmI-Solertis employs in order to integrate heterogenous and distributed services and devices under a common "roof", so as to deliver a unified programming paradigm to its developers.

Building services for AmI environments implies that multiple different technologies and protocols will be used by the various technological components in order to define and expose their functionality. The deciding factor on which specific protocol will be used, in addition to any prospective standards and guidelines that suggest certain approaches [32, 341], is their technical capabilities from a hardware (e.g., network interfaces, processing power, battery-based operation) and a software (e.g., OS, runtime environment) perspective.

As already mentioned in chapter 2, in the past many alternative communication protocols have been used to address the issues of heterogeneity and interoperability, including solutions inspired by distributed computing (e.g., Java RMI, JADE, OSGi, CORBA) and Service-Oriented Architectures (e.g., HTTP, WSDL, SOAP, MQTT). Nevertheless, the emergence of Web 2.0 [278] and

the rapid incorporation of agile development methods to accelerate the introduction of new technologies and services to the market, pushed vendors to use simpler and more versatile service definition models and protocols, with Representational State Transfer (REST) being the prevalent option. In addition to its ease-of-use, REST is also available across a wide range of devices and Operating Systems, ranging from small-scale dedicated solutions (e.g, KNX bridge [253], Libelium sensor nodes [25]) to large-scale systems (e.g., SaaS, PaaS, IaaS).

Hypertext Transfer Protocol (HTTP)

HTTP is an application protocol for distributed, collaborative, and hypermedia information systems [122] and is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext. Its development was initiated by Tim Berners-Lee at CERN in 1989. Standards development of HTTP was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), while the first definition of HTTP/1.1, the version of HTTP in common use, occurred in RFC 2068 in 1997, with two revisions at 1999 (RFC 2616) and 2014 (RFC 7230). Since 2015 a major revision, named HTTP/2, has been standardized and is now supported by major web servers, which primarily aims to improve performance by revising how the data is framed and transported between the client and the server.

In general, HTTP functions as a request-response protocol in the client-server computing model [181]. The client submits an HTTP request message to the server, which provides resources (e.g, HyperText Markup Language (HTML) files, images, array of bytes), or other content, or performs some functions on behalf of the client and returns a response message. The response contains completion status information [122, 271] about the request and may also contain requested content in its message body. HTTP defines methods (i.e., verbs) to indicate the desired action to be performed on HTTP resources, which are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifier (URI) schemes http:// and https://. Finally, HTTP uses a closed vocabulary of verbs (Table 4.1) with well-established semantics, which clients use to specify how they prefer to communicate with the server. A server is not obliged to "implement" all verbs; on the contrary it can discard requests towards methods that are not supported (e.g., a DELETE request to a server which maintains logs that are supposed to be read-only).

Table 4.1: Common HTTP 1.1 verbs and their semantics.

Method	**Idempotent**	**Description**
OPTIONS	Yes	Request for information about communication options
GET	Yes	Retrieves representation for a URI
HEAD	Yes	Retrieves meta-information for a URI
PUT	Yes	Creates or replaces a resource with a representation
POST	No	Augments an existing resource with a representation
DELETE	Yes	Deletes a resource that the URI specifies

Representational State Transfer (REST)

REST or RESTful web services is a way of providing interoperability between computer systems on the Internet. The term was introduced and defined in 2000 by Roy Fielding [123] in order to design HTTP 1.1 and Uniform Resource Identifiers (URIs). His objective was to explain how a well-designed Web application behaves when it consists of a network of Web resources (a virtual state-machine) where the user progresses through the application by selecting links (e.g, /products/electronics/computers/iMac), and operations such as GET or DELETE (state transitions), resulting in the next resource (representing the next state of the application) being transferred to the user for their use. REST-compliant Web services allow requesting systems to access and manipulate "web resources" (i.e., every thing or entity that can be identified, named, addressed or handled, in any way whatsoever, on the Web as depicted in Figure 4.2) using a uniform and predefined set of stateless operations.

In a RESTful Web service, requests made to a resource's URI will elicit a response that may be in XML, HTML, JSON or any other defined format as depicted in Figure 4.3b. The response may confirm that some alteration has been made to the stored resource, and it may provide hypertext links to other related resources or collections of resources. Using HTTP, as is most common, the kind of operations available include those predefined by the HTTP methods GET, POST, PUT, DELETE and so on (Figure 4.3a). By using a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that

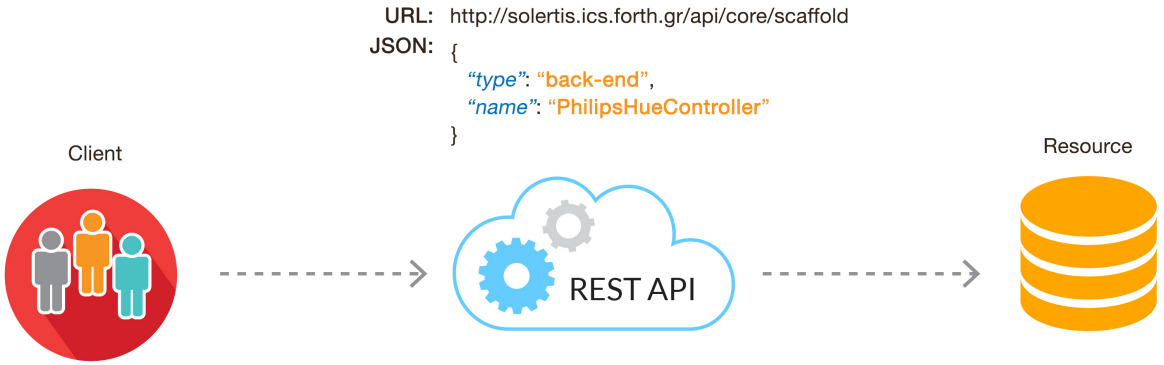

Figure 4.2: RESTful web services provide interoperability between computer systems on the Internet.

can be managed and updated without affecting the system as a whole, even while it is running. RESTful systems adhere to six guiding constraints (summarized in Table 4.2) to satisfy the following non-functional requirements: (i) performance, (ii) scalability, (iii) simplicity, (iv) modifiability, (v) visibility, (vi) portability, and (vii) reliability.

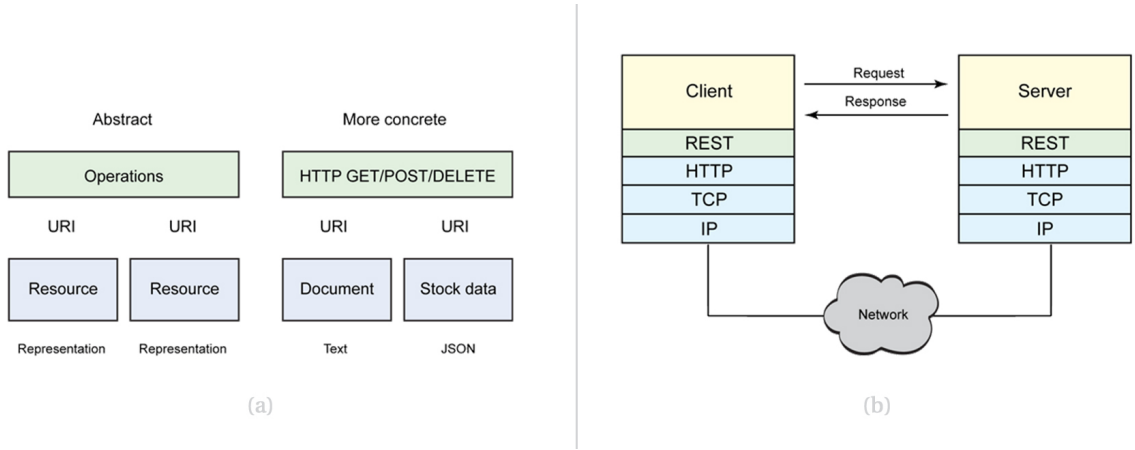

Figure 4.3: The Hypertext Transfer Protocol Protocol as explained in [372].

Synchronous, Asynchronous and Event-based Service Interaction

Every Von Nuemann computer [387] should incorporate, along with the processing and memory unit, an input device through which data will be received and an output device to communicate the computational outcome to the environment. This fundamental principle empowers computer

Table 4.2: Architectural constraints of REST.

Client-Server architecture	Separation of concerns (e.g., UI from data storage), Simplification and Independence of the server components.
Statelessness	No client context being stored on the server between requests.
Cacheability	Caching can eliminate some client-server interactions, further improving scalability and performance.
Layering	A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediaries may improve system scalability, enable load balancing, provide shared caches and enforce security policies.
Code on demand	Servers can temporarily extend or customize the functionality of a client by transferring executable code (e.g., Java Applets, JavaScript)
Uniform interface	Fundamental to the design of any REST service. It simplifies and decouples the architecture, thus each part can evolve independently by enforcing (i) resource identification in requests, (ii) resource manipulation through representations, (iii) self-descriptive messages, and (iv) Hypermedia as the Engine of Application State (HATEOAS).

programs to cooperate, by exchanging data, towards accomplishing their business goals. AmI artifact and scripts are no exception to that rule; AmI-Solertis provides appropriate mechanisms through which components could both make outgoing calls (i.e., emit events) to and receive incoming calls (i.e., synchronous or asynchronous /acHTTP requests) from other components.

By definition REST promotes the reception of incoming calls by advertising the functionality of a computer program in the form of remote procedure calls that others use. Remote procedure calls are usually orders of magnitude slower and less reliable than local calls, since as the server is processing the call, the client is blocked (i.e., it waits until the server has finished processing before resuming execution) as depicted in Figure 4.4. Nevertheless, programs instead of wasting valuable computational resources, have the ability to bypass blocking calls by applying appropriate techniques. At a CPU level a variety of design methodologies (e.g., instructional-level parallelism,

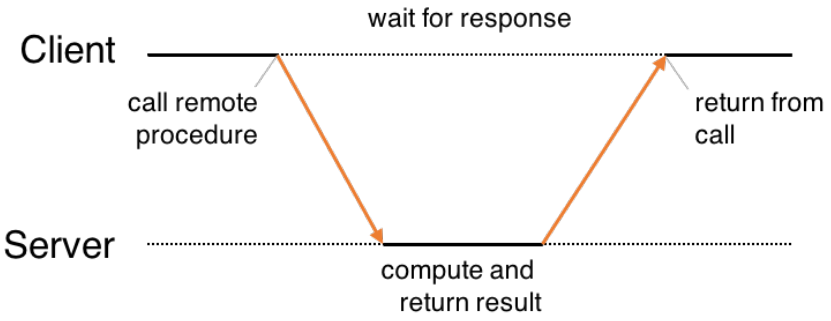

Figure 4.4: General synchronous (i.e., blocking) interaction between a client and a server (adapted from [366]).

task-level parallelism and data parallelism) cause the processor to behave less linearly and more in parallel [162], while at a higher level developers can use multiple threads to ensure that their programs will keep running independently of what happens in the background [365]. Parallelism improves the speed of a computing system and as a result delivers more computing power.

The rise of Internet permitted scientists to link thousand of computers all over the world to work together and force them address the blocking call problem at a macroscopic scale; in addition to resource waste, a solution for that problem was imperative from an Human Computer Interaction (HCI) perspective to prevent web browsers from "freezing" on the screen and becoming unresponsive, resulting into a negative user experience. To that end, the concept of asynchronous request was introduced via the XMLHttpRequest (XHTTP) object, which is an interface exposed by a scripting engine that allows scripts to perform HTTP client functionality, such as submitting form data or loading data from a remote Web site [380]. Web developers quickly took advantage of that technology and by combining it with Asynchronous JavaScript And XML (AJAX) [137] turned it into the de-facto standard when consuming web services through a web browser. An AJAX application eliminates the start-stop-start-stop nature of interaction on the Web by introducing an intermediary - an AJAX engine - between the user and the server. It seems like adding a layer to the application would make it less responsive, but the opposite is true; when an application uses an asynchronous XMLHttpRequest it receives a callback when the data has been received, thus letting the browser continue to work as normal while the request is being handled (Figure 4.5).

An AJAX request however implies that the user asked or the program explicitly needed to ei-

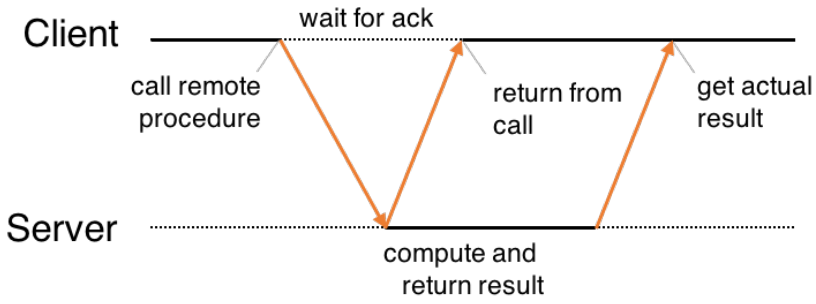

Figure 4.5: General asynchronous (i.e., non-blocking) interaction between a client and a server (adapted from [366]).

ther *pull* additional data [190] or order the execution of a certain functionality from a remote application /service. For any application to be considered intelligent - particularly within AmI environments - apart from passively responding to explicit requests, it has to be able to react proactively by responding to certain stimuli that occurs **at any time** within its area of jurisdiction. This programming paradigm derives from the Publish/Subscribe software design pattern [135] and is know as event-driven programming [255]. It states that the flow of the program is determined by events, such as: user actions (mouse clicks, key presses), sensor outputs, or messages from other programs / threads. In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected (Figure 4.6). From a software architecture, publish-subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize published messages into classes without knowledge of which subscribers, if any, there may be. Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are.

Data structures

Since multiple heterogenous services have to be integrated in an AmI environment, a common data format was selected to facilitate data exchange in AmI-Solertis. JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format, which

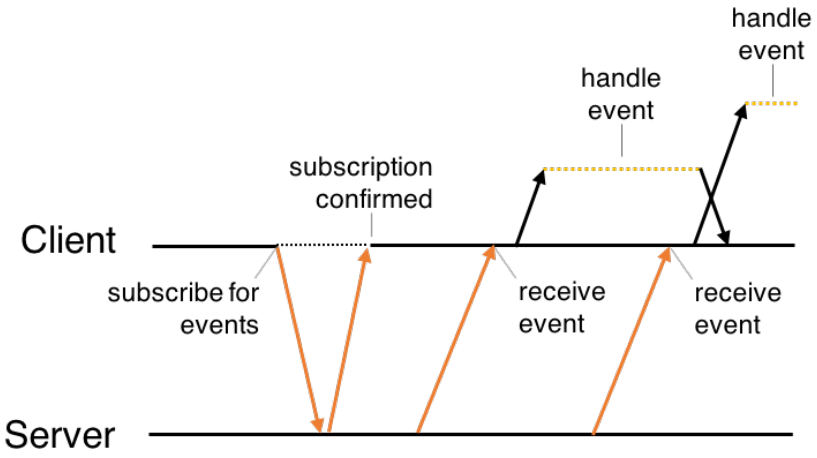

Figure 4.6: Event-based interaction (non-blocking) between a client and a server.

defines a small set of formatting rules for the portable serialization of structured data in a is a human-readable textual format. It is derived from the object literals of JavaScript as defined the ECMAScript Programming Language Standard [112], but does not attempt to impose its internal data representations on other programming languages. Instead, it shares a small subset of its textual representations with all other programming languages.

JSON can represent four primitive types: *strings*, *numbers*, *booleans*, and **null**, and two structured types: *objects* and *arrays* [88, 113] (as depicted in Figure 4.7). However, it is agnostic about *numbers* and offers only the representation of numbers that humans use: a sequence of digits. All programming languages know how to make sense of digit sequences even if they disagree on internal representations, which is enough to allow interchange (even if in any languages there might be a variety of number types of various capacities and complements, fixed or floating, binary or decimal). A *string* is a sequence of zero or more Unicode characters [80].

Programming languages vary widely on whether they support *objects*, and if so, what characteristics and constraints objects offer. Models of object systems can be widely divergent and are continuing to evolve. JSON instead provides a simple notation for expressing collections, where an *object* is an unordered collection of zero or more name/value pairs; a name is a string and a value is a string, number, boolean, null, object, or array. Most programming languages will have some feature for representing such collections, which can go by names like record, struct, dict,

map, hash, or object. JSON also provides support for *arrays* (i.e., ordered lists of values). All programming languages will have some feature for representing such lists, which can go by names like array, vector, or list. Because objects and arrays can nest, trees and other complex data structures can be represented. By accepting JSON's simple convention, complex data structures can be easily interchanged between incompatible programming languages. Finally, JSON does not support cyclic graphs, at least not directly, and is not indicated for applications requiring binary data.

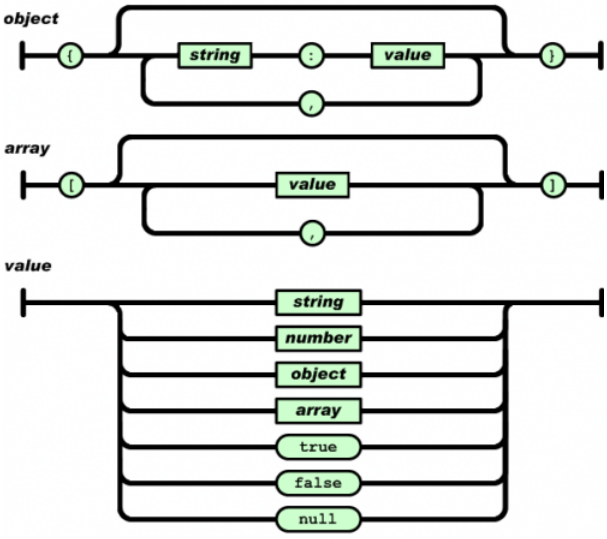

Figure 4.7: The JSON grammar as defined in [112].

JSON's design goals were for it to be minimal, portable, textual, and a subset of JavaScript. Because it is so simple, it is not expected that its grammar will ever change, which gives JSON tremendous stability, as a foundational notation. Those qualities and attributes established JSON as a very common data format used for asynchronous browser-server communication, including as a replacement for XML in some AJAX-style systems [137].

Finally, JSON is not the only option regarding data exchange across computer applications (including web and AmI services), since various text- and binary- based alternatives exist, such as: eXtensible Markup Language (XML), Yet Another Modelling Language (YAML) [43], Apache Thrift [302] and Google Protocol Buffers ProtoBuf [381]. Nevertheless, relevant experiments have shown that even if binary formats are faster and smaller than the text-based formats, they are not

as adaptable since data sent in a binary format cannot be parsed unless the receiver has both the relevant description files and the necessary means (i.e., software libraries) to understand them [360]. With respect to XML [224, 272] and YAML [117] data format, JSON manages to avoid the negatives of XML while maintaining many of its advantages (i.e., easy to use, supports a wide variable of applications, human-legible, easy to create). Therefore, JSON constitutes a lightweight data carrier with: (i) small space occupancy, (ii) fast transmission speed, (iii) human readable, (iv) a large user base, and (v) is used in a large number of web services, which gives it a unique advantage in becoming the dominant option. Thus, AmI-Solertis employs it as the main data interchange format between AmI artifacts and AmI scripts.

The AmI-Solertis paradigm

Reflecting on the benefits of asynchronous and event-based communication, AmI-Solertis introduces a unified Hybrid Communication protocol that combines the widely-used REST protocol with asynchronous and event-based communication facilities to integrate heterogenous services in a standardized -yet agnostic- manner (Figure 4.8). Therefore, an AmI artifact or an AmI script on the one hand exposes a REST interface to receive incoming calls, and on the other hand communicates its intention to the AmI ecosystem by emitting appropriate events. Since REST does not accommodate a standardized event mechanism by design, AmI-Solertis, following the widely-used and well-established practice of intermediary message brokers [169, 251], introduces a custom universal Event Federator server - that is built on top of the Redis [69] Pub/Sub mechanism - to enable asynchronous communication amongst the various artifacts. Therefore, publishers post messages to the federator and subscribers register their interest with that federator. The federator performs the necessary messages filtering and routing from publishers to subscribers, while it has the ability to prioritize messages in a queue before routing.

Writing asynchronous or event-based code though is a rather complicated task, especially in Javascript [275] (where it is often referred to as callback-hell). To address such shortcoming, AmI-Solertis employs the Promises API [285] provided by JavaScript as a well-documented and widely-used solution. Promises are objects that are used to execute an asynchronous operation, and invoke a “success” callback method with the returning value when an asynchronous operation completes successfully, or a “failure” callback method in case of an error.

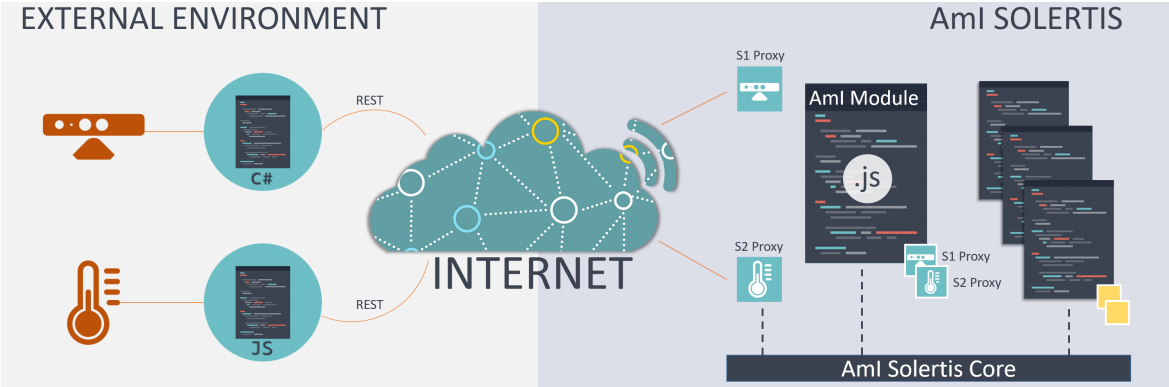

Figure 4.8: The AmI-Solertis Hybrid Communication Protocol

Finally, in order to assist developers using its communication protocol, AmI-Solertis delivers a set of tools and libraries (i.e., artifact proxies) that streamline remote service usage and encapsulate communication-related boilerplate code. The Promises API is integrated at a proxy level, thus every function that makes a remote call is "promisified" and returns a promise object. Masking remote calls to resemble local methods, along with the fact that promises support operation chaining, allows the definition of asynchronous AmI scripts that are written in a programming style close to its hypothetical synchronous equivalent (Listing 4.1).

4.2.2 Formalizing Functionality Specification

AmI environments rely on the interoperability of many heterogenous services and components in order to be functional. The literature review presented in chapter 2 suggests that some kind of formal service API specification (e.g., WSDL, Domain Specific Language (DSL)) is imperative to ensure efficient integration within such distributed environments. AmI-Solertis relies on the formal service definition to automatically generate service-specific wrappers that expose the underlying remote API as local functions, by encapsulating the communication protocol details, in order to simplify the overall programming process.

OpenAPI Specification (OAS)

REST enables developers to quickly create and distribute remote services that can be reused by others. But despite its popularity, it does not inherently support, nor an universally acceptable

method exists, permitting the formal specification of a REST service. Nonetheless, AmI-Solertis relies on the widely-used and well-defined OpenAPI Specification (OAS) to transparently unify the various service definitions under a common umbrella and allow the various services and systems to expose their APIs in a standardized manner. OAS is a powerful definition format to describe RESTful APIs. The specification creates a RESTful interface for easily developing and consuming an API by effectively mapping all the resources and operations associated with it. It is easy-to-learn, language agnostic, and both human and machine readable (Figure 4.9).

In the context of this work, the OAS schema is used to describe the available operations; in particular for every operation the following fields are necessary: (i) **Paths and Path Item Objects** to the individual endpoints, (ii) **Operation Objects** to provide details about the remote operations in those paths (e.g., summary, description, parameters, responses), (iii) **Parameter Objects** to describe in detail the expected parameters , (iv) **Response Objects** to outline expected responses of the operations , and (v) **Schema Objects** to allow the definition of custom service-specific input and output data types. Moreover, with the available example fields in the Parameter, Response, Schema Objects and the entire Example Object are used to document the values that an operation expects as parameters or might return as responses. Finally, since OAS does not inherently supports events, a custom micro-language has been devised to describe the events that a service might emit to the ecosystem via the Event Federator module. In particular, a path that contains the suffix *__AMI__EVENT__* signifies that this is not an actual operation that can be invoked, but is rather an event, which when emitted will distribute an piece of information structured as the response object describes.

Apparently OAS is an integral of the AmI-Solertis. All the artifacts that have been or will be integrated in the future in AmI-Solertis (including the its core services) expose their functionality as REST services, whose API is described in this specification. Actually, AmI-Solertis uses the formal OAS definition of every service in order to automatically generate service-oriented proxies, which aim to facilitate developers with remotely consuming the operations that it exposes in their programs.

Defining an AmI-Solertis compliant artifact

One key AmI-Solertis requirement is that it should enable: (i) fast, easy and error-free integra-

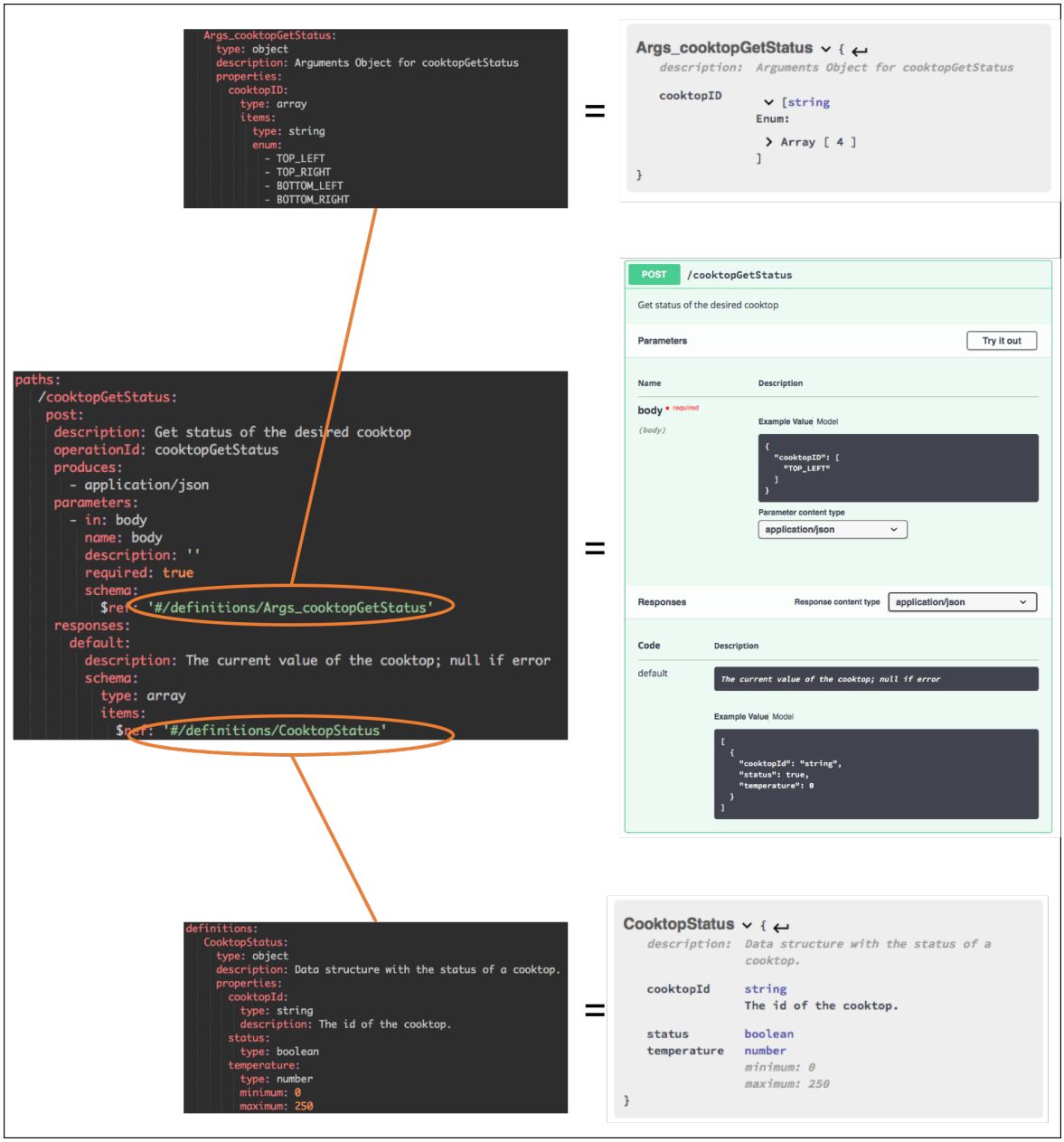

Figure 4.9: Part of the OAS specification of a Smart Kitchen artifact.

tion of external AmI artifacts (i.e., services) independently of their kind (e.g., pure web services, GUI applications that expose part of their functionality as a service following the Software-as-a-Service paradigm), and (ii) the creation of AmI scripts that make use of other AmI scripts or artifacts. Unfortunately, in practice, designing any kind of API is a rather complex and time con-

suming task; even if the functionality is available, its formal definition in a specification is still quite tricky. AmI-Solertis by design aims to address that shortcoming and transform the process of defining the formal specification of a service into an easy task.

AmI-Solertis introduces an automatic mechanism for generating the OAS specification of a service before installing it in the ecosystem. It relies on the fact that code documentation is considered as a good programming practice [111] and is being effectively used for automatic API documentation from source code comments [206] even for REST services [310]. In reality though, AmI-Solertis uses OAS as the intermediary towards automatic code generation of appropriate software wrappers that harmonize the overall programming approach independently of the service type and enable the employment of advanced software techniques such as metering [282], dynamic service adaptation [226], short-circuit [266], and metaprogramming [389]. To that end the OAS Bridge Creator tool was implemented, which accommodates the (semi-) automatic and manual installation of a new AmI artifact that provides its API in the following ways:

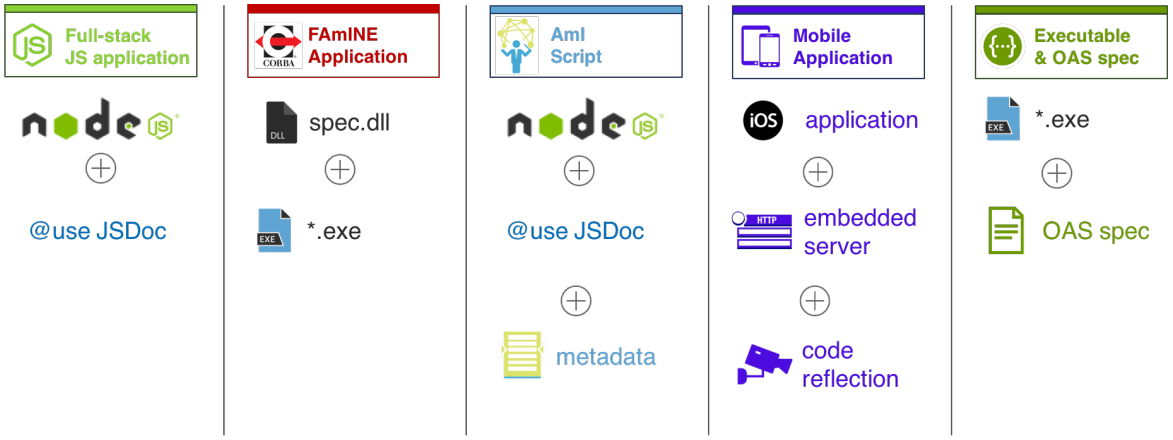

Figure 4.10: Alternative methods for defining an AmI-Solertis compliant artifact

- as **a set of REST endpoints written using JavaScript and ExpressJS** [238] that after static analysis generate a potential API that the service integrator has to validate before its final installation,
- in the **domain-specific Interface Definition Language (IDL)** that powers the FAmINE system [142], which is inherently supported and automatically generates the specification of any FAmINE-enabled service without requesting any user input

- as an **AmI script** that inherently exposes its functionality to the rest of the ecosystem
- as a mobile application that uses the **AmI-Solertis Mobile SDK for iOS**
- using **any 3rd-party technology** (e.g., C++, C#, Android Java), as long as the service implements the REST protocol and defines its API using the OAS specification.

Besides the last case where the developer has to manually provide the service specification, all the other cases are automatically handled by the AmI-Solertis system using a combination of the respective Authoring Tools (Chapter 6) and the back-end components that will be described in the next sections.

4.2.3 The AmI-Solertis Proxy

Invoking a local function in every major programming languages (e.g., Java, C#, C++ Javascript, Python) is just a single command; this is not the case for remote functions though. On the contrary, almost all of languages rely on additional libraries to handle the respective complexity (Table 4.3) and even then, making a "seemingly simple" function call requires a couple lines of code to properly configure the invocation (as presented in the figure below). AmI-Solertis encapsulates this complexity within service-specific proxies that expose functions that internally configure and perform the actual remote call.

Table 4.3: Indicative list of the available libraries to consume REST services.

Language	**Libraries**
C++	libCurl, neon, POCO, C++ Requests, QHttp, C++ REST SDK (codename "Casablanca")
Java	HttpUrlConnection, URLConnection, Apache HttpComponents, Google Java HttpClient
C#	HttpWebRequest, WebClient, HttpClient class, RestSharp NuGet package, ServiceStack Http Utils
Javascript	request, got, superagent, XMLHttpRequest, jQuery $.ajax, Axios
Python	httplib, requests, urllib

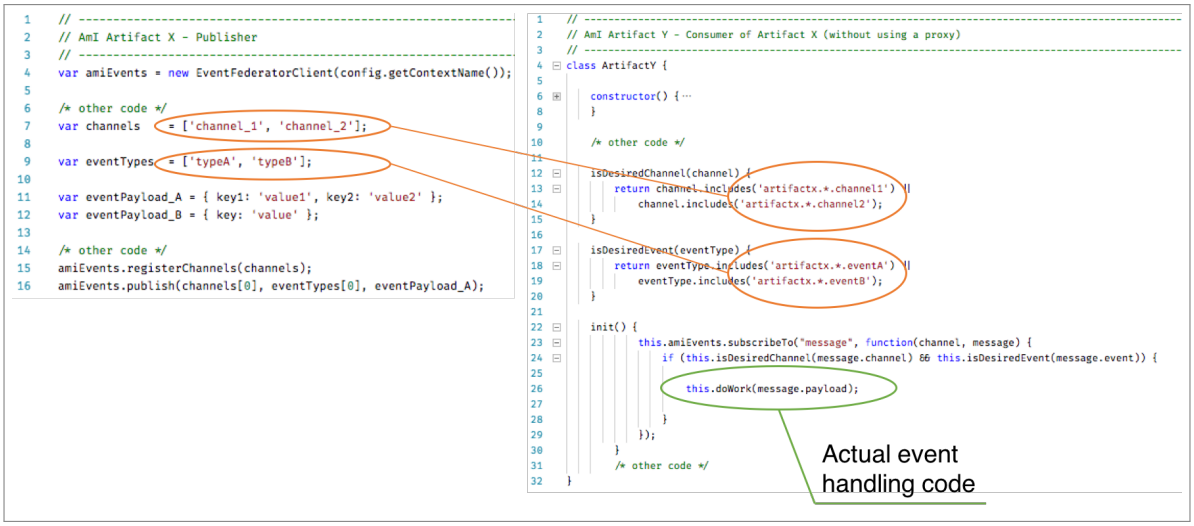

Figure 4.11: Without a proxy, an artifact requires low-level details to consume a service.

Therefore, developers use remote functionality as if it was offered by local components that reside within their program. In addition to code minimization, proxies also empower AmI-Solertis to dynamically adapt and adjust the invocation process (as depicted in Figure 4.12) in order to address emerging requirements such as re-routing a call to a replicated host to balance loading, immediately terminating the call if the remote endpoint is unavailable, intercepting a call and logging relevant QoS-related metrics, replacing the target endpoint with another that offers semantically similar functionality (e.g., use AccuWeather™[1] instead of YahooWeather™[2] web services).

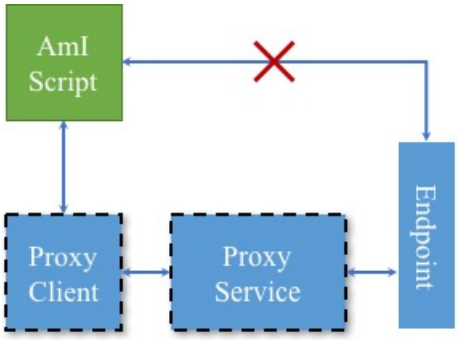

Figure 4.12: The AmI-Solertis proxy concept.

In addition to simplifying remote calls, proxies enable remote artifacts consumers to regis-

[1] See http://apidev.accuweather.com/developers/
[2] See https://developer.yahoo.com/weather/

ter their interest to events coming from those artifacts component without specifying any details about the underlying topology and infrastructure. In particular, publishers use the AmI-Solertis Event Federator to emit their events to named channels, whereas subscribers have to register to those channel by name to receive any relevant events. Usually, if this process is done manually, the developers of the artifacts that emit events have to advertise the channels that they use, so that interested consumers register to them (Figure 4.11). AmI-Solertis simplifies that process via the **Event Federator client**. It is a javascript module that artifact developers can use in order to publish their events to the main AmI-Solertis Event Federator without having to specify the exact channel, as this is automatically configured at runtime by the client with the guidance of the main federator. Likewise, an AmI-Solertis proxy internally packs an instance of that client, which is automatically configured to listen only to the respective channel(s) where the wrapped service is known to emit events. The event-related methods enable any interested parties to subscribe their callback methods to be called when an event is emitted by the specific component (Figure 4.13).

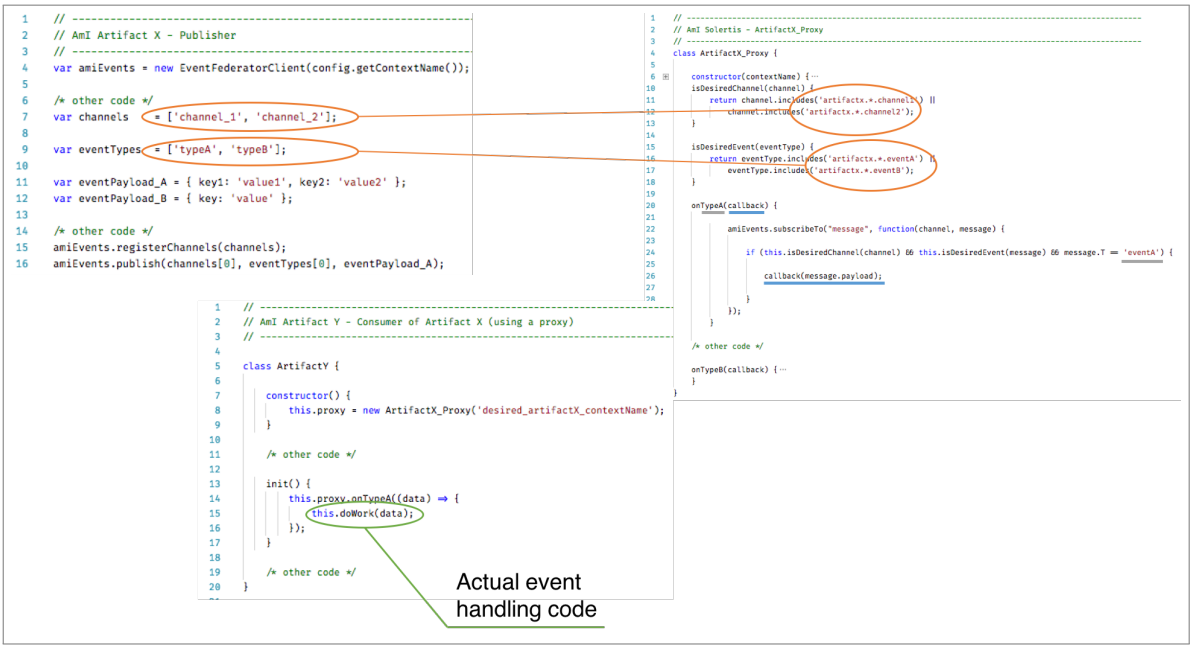

Figure 4.13: Using a proxy, an artifact only focuses on defining the handling logic.

4.2.4 Proxy Generation Process

AmI-Solertis offers five alternative methods through which an new AmI artifact can be incorporated in the AmI ecosystem (Figure 4.10). Towards harmonizing the overall programming approach independently of the type of integration method, AmI-Solertis employs a three-stage proxy generation process that ensures scalability and maintainability (Figure 4.14), namely: (i) methods and data types extraction, (ii) OpenAPI Specification creation, and (iii) code and configuration files generation. In particular, during the first stage the respective components are inspected in order to identify which are the operations and events that are intended to be exposed to the ecosystem and what data types do they use to specify their input and output parameters. Next, these information are encoded in a formal OpenAPI Specification that describes the new service, which is used in the third stage so as to generate the actual proxy. In the last step, in addition to the software wrapper that developers will onwards use to consume that service (i.e., artifactX_proxy.js), various configuration and definition files that are necessary for other AmI-Solertis components (e.g, Authoring tools, Deployment Manager, Executor) are automatically generated (i.e., artifactX_definition.json, artifactX_mock.js, artifactX_package.json, artifactX_visual_comp.json). As depicted in Figure 4.14, depending on the integration method, the overall process is loosely coupled as on the one hand method-specific components extract the OAS specification, and on the other hand the proxy generator only relies on the formal specification to dynamically generate the necessary elements. These intermediary components will be presented first, and then the section will conclude by describing the generation process per se.

CASE- 1: Full-stack Javascript application

Using Javascript on top of NodeJS™, developers are able to build full-scale applications including web-services and full-stack web applications [3]. Both cases are relevant to AmI-Solertis, as in the former, the integrated software delivers a self-contained service that the ecosystem can use, whereas with respect to the latter, AmI-Solertis is only concerned with applications that expose their functionality not only via a GUI to their end-users, but also expose an API in the form of REST endpoints that external services can invoke in order to modify and control the User Interface and the overall interaction with the user (e.g., navigate to a new screen or modify the displayed content

[3] A web-based application that most of its components (i.e., server and hosting environment, data modelling, business logic, API, GUI) are written in Javascript

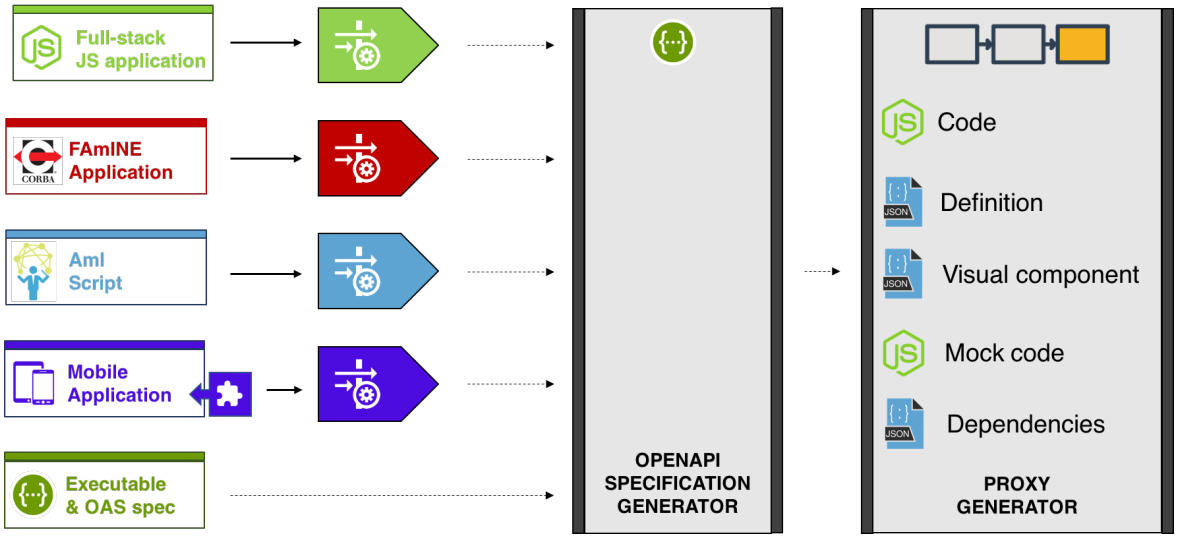

Figure 4.14: AmI-Solertis three-stage proxy generation process.

due to contextual stimuli or due to proactive decision of the intelligent environment).

Usually, when a full-stack web application is implemented using JS and NodeJS, the extremely popular Express™[4] application framework is used as the foundation layer towards building the desired functionality. Among others, Express facilitate developers in defining the basic routing, that is how the application will respond to a client request to a particular endpoint, which is a URI and a specific HTTP request method (GET, POST, and so on). Each route can have one or more handler functions, which are executed when the route is matched, whereas route definition takes the following structure:

Listing 4.2: A hello world Express application

```
var app = express();
// GET request
app.get('/hello', function (req, res) {
  res.send('Hello World (using HTTP-GET)');
});
// POST request
app.post('/hello', function (req, res) {
  res.send('Hello, ' + req.body.username + ' (using HTTP-POST to send data
      )');
});
```

[4]See https://expressjs.com

Listing 4.2 presents a simple web application that either responds with the generic “Hello World” message when an “anonymous” *GET* request is made to the URL: */hello*, or if the client sends a *POST* request that contains the name of the user, then responds with a personalized message “Hello, Asterios” (in case the provided username is “Asterios”). As already mentioned, Express permits multiple functions to be chained together to handle a single request; nevertheless, for compatibility purposes, it requires all of them to share the same signature (Listing 4.3) and expect two incoming parameters: the *req* and the *res* object; the req object encapsulates any incoming parameters send via the /acHTTP protocol, whereas the res object holds (among others) a connection with the client through which data can be streamed as a response to the request. Optionally, when a handler completes processing it can pass control to the next one via the *next()* method.

Listing 4.3: Generic Express REST Handler

```
function handler(req, res, next) {
  // incoming parameters are accessible via the req object
  // outgoing messages can be forwared via the res object
  // when done
  next();
}
```

Currently Express powers more that 250.000 web sites worldwide[5] and has been adopted by Fortune-500 and other well-known companies like: Microsof, IBM, Paypal, Accenture, Uber, Yandex, Fox Sports, etc, which confirms the wide acceptance and successful application of that practice within just 7 years from its initial release in 2010 [6].

AmI-Solertis, ackowledges the popularity of that practice and facilitates developers that choose it to quickly integrate their software in the AmI ecosystem. Since neither Javascript nor Express routing mechanism provide type information by design, AmI-Solertis relies on another extremely common and highly desirable programming practice, namely code documentation [111], to extract the needed information towards simplifying the OAS generation and eventually integrating the intelligent artifact into AmI-Solertis. In particular, it suggests developers to use Express to define their routing schemes and fully-document (using JSDoc [7]) the respective handler functions.

[5] See https://wappalyzer.com and http://expressjs.com/en/resources/companies-using-express.html
[6] Release 0.0.1: https://github.com/expressjs/express/blob/master/History.md#001–2010-01-03
[7] See http://usejsdoc.org/index.html

Listing 4.4: Definition of a simple REST endpoint (as it would appear in main express.js file).

```
1 var express = require('express');
2 var controller = require('./kitchen.controller');
3 var router = express.Router();
4 router.post('/cooktopGetStatus', controller.cooktopGetStatus);
```

An illustrative example of such a definition is presented in Listing 4.4, where a simple web service that communicates with a kitchen is defined. Specifically, the application exposes a single endpoint (i.e., remote function) at *"/cooktopGetStatus"*, through which the AmI environment can query the status of the cook top. Instead of directly defining the handler function as done in Listing 4.2, the example applies the delegation pattern of the Object-Oriented Programming (OOP) paradigm [258] and specifies that the handler is a method defined in the controller object - a practice that allows object composition to achieve the same code reuse as inheritance -. AmI-Solertis's Rest2Swagger component uses the Esprima parser [8] in order to both detect and extract endpoint information from its definition (line 4), and identify which method's documentation should it search to fully describe that remote operation. In this, example, Rest2Swagger detects that it should analyze the function *cooktopGetStatus* that belongs to the object *controller*, whose class definition will be found in the file *kitchen.controller.js* that resides in the same working directory.

JSDoc uses a Javadoc-like syntax to primarily document JavaScript code, but it has other more interesting uses as well; for example, developers can annotate using JSDoc their code so that compilers (such as Google's Closure Compiler [49]) could look for type information in JSDoc tags, optimize the code and check it for possible type errors and other mistakes [9]. AmI-Solertis applies a similar approach in order to formally specify a REST operation of interest in OAS. As long as the Rest2Swagger components knows which function to look for (and where to find them respectively), it parses their associated documentation in order to collect information regarding their objective and their parameters.

Given that every handler method takes two arguments only (i.e., req and res object) as specified by the Express framework, AmI-Solertis suggests developers to provide extended documen-

[8] See http://esprima.org: ECMAScript parsing infrastructure for multipurpose analysis

[9] See Annotating Javascipr for the Closure Compiler: https://github.com/google/closure-compiler/wiki/Annotating-JavaScript-for-the-Closure-Compiler

tation about actual values of the parameters that the remote caller will encapsulate and send in the body parameter of the request object (which is exactly what JSDoc recommends for composite parameters as well). For instance, in the previous example, as seen in line 15 of Listing 4.5, the developer specifies that the caller should append in the "POST" request's body a parameter named *cooktopId*, which should hold an array of strings that correspond to the identifiers of the cooktops for which information is needed. Upon successful execution, the operation will return an array of cookstatus objects; each item will contain the id of the cooktop it refers to, along with two variables: a boolean that will report its status and a numeric one that will contain its current temperature.

Listing 4.5: Documenting REST handler functions using JSDoc.

```
1  /**
2  * Data structure with the status of a cooktop.
3  * @typedef {Object} CooktopStatus
4  * @property {String} cooktopId - The id of the cooktop.
5  * @property {boolean} status - Indicates whether the cooktop is open.
6  * @property {Number} temperature - The current temperature => VALUES: [0
       ..250]
7  */

9  /**
10 * Get status of the desired cooktop
11 *
12 * @export
13 * @param {Object} req - http request object
14 * @param {Object} req.body - the request parameters
15 * @param {String[]} req.body.cooktopId - The cooktop id => VALUES: [
       TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, BOTTOM_RIGHT]
16 * @param {Object} res - http response object to report any issues
17 * @return {CooktopStatus[]} The current value of the cooktop; null if
       error
18 */
19 function cooktopGetStatus(req, res) {
20   // do work
21 }
```

As the example suggests, AmI-Solertis enables developers to encode detailed descriptions about the operations, parameters and data types they use, and also permits them to provide additional hints to the consumer of their services regarding the expected values of any incoming parameters or the potential values of any outgoing parameters. Specifically, in this example, a cook-

top identifier should take one of the following values *"TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, BOTTOM_RIGHT"*, while the current temperature of a cooktop will range between 0 and 250.

CASE- 2: FAmINE Application

FAmINE [142] is a custom CORBA-based middleware that enables two services to interoperate independently of their implementation language; it currently support Java, C#, C++, Actionscript and Python. FAmINE has introduced its own Interface Definition Language (IDL) through which developers can describe their service's operations, events and data types (as presented in Listing 4.6). The service definition usually consists of three discrete sections, namely:

- the *data types definition section*, where a developer can define any composite structures that the service will either send or receive data
- the *methods section*, where the methods that can be directly invoked are defined in detail (i.e., name, arguments, returning type)
- the *events section*, where the developer can designate what kind of events their service emits and what will these events will contain as a payload. In particular, methods that start with the prefix *Event_* define the various outgoing events and the type of their argument specify the type of the payload that the event will hold.

Listing 4.6: Definition of a SmartKitchen service in FAmINE.

```
interface SmartKitchen {
  // Service-specific data types
  struct CooktopStatus {
    string  id;
    boolean status;
    float   temperature;
  };
  typedef sequence<string> CooktopIdentifiers;
  typedef sequence<CooktopStatus> CooktopStatuses;
  // Methods
  CooktopStatuses GetCooktopStatus (in CooktopIdentifiers cooktopIDs);
  // Events
  void Event_CookingDone (in string cooktopId);
};
```

As soon as a service definition is available, a developer through a custom tool can generate a service-specific library (e.g., a DLL file) that encapsulates the static definition of that service. Afterwards, they can implement or use that service using any of the supported languages.

A developer who wants to either publish (i.e., implement) or consume (i.e., use) a FAmINE service, has to use its static definition properly. FAmINE forces the correct (i.e., typed) usage of services, which on the one hand simplifies implementation and on the other hand facilitates the use of CORBA's interoperation facilities. Thus, for every FAmINE-enabled service, it generates a *language-dependent abstract definition* (i.e., abstract class, interface) that contains only the signatures of methods along with auxiliary data type definitions.

Listing 4.7: Publisher of the SmartKitchen service (written in C#).

```
1 class Main
2 {
3   public void int Main() {
4     SmartKitchenImplementor kithen = new SmartKitchenImplementor();
5     Ami.Famine.Register<SmartKitchen>(kitchen, "mainKitchen");
6   }
7 }
```

Consequently, (i) a class that publishes that service must implement (at compile time) that abstract definition (i.e., extend the abstract class, implement the interface) and use it as a reference when registering it to the broker (Listing 4.7), while (ii) a class the uses that service must declare a variable of that type (at compile time) , which will hold the reference to the running service after its resolution from the broker (Listing 4.8). Therefore, it is guaranteed that both the publisher and the consumer at run-time will share the same service definition and one-way interoperation will be achieved.

Listing 4.8: Consumer of the SmartKitchen service (written in C#).

```
1 class Main
2 {
3   public void int Main() {
4     SmartKitchen kithen = Ami.Famine.Resolve<SmartKitchen>("mainKitchen");
5     kitchen.GetCooktopStatus(new String[] {"TOP_LEFT"});
6   }
7 }
```

FAmINE is being extensively used in the context of the Ambient Intelligence Programme[10] of FORTH-ICS[11], therefore many services and applications have been developed with it. AmI-Solertis introduces an automatic method through which existing services and applications that expose their functionality through FAmINE can be easily integrated in its AmI ecosystem. In particular, AmI-Solertis, through the IDL2Swagger component, can automatically (i) extract the formal specification of a FAmINE service from the respective definition file - currently it uses the DLL file that FAmINE generates when it is supposed to be used by C# (Listing 4.9) - so as to create its OpenAPI Specification, and (ii) generate a wrapper that exposes the same service via the REST protocol (see section 4.3.5).

Listing 4.9: Implementation of the SmartKitchen using C#.

```
class SmartKitchenImplementor : AmIHome.SmartKitchen
{
  struct CooktopStatus
  {
    public string id;
    public bool status;
    public float temperature;
    public CooktopStatus(string id, bool status, float temperature);
  }
  public void Event_CookingDone(string cooktopId) { /* emit event */ }
  public CooktopStatus[] GetCooktopStatus(string[] cooktopIDs) { /* do */
      }
}
```

IDL2Swagger relies on the powerful reflection capabilities of the Microsoft .NET Common Language Runtime (CLR) in order to inspect the FAmINE generated CLR Assembly and detect the methods that it offers, the events that it emits and the data types that it uses. Upon completion, all the necessary information have been collected and the next stage of the pipeline can be activated (i.e., OAS generation).

CASE- 3: AmI script

An AmI script is software that developers can manipulate only via the built-in authoring tools of AmI-Solertis. Behind the scenes though, AmI-Solertis automatically generates a web-based JS application where it injects developers' code to the appropriate places or creates entirely new

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

[11] Foundation for Research and Technology - Hellas, Institute of Computer Science, www.ics.forth.gr

files when needed. This process is totally transparent to the developer. Conceptually, it constitutes a future-proof feature of the AmI-Solertis framework, as having its own projects compatible with commonly used approaches, in the future the system could extend its offered facilities and support advanced programming schemes (e.g. projects, web-based applications spanning across hunderds of files). Realistically though, AmI-Solertis applies such an approach to re-use existing practices (i.e., CASE-1).

Listing 4.10: Exporting AmI script operations by appropriate annotating in their JSDoc.

```
class KitchenController {
  /**
  * Turn on a cooktop
  *
  * @export
  * @param {String} cooktopId - The cooktop id => VALUES: [LEFT, MIDDLE,
      RIGHT]
  * @return {Number} The new temperature of the cooktop
  */
  async turnOnCooktop(cooktopId, temperature) {
    await this.setTemperatureViaKNX(cooktopId, temperature)
  }

  /**
  * Turn off all cooktops
  *
  * @export - /shutdown
  * @return {Boolean[]} The new status of all cooktops
  */
  async turnOffAllCooktops() {
    await this.sendTurnOffSignalViaKNX();
  }

  setTemperatureViaKNX(id, value) {
    // do work
  }

  sendTurnOffSignalViaKNX() {
    // do work
  }
}
```

Nevertheless, the programming process is slightly different. In CASE-1, developers are fully aware about the Express framework, therefore they have to correctly setup their routing scheme on their own. AmI scripts are supposed to simplify the overall process, therefore developers are

only required to annotate which functions they want to expose to the ecosystem and optionally define the URL for each; then AmI-Solertis automatically combines the function name along with the script name in order to generate a random endpoint URL. Listing 4.10 provides such an example. The overall AmI script has four functions, namely: *turnOnCooktop*, *turnOffAllCooktops*, *setTemperatureViaKNX*, *sendTurnOffSignalViaKNX*, out of which the developer has selected to expose only *turnOnCooktop* and *turnOffAllCooktops*.

Given, that these services will become available for others, AmI-Solertis has automatically generated the placeholder JSDoc that the developer could fill-in as deemed necessary. The key parameter is the JSDoc Tag **@Export**, which signifies that this operation should be exposed via REST. In this example, the endpoint for the first function will be randomly generated, whereas for the second function the endpoint will be a combination of the AmI script's name with the constant string "/shutdown" that the developer has explicitly set. After determining which functions need to be exposed (by inspecting the available documentation), AmI-Solertis executes a process similar to CASE-1 and delegates control (and the extracted data) to the Proxy Generator module.

CASE- 4: Mobile Application (iOS)

Applications targetting smartphones and tablets are also potential (and desirable) candidates for integration in the AmI ecosystem due to their inherently pervasive nature. Moreover, given that their popularity keeps on increasing, AmI environments definitely need to use them to benefit from their: (i) processing power [159], (ii) networking [17,18,291], (iii) sensing [141,363] and (iv) interface [36] capabilities. Despite their omnipresence, smartphones are mainly characterized by opportunistic interaction [397], as task switching happens quite frequently [188, 218].

Their ephemeral nature poses some interesting challenges from an engineering perspective as well. Services deployed on stationary hosts are considered to be always available and even in case of failures, a variety of approaches (e.g., replication, redundancy) enables quick and effortless recovery. The situation is radically different for mobile hosts; if a host becomes unavailable, nothing guarantees that the host will become available again. Nonetheless, the AmI environment should be able to accommodate AmI artifacts that are deployed on ephemeral hosts including mobile devices, to eventually benefit users who regularly visit the intelligent environment.

For that to be achieved, this work introduces a the AmI-Solertis Mobile SDK for iOS™, which

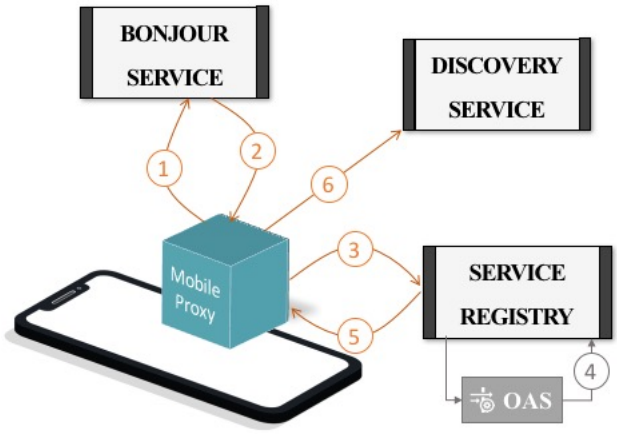

Figure 4.15: Bonjour protocol is used to assist negotation process

aims to assist developers on building AmI applications that offer opportunistic interaction with the intelligent facilities of an AmI environment.In more detail, the industry standard Bonjour™zero-configuration networking software [182] is used to dynamically discover and install mobile services offered by smartphone applications. To that end, a iOS application, via the Mobile SDK, uses Bonjour to advertise and facilitate the discovery of its functionality from the AmI ecosystem; the respective server-side Bonjour components monitor the environment for such signals and upon receipt they automatically install the service and initiate the generation of the appropriate proxies (Figure 4.15). Moreover, the Mobile SDK internally implements a micro HTTP server that receives requests and a Redis client that emits event as every other AmI-Solertis compliant artifact does, hence AmI scripts can use such applications as if they were regular services deployed on stationary hosts.

Since simplicity is a key aspect of the AmI-Solertis framework, the SDK encapsulates the necessary complexity (i.e., server configuration and launching, service advertising through Zeroconf, data transformation from native data types to JSON and backwards, event triggering) and simply requires from the developers to: (i) define the data types that the application will be sending or accepting, (ii) implement the functions that they want to expose over REST and connect them with their local delegates, (iii) declare the endpoints that remote clients could use to contact them, and (iv) specify outgoing events (Figure 4.16). Aftewards, the AmI-Solertis Mobile SDK collects these information (some at compile-time and others at runtime using reflection) and advertises

the service's availability via Bonjour. On a mobile device entering an AmI environment, the SDK publishes to AmI-Solertis all the details about its functionality (i.e., operations, events, data types) and eventually gets integrated as an AmI artifact. From that point onward, other AmI scripts can use it via its proxy, without even knowing that this is an ephemeral service offered by a mobile host.

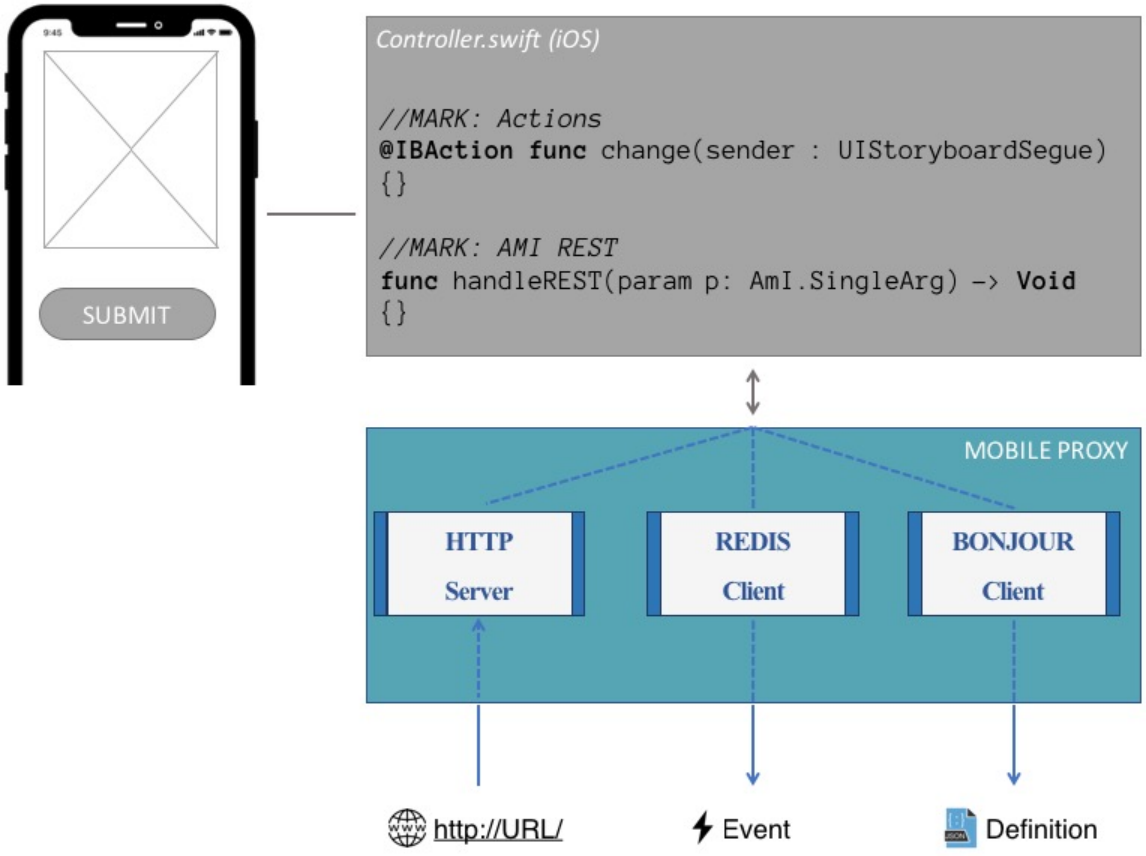

Figure 4.16: The code that a developer has to write to offer the functionality of its mobile app as a service.

CASE- 5: Third-party applications and services

Finally, with respect to any other 3^{rd}-party applications and services that need to be integrated to AmI-Solertis, but they have been implemented following different approaches (e.g., dynamic definition of the routing scheme) or using other technologies (e.g., Django, Ruby on Rails), AmI-Solertis offers the entire manual process where the developer/integrator has to specify the OAS

by hand and explicitly provide it. In such cases, AmI-Solertis simply validates whether the specification file is syntactically correct and contains the required information that will guide the proxy generation process, and notifies the developer accordingly.

Code Generator

As soon as the necessary information are gathered from the various intermediary components and the respective OpenAPI Specification is created, the same code and files generation process starts so as to create the following five items: (i) the **proxy class** that developers will use in their code, (ii) the **type definition file** of the proxy class that facilitates the authoring tools in providing code completion and type-checking facilities (iii) the **visual component definition file** that guides proxy's integration into the visual editor toolkit library, (iv) a **basic mock class** that virtualizes the service and can be used by the AmiTest framework [230], and (v) the **package.json** file that documents the run-time dependecies that the proxy class requires to execute properly.

Code generation is based on logic-less templates[12] [391] and on the Handlebars library[13] [236] that let AmI-Solertis build semantic templates effectively. In particular, AmI-Solertis has a set of files that contain both predefined code or values and some text placeholders (i.e., tags) that get "expanded" (i.e., replaced) by the template engine using values provided in a hash or object, which gets populated by the information available on the OAS of every installed service (as depicted in Figure 4.16 below).

In principle code generation is a streamlined process, however a few interesting aspects are worth mentioning with respect to the generated proxy class. First, an AmI-Solertis proxy aims to encapsulate the difficulties of asynchronous programming by masking remote services and make them to appear as if they were locals. For that to be achieved, the Swagger2Proxy component creates a Facade object [135] that integrates the Promises API at a proxy level by "promisifying" every function call and making it return a promise instead, which can be used by the caller to monitor the progress of the call. Second, when an artifact defines that it emits events via the Event Federatior, the Swagger2Proxy injects a series of event-related functions that developers can use in order to declare their interest and at the same time pass a callback function that will

[12] The term "logic-less" means that there are no if statements, else clauses, or for loops. Instead there are only tags, some of which tags are replaced with a value, some nothing, and others a series of values.

[13] See http://handlebarsjs.com

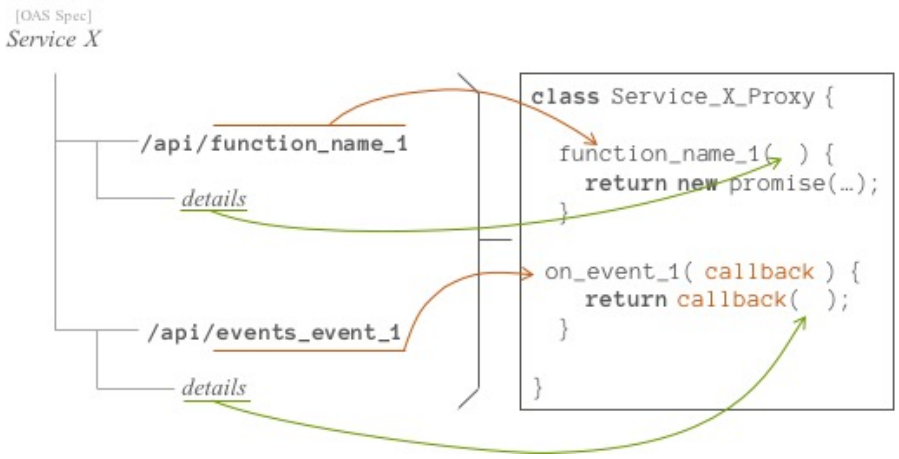

Figure 4.17: The translation process

be invoked when they are emitted. Finally, if necessary, Swagger2Proxy also generates auxiliary software components (i.e., bridges) to accompany and be executed alongside with the original service in order to transform as needed. For example, in case of a FAmINE-based service that relies on CORBA to facilitate the interprocess communication, AmI-Solertis generates a Javascript web application that maps and exposes those non-REST, remote CORBA methods to REST endpoints, thus making them widely accessible.

4.2.5 The AmI-Solertis Registry Service

AmI-Solertis introduces its own artifacts' and scripts' registry that holds a variery of information (including metadata) for all the components that have been integrated into the ecosystem. Since, AmI-Solertis also empowers developers to deploy and execute those intelligent components on the available AmI-Solertis compliant hosts, it also incorporates multiple repositories that store the actual executable software that implement these services; that software can either be executable binary files or distributable source code that runs on top of the an auxialiary runtime environment (e.g., NodeJS, Python, JRE). In particular, the registry is internally composed by the following components:

- a service registry to store basic information about the registered artifacts and scripts (including their formal OAS file),

- a metadata storehouse to hold additional information about these (e.g., type, category, tags, dependencies)
- an AmI artifacts repository that contains all the external components independently of their kind (i.e., binaries and distributable source code)
- an AmI scripts repository that stores all the scripts that have been developed using the built-in authoring tools
- a registry of AmI-oriented modules that accommodate all the packages of reusable code that could be shared across artifacts, scripts or external systems.

The **Services and Metadata registry**, supports the four basic functions of persistent storage, namely Create-Retrieve-Update-Delete (CRUD), so that AmI-Solertis could effectively incorporate new and manage existing content using the available REST API. These two registries do not distinguish the integrated components into artifacts or scripts; they rather unify them under basic (yet modifiable) categories. Moreover, additional tags can be assinged to them to facilitate browsing or searching and empower developers to create their own informal, arbitrary, classification schemes, which can be useful both to them and to the larger community of the AmI-Solertis users. Finally, through tags semantic information coming from external sources (i.e., ontologies) can be attached to services, thus enabling dynamic service mapping and semantic reasoning (e.g., equivalence, (in)compatibility) when necessary.

With respect to the **AmI Components Repositories (artifacts or scripts)**, they are both build on top of the Git version control system [14] [228, 373] so as to keep track of different versions of the software they contain. Particularly, the AmI artifacts repository, retains only the stable versions that still appear to be in use, thus balancing a minimal storage footprint with maximized compatibility. For instance, if an AmI script A depends on a version X of AmI artifact B, and artifact B has been updated to version Y (where $X < Y$), version X will remain available until script A gets updated to use the newer version of artifact B. On the contrary, regarding the AmI scripts Git repository, the full version tree is retained, as every change is recorded over time, and developers can recall specific versions later. This fact not only permits developers to keep track of their

[14] See https://git-scm.com

script builds by being able to identify which version is currently in development or production, but mainly relieves them from having to use external versioning systems to securely store their work, as this functionality is inherently offered by the AmI-Solertis authoring tools. Moreover, AmI-Solertis conveniently utilizes the distributed nature of Git and its speed in order to easily deploy the various components on the AmI-Solertis compliant hosts (see section 4.3)

Finally, the **registry of AmI-oriented modules** accommodates all the packages of reusable code that are automatically generated when a new AmI artifact or AmI script gets incorporated in the system (i.e., proxies that simplify their use). Given that, from an engineering perspective, a proxy is just a standard Javascript class that exports a set of functions to be used by others, AmI-Solertis distributes them as NPM packages that can be easily shared across artifacts, scripts or external systems written in Javascript, while the Node Package Manager (NPM) system automatically handles their installation. In terms of implementation, AmI-Solertis uses Verdaccio [15] to build and maintain its own local private NPM registry with zero configuration. This approach empowers AmI-Solertis to privately store and classify any automatically generated proxies. At the same time, with zero additional cost, AmI-Solertis also permits developers to access via its user interface the npmjs.org registry in order to use any external libraries that they wish to use in their program. Therefore developers can combine their preferred public modules with private proxies so as to efficiently assemble their code. AmI-Solertis categorizes proxies under the following three labels based on their expected frequency of use (from the least to the most popular):

- **@amisolertis/core-***: any software bridge that provides "controlled" access to core AmI-Solertis components and services (e.g., Registry, Executor)
- **@amisolertis/utils-***: auxialiary libraries that are considered to be part of the AmI-Solertis Standard Library (e.g., Event Federator Client, Datastore Client)
- **@amisolertis/proxies-***: components that automatically generated based on the OAS specification of an AmI artifact or script and facilitate its use.
- **@amisolertis/generator-***: the project scaffolding components.
- **@amisolertis/various-***: general purpose components.

[15] See https://github.com/verdaccio/verdaccio

Components Directory Service

AmI-Solertis targets highly mutable and dynamic environments that change quite rapidly. Therefore, references to any of its components (i.e., AmI artifacts and scripts) should minimize (or even eliminate) the amount of static information they require to discover and communicate with each other; on the contrary, the ecosystem should offer a dynamic and flexible naming system that promotes ubiquitous and distributed access to any requested resource or service [168]. To that end, AmI-Solertis employs an enhanced Domain Name System (DNS) tool, named Components Directory Service, that associates various information with the domain names assigned to each of the participating entities and serves as an advanced "phone book" for the ecosystem. In principle, Components Directory Service is used by AmI-Solertis proxies in order to optimally resolve [274] the necessary connection information and communicate with the running instance of the AmI artifacts or AmI scripts that they represent.

Every entry in the Components Directory Service is a quad of the following form: **<service type, context name, hostname, port>**, that includes:

- the *service type* that specifies what kind of AmI artifact or script is active
- the *context name* that constitutes a human-friendly label that developers can use to identify the currently running instance among all running instances of the same type
- the *hostname* that uniquely identifies the host within the current AmI-Solertis infrastructure (Section 4.3)
- the *port* which indicates where exactly the REST service is running

To eliminate conflicts and ensure that resources are uniquely identified, Components Directory Service has established a single rule, which states that: *"multiple services can be registered using the same context name, as long as they are of different types"*; thus, a quad is characterized by its sub-tuple <context name, service type>. Components Directory Service exposes a REST interface that other modules can use in order to (among others): (i) register the connection details (i.e., hostname, port) of a new running instance of a certain service type, (ii) resolve the physical network address and port on which a given service type within a given context name is currently running, and (iii) discover which services are currently active in the entire ecosystem. Behind the scenes,

Components Directory Service has to communicate with (i) the centralized Executor service to get notified about the running status of the various components and (ii) the Infrastructure Manager (Section 4.3) to maintain an up-to-date list of the connected AmI-Solertis compliant hosts and their active network addresses.

Artifacts and Scripts Importer

AmI-Solertis offers a sophisticated method which undertakes the necessary tasks in order to assist developers on importing and integrating their software to the ecosystem. The import process is almost similar for both AmI scripts and AmI artifacts, with just one additional step needed in the case of an AmI script. In general, the importer requires that the 3-staged proxy generation process has been completed and its outcomes (i.e., the OAS, metadata, the proxy class) have been generated successfully. From that point onwards, the importer executes the following process (Algorithm 1) in order to automatically import the new item:

Algorithm 1 Import a new AmI artifact or script in the AmI-Solertis ecosystem.

Require: type, service, source **or** bin_archive
1: **if** ($type = AMI_SCRIPT$) **then**
2: $executable \leftarrow$ Utils.$ImportScript(source)$
3: $bin\_archive \leftarrow$ Datastore.$upload(executable)$
4: **end if**
5: Registry.$importService(service.data)$
6: Registry.$importSpecification(service.spec)$
7: Metadata.$import(service.metadata)$
8: Proxies.$import(service.proxy)$
9: **if** ($type = TYPE.SCRIPT$) **then**
10: AmIScripts.$import(bin\_archive)$
11: **else**
12: AmIArtifacts.$import(bin\_archive)$
13: **end if**

Apparently, incorporating a new item is mostly a streamlined process, as the importer just places the content in the right data store, with two exceptions where it has to perform some reasoning and conditionally do extra work. Firstly, if the new item is an AmI script, then the importer has to collaborate with the ProjectScaffolding Service and the AmIDatastore (Chapter 5) utility components, so as to generate, from the provided source code, the archive that will contain the service's executable code. Secondly, depending on the service type, the importer has to store the

service binaries to the right data store (i.e., AmI scripts repository or AmI artifacts repository).

Project Scaffolding Service

The AmI-Solertis Project Scaffolding Service aims to provide developers with an improved tooling workflow so that they can spend less time on process and more time focusing on building AmI-oriented applications and services. Behind the scenes it relies on the Yeoman tool [16] to kick-start new projects, prescribing best practices and tools, and scaffolding complete solutions. To that end, a variety of generator templates have been created and installed so that developers can easily prepare their working environment to build an AmI-Solertis compliant web-service, front-end or full-stack application.

Currently, the Project Scaffolding Service endorses the "Yeoman workflow" that enables developers to customize their stack and get everything that they need to get started without any manual setup. A scaffolded project generates a well-defined file structure that guides developers on where to place their code, utilizes the Gulp build tool [17] to automate time-consuming tasks in the development workflow (i.e., code transpiling and minification, obfuscation, compression, resource packing), and relies on the Node Package Manager (NPM)[18] for dependency management, so that developers no longer have to manually download and manage your any external libraries.

Finally, the Project Scaffolding Service also exposes a REST API that enables other components to use it in order to scaffold projects programmatically; for instance, the Artifacts and Scripts Importer of the AmI-Solertis Registry Service uses it to dynamically convert the script that the user has authored via the AmI-Solertis Authoring Tools (Chapter 6) into a full project that will be eventually imported, deployed and launched by AmI-Solertis.

API-Exploration Service

The AmI-Solertis framework exposes part of the functionality of its core components (FR2, Chapter 3) that external applications can: either integrate their functionality in their own system using the respective REST APIs or embed them as standalone, isolated and independent interactive elements that displays their own content (Chapter 5).

[16] See http://yeoman.io
[17] See https://gulpjs.com
[18] See www.npmjs.org

The Universal AmI Artifacts and Scripts Repository is such a component that offers a subset of its functionality through the API-Exploration Service component, via its REST API that enables developers to:

- query the registry so as to: (i) list all the integrated services, (ii) filter services by name, type, category or tag, and (iii) get extended details about them
- facilitate authenticated users or systems to administer the registry by executing CRUD operations (i.e., insert a new or delete an existing component).

The API-Exploration Service has been already used by the LECTOR framework [197] so as to: (i) integrate AmI artifacts and AmI scripts in its reasoning process (Figure 4.18), and (ii) publish new executable components that detect a variety of behaviors at runtime and apply context-aware interventions when necessary.

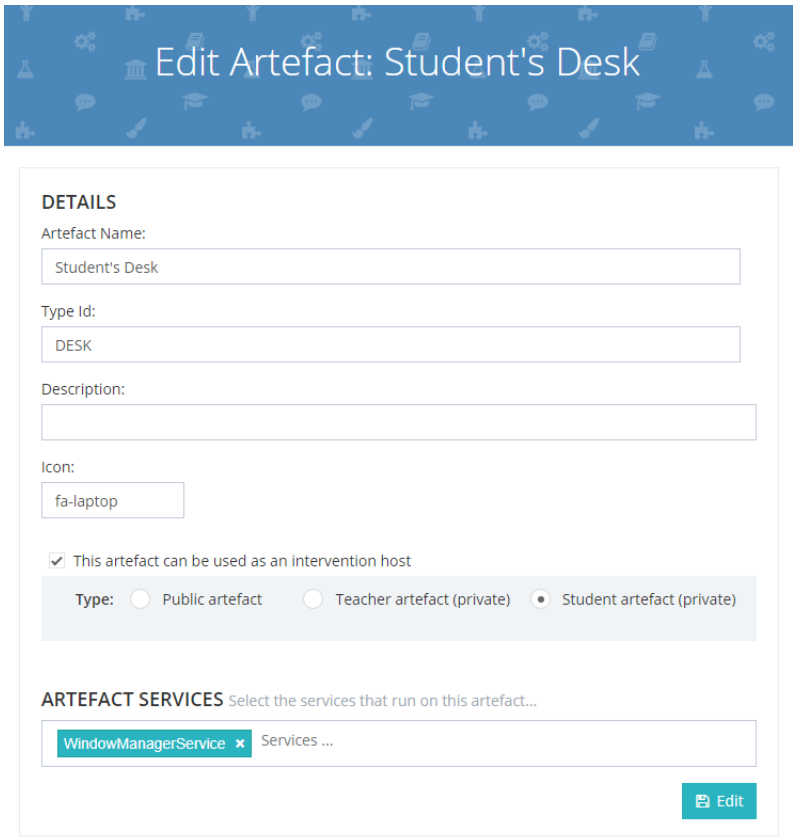

Figure 4.18: A snapshot of the LECTORstudio integrating the API-Exploration service [196].

4.3 Packaging, Deployment and Execution

Ambient Intelligence (AmI) environments incorporate a variety of heterogenous devices with different capabilities with fluctuating availability. AmI-Solertis aims to be deployed into and manage intelligent environments that incorporate many different stationary and mobile devices, in order to assist -amongst others- developers in distributing their components in the environment and monitoring their operation. To that end, it consolidates well-established practices and approaches in order to simplify deployment and minimize its cost, facilitate real-time management and operation monitoring, thus enhancing efficiency and increasing productivity. This section will discuss the core technologies that AmI-Solertis employs in order to simplify the packaging, deployment and execution process from the perspective of the developers (Figure 4.19).

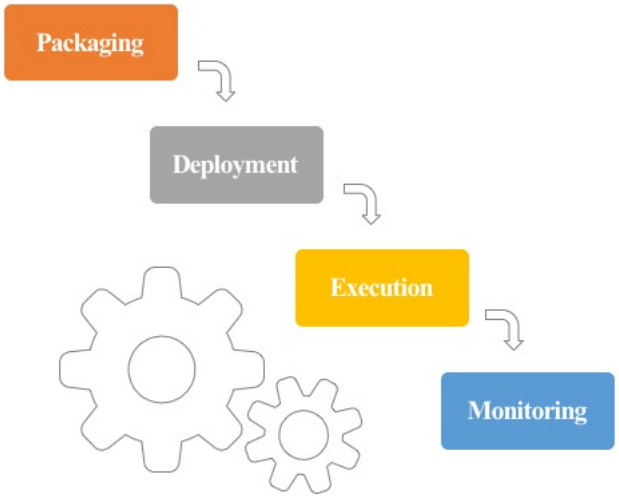

Figure 4.19: The outline of the deployment process

4.3.1 The Necessity of DevOps in AmI

DevOps is considered to be a culture, a movement, a philosophy, which as the name implies is a software engineering practice aims at unifying development and operations [229]. It emphasizes a shift in mindset, better collaboration, and tighter integration, as under a DevOps model, development and operations teams are no longer “siloed” [191]. DevOps integration targets continuous product delivery, quality testing, feature development, and maintenance releases in order

to improve reliability and security, provide faster development and deployment cycles, and deliver higher value to businesses and customers.

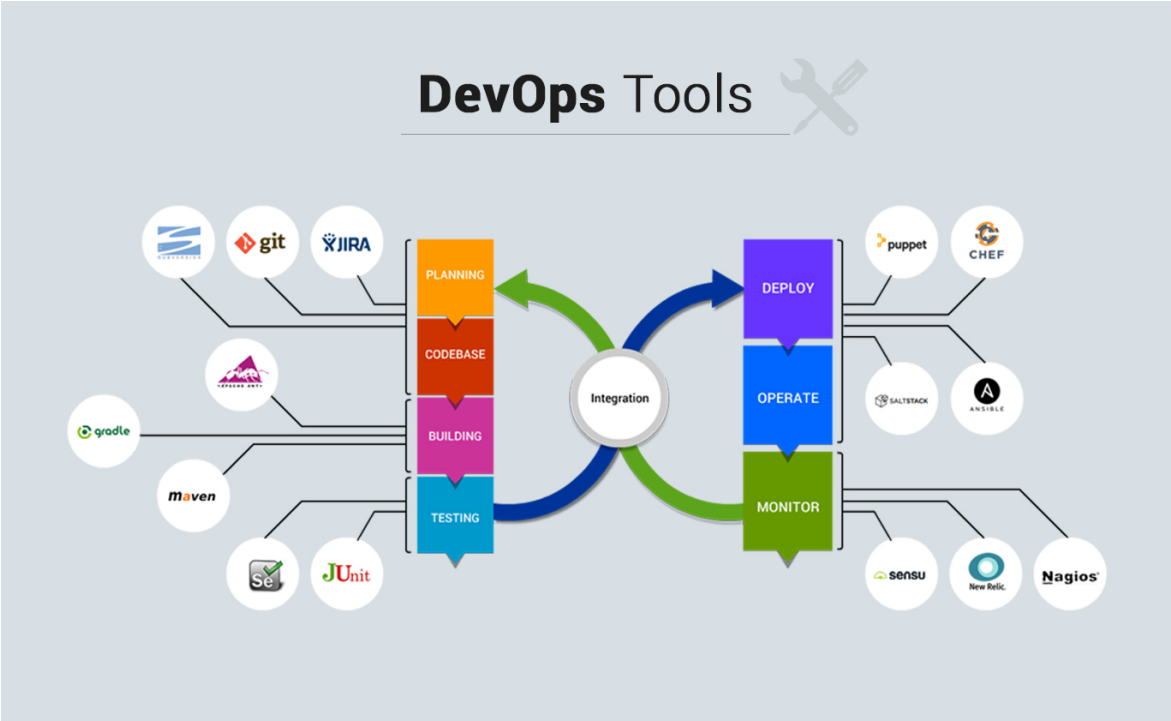

Figure 4.20: An indicative technology stack for supporting DevOps.

As literature suggests [39,326], DevOps offers some promising benefits, including: ability to deliver applications and services at high velocity thus significantly shortening time to market, lower failure rate of new releases, faster mean time to recovery (in the event of a new release crashing or otherwise disabling the current system), shortened lead time between fixes and ability to solve critical issues quickly thus improving the overall product quality, improved productivity and efficiency, increased ability to build the right product by fast experimentation, better management of unplanned work, improved customer satisfaction and increased trust. In general, by speeding up the overall process DevOps enable organizations to better serve their customers and compete more effectively in the market.

The main characteristic of the DevOps movement is to strongly advocate automation and monitoring at all steps of software construction, from integration, testing, releasing to deployment and infrastructure management [176]. Simple processes that historically have been manual and

slow, now become increasingly programmable and dynamic. A large technology stack that consists of a variety of tools (i.e., "toolchains") help engineers to independently accomplish tasks (for example, deploying code or provisioning infrastructure) that normally would have required help from other teams, thus further increasing a team's velocity (as presented in Figure 4.20). A DevOps toolchain include tools that fit into one or more of these categories, reflective of key aspects of the development and delivery process:

- Planning and Coding tools that facilitate code development and review, source code management tools and code merging
- Building automation via continuous integration tools that report at real-time the build status
- Testing tools that enable continuous testing and provide feedback on business risks
- Deployment tools that manage the artifact repository and prepare the application
- Operation tools that control the release process, while facilitating infrastructure configuration and management
- Control tools that permit real-time monitoring (e.g., performance, availability, congestion) of the overall AmI environment in order to identify potential issues that affect negatively the overall User Experience

Summing up, given that AmI environments are highly distributed, fluid and ever-changing, an appropriate DevOps toolchain can significantly automate the development, deployment and monitoring process of AmI artifacts and scripts within them, thus permitting developers to focus on building intelligent software that proactively satisfy the user needs and improves their overall quality of life.

4.3.2 Core Components

AmI-Solertis aims to be deployed into and manage intelligent environments that incorporate many different stationary and mobile devices. These device share their functionality over multiple network protocols and interoperate in order to create an advanced technological shell over the phys-

ical surroundings and to make the environment flexible, adaptive and naturally controllable, resulting in enhanced efficiency, increased productivity and overall greater well-being.

Table 4.4: CHEF Relevant Glossary

Term	**Description**
Chef-node	A physical, virtual, or cloud machine maintained by a Chef-client.
Chef-client	A command-line tool that runs Chef.
Chef-server	The hub for configuration data. The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by the chef-client. Nodes use the chef-client to ask the Chef server for configuration details, such as recipes, and file distributions.
Chef-repo	The structure in which cookbooks are authored, tested, and maintained.
Cookbook	The fundamental unit of configuration and policy distribution.
Recipe	A collection of resources that tells the chef-client how to configure a node.
Run-list	A list that defines all of the configuration settings that are necessary for a node that is under management by Chef to be put into the desired state and the order in which these configuration settings are applied.
Role	A way to define certain patterns and processes that exist across nodes in an organization as belonging to a single job function.
Knife	A command-line tool that provides an interface between a local Chef-repo and the Chef server. It can be used to manage nodes, cookbooks, recipes, roles, bootstrapping nodes, and searching the Chef server.

For that to be optimally achieved, developers and administrators of such environments have to be able to effectively manage the interplay between these distributed, dynamic and heterogenous hardware nodes with the software components that bring ambient intelligence into the picture. To that end, AmI-Solertis introduces appropriate models to describe: (i) the underlying hardware infrastructure, (ii) the AmI software packages, and (iii) their interplay, along with a variety of tools to facilitate the overall management of the AmI environment. AmI-Solertis uses CHEF.IO[19] (Table 4.4) as the basis to built its own AmI-oriented components that facilitate continuous au-

[19] See http://chef.io/

tomation (i.e., deployment, execution, monitoring) for the overall AmI infrastructure and the artifacts and scripts that it contains.

The AmI-Solertis Host

An AmI-Solertis Host is any physical, virtual, or cloud stationary machine (e.g., personal computer, Single-Board Computer) or mobile device (e.g., smartphone) that has be configured to host an arbitrary number of AmI artifacts and/or AmI scripts and is automatically maintained by the AmI-Solertis Infrastructure Manager. Every stationary host is initialized via an automated one-step process that starts when a user runs the AmI-Solertis Installer on any computer with a Fully Qualified Domain Name (FQDN) to automatically bootstrap the essential runtime components and register it as an compliant host. On the contrary, every mobile host is implicitly registered when any application that uses the AmI-Solertis Mobile SDK launches for the first time, and remains connected as long as the application runs on the foreground or the relevant device token that can be used to send Push notifications [24] to it remains active.

Infrastructure Manager

The Infrastructure Manager is responsible to coordinate the connected hosts via their respective Host Managers (see below) and cooperate with the CHEF-server during the software deployment process. This component maintains a list of all the machines that have ever been added in the AmI environment along with their current status. In addition to information listing, the Infrastructure Manager sets the role(s) of the connected hosts according to their functional role (e.g., AmIDesk [22], Hotel Room Controller [220], Teacher Desk [243]) and commands the local Host Manager to: (i) apply the deployment rules so as to ensure that a host will be ready to execute an AmI artifact or script in terms of the required software, and (ii) control the execution status of the already deployed components either automatically based on the role's execution rules or manually as defined by the administrator. In order to ensure that the Infrastructure Manager is constantly up-to-date regarding the status of the hosts that it monitors, it periodically communicates with the local Host Managers to verify their current state, while it exposes a status-oriented API that they can use to post updates about any change regarding their running state. The Infrastructure Manager exposes an more general REST API as well, through which other core modules can query

tha actual running instances about detailed information for their status or remote control their execution status (e.g., deploy a new application package, stop a running AmI script).

Roles Manager

Every artifact has at least one label that specifies its role in the AmI environment. In particular, a role defines a high-level category, which describe the expected minimum functional and non-functional requirements that a host must satisfy, so as to ensure that it will be able to effectively execute its designated duties that belong into a single job function. Its model includes: (i) a collection of *depedency rulesets* that specifies the necessary low-level software prerequisites (e.g., execution runtimes, libraries), (ii) a list of *application bundles* that determines which AmI software components (i.e., artifacts and scripts) should be deployed and running at all times, (iii) optionally a *set of tags* that provide additional information about the capabilities that the host should have (e.g., CUDA-ready [273]), and (iv) a CHEF role to facilitate its automated management via the CHEF infrastructure.

To facilitate management, multiple roles can be assigned to a host in order to automate its management in a macroscopic scale (i.e., an update in a role will automatically affect every host with that role). Using roles, engineers can create arbitrary groups of hosts to implement a variety of distributed computing models (e.g., Fog Computing, Cloud Computing). For instance, the Fog Computing model dictates that edge devices near users can be utilized to carry out a substantial amount of storage before communicating with the Cloud; such a model could be easily supported by AmI-Solertis in the context of an Intelligent Home.In more detail, the dedicated PC in every room could serve that room's needs (e.g., interpret sensor readings, control devices), whereas the home mainframe machine could act as the main cloud server. Similarly, in an Intelligent classroom, the workstation of each student can be the edge resource (i.e., Fog edge node) that when necessary propagates any high-level request to the classroom mainframe (i.e., Cloud resource). Finally, every AmI-Solertis Host contains an instance of the AmI-Solertis Host Manager that exposes appropriate REST APIs to empower its remote management (e.g., start or stop an AmI component, deploy a new component, report running state).

Dependency Rule

Dependency A dependency rule refers to any kind of software such as: programming library (e.g., NPM package), runtime environment (e.g., .NET CLR), application (e.g., Google Chrome™) or another AmI component (e.g., Card Detection Service [239]), which is absolutely necessary to an AmI artifact or script in order for it to be able to run properly. Multiple dependency rules can be combined into a rule set so ensure their reusability.

AmI-Solertis Software Package

An AmI-Solertis Software Package contains an AmI artifact or an AmI script in an executable format (e.g., the production version of an Express-based webapp) along with any auxiliary, yet necessary, binaries. For instance, a FAmINE-based application is accompanied by the automatically generated bridge webapp that exposes its functionality over the REST protocol and the AmI-Solertis Event Federator. Internally, a software package contains a manifest file that includes an automatically generated CHEF cookbook with the CHEF recipe for that particular instance and the CHEF run-list, which guides the CHEF-client on what to download and install when instructed to do so by the Infrastructure Manager. Multiple packages can be combined into an application bundle to simplify batch management.

Software Manager

Software Manager exposes an API to the AmI ecosystems that empowers other components (including the graphical Authoring tools) to create, modify and retrieve the available Dependency rules, Dependency rule sets, Software Packages and Application bundles.

Deployer

AmI-Solertis Deployer is a the module that orchestrates the deployment process of an AmI artifact or script on one or multiple AmI-Solertis Hosts. For that to be achieved, the Deployer communicates with Universal AmI Artifacts and Scripts Repository, the Infrastructure Manager and the Software Manager in order to collect the necessary data (i.e., software package, dependencies, host connection details, CHEF-related parameters) and forward them to the local Host Manager of the specified host to start the installation process.

Executor

AmI-Solertis Executor communicates the Host Managers of the connected host so as to control the execution status (i.e., start, stop) of the deployed AmI software components (i.e., artifacts or scripts). The Executor communicates with the Infrastructure Manager and the Components Directory Service in order to update them regarding the status of the various hosts. Moreover, to empower AmI scripts to programmatically control the execution status of a given host (and in principle of the the overall AmI environment), it exposes most of its functionality in the form of a REST API.

AmI-Solertis Host Manager

The Host Manager is a collection a various mini-tools that run on a host and enable its remote control. In particular, the Host Manager includes: (i) the Configurator that enables remote parameters modification (e.g., Role), (ii) the CHEF-client bridge that uses the local CHEF-client to communicate with the main CHEF-server when necessary (e.g., to execute a CHEF recipe given its recipe identifier, to install a required software library via Chocolatey [30]) (iii) the Deployer client that interacts with the central Deployer regarding the installation of AmI software packages and/or application bundles, (iv) the Executor client that receives start and stop commands from the central Executor component and propagates them to the OS, through custom controllers that manage common runtime environments, such as a PM2 [342] wrapper to control NodeJS applications, a C# mini-application that handles Windows-oriented applications (i.e., .EXE), and (v) the Health client that reports detailed information about host's current status to the AmI-Solertis Health Monitoring Component (Chapter 6). Finally, the host manager in cooperation with the local CHEF-client ensures that the host will always be configured into the desired state. Specifically, it will ensure that all the software components that defined either in the roles' dependency rule sets and application bundles or have been manually added via the AmI-Solertis Authoring Tools or programmatically via other AmI scripts, are deployed and currently running. Therefore, AmI-Solertis ensures that the host will always be able to achieve is intended objectives.

4.3.3 Packaging process

In the context of AmI-Solertis, packaging is the process where an AmI artifact or an AmI script, which has been already integrated in the AmI ecosystem via the import process, gets prepared for deployment (i.e., transfer and installation) in an AmI-Solertis compliant host (as depicted in Figure 4.21). Through packaging, AmI-Solertis generates and packs together all the necessary resources (i.e, executable files, configuration files, CHEF directives) that should be transferred to the host (via the Deployment process) in order to be present and ready to be used when a relevant execution commanded is given.

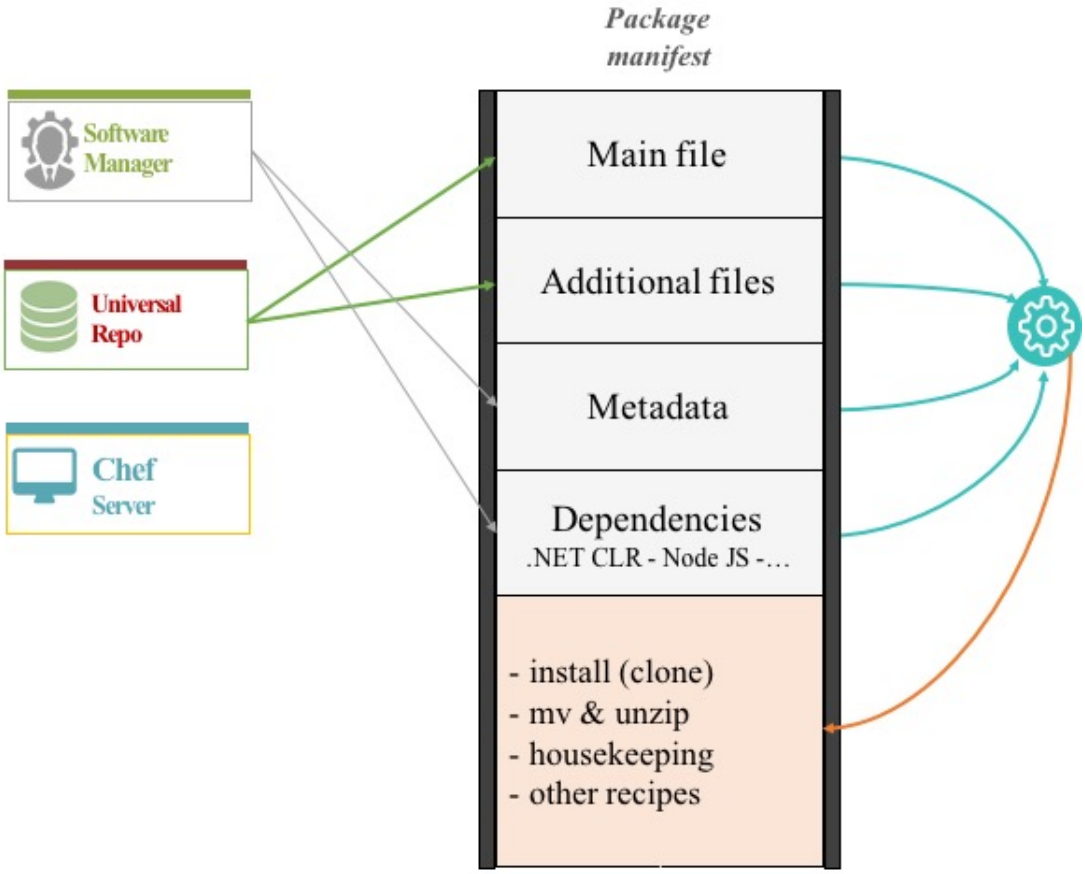

Figure 4.21: The packaging process prepares an AmI Script for deployment.

During packaging, AmI-Solertis generates a package manifest that aggregates: (i) the URL of AmI component's executable file(s) (either in a binary format or as executable source code), (ii) the

URL of any auxiliary executables that will be necessary (e.g., a FAmINE-to-REST bridge), (iii) metadata about the AmI components to be deployed, (iv) directives to AmI-Solertis about its configuration requirements, (v) the dependencies rule set that the host should satisfy, and (vi) a CHEF cookbook with the relevant recipe(s) that describe which command(s) the CHEF client will have to execute to properly install this components along with any dependencies.

All the executable files (e.g., AmI component, auxiliary bridge) are stored in the appropriate AmI Components Repository, whereas the package manifest is uploaded in the AmI-Solertis Datastore. Most of these information are retrieved by components that belong to the Universal AmI Artifacts and Scripts Repository. A software packages is generated after the import process of the respective AmI component completes, while its manifest is populated as follows:

1. As soon as the executable files gets upload in the repository, the relevant repository name and its URL are noted, so that Host Manager will be able to retrieve them using the relevant installation mechanisms (i.e., git clone command). If the component being uploaded now is a new AmI script, before storing it to the repository, AmI-Solertis firstly scaffolds a new web project in which it injects the user-authored content, next it "generates" the production version of that web app and finally it uploads the final outcome in the repository.

2. If auxiliary executables are necessary, then they are generated as soon as the basic import process completes; in particular, the importer instructs its internal software factories to create a bridge that will wrap the new component. Currently, AmI-Solertis supports the automatic creation of software bridges (using Nustache[20] and Edge.js [21]) that mask FAmINE-based services or applications into webapps that expose their functionality over REST and emit their events via the AmI-Solertis Event Federator.

3. Metadata are extracted directly by the Service Importer when inspecting the new component to generate it formal OAS specification.

4. Dependencies to specific runtime environments (e.g., NodeJS, JRE, .NET CLR) and the relevant executor wrappers (see below) are manually provided by the user.

[20] See https://github.com/jdiamond/Nustache
[21] See http://tjanczuk.github.io/edge/ and https://msdn.microsoft.com/en-us/magazine/mt703440.aspx

5. CHEF-related information are generated after the import process completes. Specifically, for every software package a new cookbook is created with a single recipe that: (i) instructs the CHEF-client to retrieve the main and auxiliary executable(s) from the respective AmI Components Repository, (ii) moves the downloaded content to the right directory, (iii) unzip it if necessary, (iv) perform general "house keeping", and (v) defines its dependencies to other system-level recipes that install the required runtime environments (if needed).

With respect to application bundles, users via the respective graphical tool available in the AmI-Solertis studio, can create their own bundles (i.e., application suite) that will collect together multiple AmI artifacts and/or scripts that dependent to each other and are expected to be invoked in the same host. Therefore, instead of manually downloading them separately an application bundle automatically aggregates the individual manifests and when necessary guides the AmI-Solertis components towards their deployment.

4.3.4 Deployment process

The AmI-Solertis's deployment process is a multi-stage pipeline, which requires the cooperation of many core components so as to automate the overall process by combining information from multiple sources (i.e., current state of the environment, user-input, AmI artifact or script meta-data, configuration information). The deployment pipeline (Figure 4.22) consists of following four discrete stages:

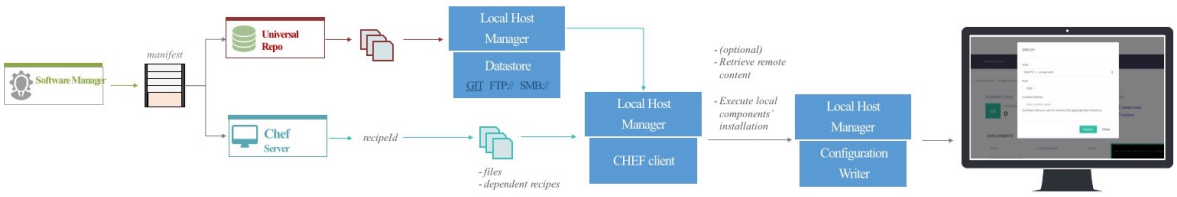

Figure 4.22: The pipeline of the deployment process.

Stage 1: Data Collection

1. Retrieve via the Software Manager the respective AmI Component manifest.
2. Retrieve the AmI component's metadata and executable files from the Universal AmI Repository.

3. Upload the CHEF-related information (i.e., cookbooks, recipes,) on the CHEF Server using CHEF's Knife command line tool.

Stage 2: Host Preparation

4. Retrieve the connection details of the target host from the Infrastructure Manager.
5. Forward the new recipe identifiers to the local Host Manager to update the host-specific CHEF run-list and include the new CHEF recipe(s).
6. Command the Host Manager via the central Deployer module to start the deployment process.
7. The local CHEF client connects with the CHEF server in order to retrieve the detailed recipes that comprise its run list.
8. The local CHEF client inspects the provided run list and executes the recipes that are not currently synchronized (i.e., their components are not currently present on the host).
9. For every new recipe, the CHEF client downloads and extract the executable files of the AmI component along with the additional runtime environments if necessary (e.g., JRE).

Stage 3: AmI Component Installation

10. Upon download and extraction completion the Host Manager receives control from the CHEF client and it moves on with the installation of any additional libraries independently of CHEF (e.g, install required NPM packages).

Stage 4: User-driven Configuration

11. Before completing the overall process, the user is asked to fill-in: (i) the required component-independent configuration details that should be present so that the component could successfully launch (e.g., context name, port), and (ii) any component-dependent configuration parameters that has been defined in the relevant metadata.

As soon as the installation of every component and its dependencies completes, the local Host Manager reports to the central Deployer that the deployment process has been completed; from that point onwards, the target host will permanently have the new software installed and will be able to launch it when instructed to do so (if not active).

4.3.5 Execution and Live Management

AmI-Solertis relies on the central Executor module to control the execution of the various AmI components (i.e., AmI artifacts and AmI scripts) as depicted in Figure 4.23. Every local Host Manager is itself a REST service that can be instructed to control (i.e., start or stop) any AmI component that has been deployed in the host that it controls. Internally, the Host Manager contains various "executors" that can launch different types of executables (e.g., NodeJS applications, Windows Executable Applications).

Currently, the Host Manager module supports NodeJS applications by wrapping the PM2 process manager [22] and Windows Executable Applications via a custom daemon that resides on the Windows tray, which can launch and manage multiple .EXEs; additionally the Host Manager internal architecture enables over-the-air update of such sub-modules in order to support new types of executable software (e.g., Docker [252], Python scripts). In addition to managing execution, these custom daemons also watch to the current status of the component at real-time in order to either auto-restart them in case of a failure or just report these data to the central Executor. Finally, local executors facilitate management of logs coming from the AmI Components in real-time. They intercept any output messages emitted to the system's main output stream and forward them to the AmI-Solertis Studio so that users can easily check their operation without having to remotely access the host and check the console outputs.

An interesting aspect of the AmI-Solertis's central Executor is again the fact that it also exposes its functionality in the form of an API that other components can use to programmatically communicate with any local Host Manager and control the execution of any AmI component. The execution process is straightforward:

1. Either the user manually via the AmI-Solertis Studio or some AmI script automatically has notified the central Executor to start or stop an AmI component on a single or multiple hosts.

2. The central Executor communicates with the Infrastructure Manager to appropriately route the command to the right local Host Manager.

[22] See http://pm2.keymetrics.io

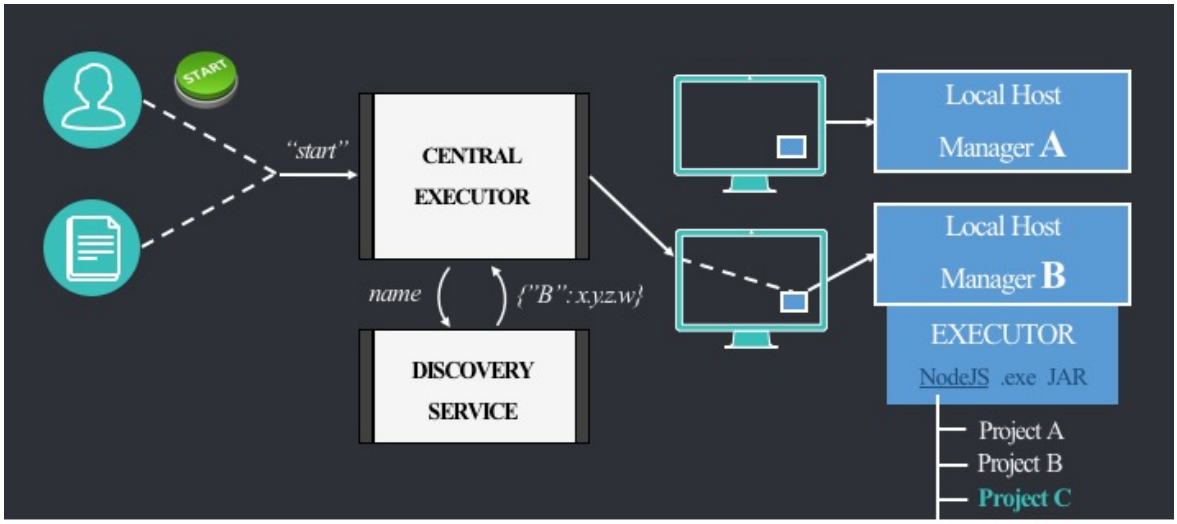

Figure 4.23: The execution process.

3. Based on the type of target executable, the Host Manager resolves the appropriate executor and propagates the remote instruction.

4. The type-specific executor executes the command by taking into consideration whether it affects a single component (i.e., AmI script) or multiple ones (i.e., application bundle, an AmI artifact and its auxiliary bridge).

5. The local Host Manager reports the outcome of the command back to the central Executor, which in turn notifies the Components Directory Service and the Infrastructure Manager about the current status of the host that it controls (e.g., host down, AmI component with context name X and type Y has started in port P).

4.3.6 The AmI-Solertis approach to Continuous Integration

AmI-Solertis endorses the concept of Continuous Integration [110, 130] as it considers it to be a vital importance in such dynamic environments since it enables, among others, fast introduction of new features and quick resolution of critical issues. As modern DevOps approaches state, multiple ways exist towards launching new features or updating existing ones (i.e., Dark Launching, Blue-Green Deployment, Canary release, Rolling Back Deployments) [174, 176].

AmI-Solertis offers a flexible mechanism that enables Hot Deployment Services to apply any

of the aforementioned continuous deployment practices. In particular, the fact that the intercommunication between AmI artifacts and scripts is fully decoupled from their actual instances via the AmI-Solertis Proxies, corroborates that by simply switching their implementation, updates can be performed on-the-fly. Moreover, the AmI-Solertis Standard Library permits developers to also add their own strategy towards replacing proxies with respect to current uses (i.e., drop requests while updating, queue them and forward them when restored in order).

Chapter 5

AmI-Solertis Standard Library

AmI-Solertis, in addition to its core components, relies on a number of auxiliary tools and libraries that are necessary for its proper operation. This software is referred to as AmI-Solertis Standard Library, and is available to any developer who wants to create a new AmI artifact or script, implement new core functionality for the AmI-Solertis framework, or build external tools that interoperate with AmI-Solertis to introduce new features.

5.1 Analytics and History Logs

AmI-Solertis collects various statistics and logs in order to facilitate offline analysis that can improve the overall system performance by detecting: (i) poorly performing components, (ii) potential bottlenecks, (iii) potential problems, (iv) performance issues, (v) anomalies, (vi) components that frequently fail, (vii) unmet QoS goals, (viii) common patterns of use that can be optimized, (ix) components that exhibit excellent behavior and could be used as best-practice. For that to be achieved, a variety of components monitor the AmI environment to aggregate and store health-related information.

5.1.1 Event Recorder

The Event Recorder communicates with the central Event Federator in order to obtain information about the usage of the event-based communication channel and facilitate its introspection. The Event Recorder also exposes an API that the Event Federator Client incorporates, which publishers

and subscribers can optionally activate in order to send detailed information about their activities (i.e., explicitly mark the emission / receipt of an event to/from a channel).

Basic data of interest include:

- total channels
- total subscribers
- total number of events sent
- total subscribers per channel (indicator of channel's popularity)
- mean number of subscribers per channel
- total number of events sent per channel (indicator of channel's activity)
- mean number of events sent per channel
- number of subscribers per channel
- number of events per channel (activity indicator)
- active or inactive channels
- popular or less popular channels.

Advanced data of interest include:

- number of publishers per channel (indicator of channel's popularity)
- mean number of publishers per channel
- total events by publisher (indicator of publisher's activity)
- total events by publisher per channel (indicator of publisher's activity in channel)
- most active publishers in total (based on the total events sent in a channel)
- most active publishers per channel (based on the total events sent in a particular channel)
- most influential publishers (based on the total events sent that were handled)
- most influential publishers per channel (based on the total events sent that were handled in a particular channel)
- total events handled per channel
- total events handled by subscribers

- total events handled by subscribers per channel (how many responded in that channel)
- most active subscribers (based on their total subscriptions)
- total handled events by subscriber (based on the number of handled events)
- total handled events by subscriber per channel (based on the number of handled events in a particular channel)
- total handled events by subscriber per publisher (based on the number of handled events from a particular publisher)
- total handled events by subscriber per publisher per channel (based on the number of handled events in a particular channel from a particular publisher)
- popular subscribers per channel (based on the number of events that they have handled in that channel)
- popular subscribers per publisher (based on the number of events that they have handled from that publisher)
- popular subscribers per publisher per channel (based on the number of events that they have handled from that publisher in that channel)
- frequency of event emission in total, per channel, per publisher, per channel and per publisher.

5.1.2 Logging, Tracing and History Trails

AmI-Solertis is a highly distributed system that follows a microservice-based architecture. As in any kind of software, developers usually log various information that enable them to both verify the behavior of their program and detect potential errors. Since AmI-Solertis aims to facilitate developers in building their software, it exposes a common distributed logging service that internally encapsulates a powerful async logging library, namely Winston [1]. The logging service enable developers to log events with different severity levels (e.g., log events with at least a warning level vs. verbose log lines). Currently six different levels are provided, namely: (i) error, (ii) warn, (iii) info, (iv) verbose, (v) debug, and (vi) silly, which can be further customized as necessary.

[1] See https://github.com/winstonjs/winston

In addition to logging, to ensure its efficiency and verify its proper functioning, AmI-Solertis enables tracing of its various distributed operations, so as to facilitate engineers in their day-to-day work. In such distributed systems, error detection and execution tracing is a very complicated task due to the flood of alerts, messages, events. Distributed tracing and tools [37, 343] that aid in understanding system behavior and reasoning about performance issues by providing insights on transactions and errors are invaluable in such an environment. To thad end, AmI-Solertis adopts the widely used practice of correlation identifiers passed in all the service transactions, so as to enable searching on labeled data via appropriate front-end tools [2] ; the AmI-Solertis Proxy integrates the appropriate tracing mechanisms that can be activated and deactivated on demand, whereas a Tracing service is available so that AmI scripts can use it as well.

Finally, tracing along with message resilience can also be used to reveal history trails and compute the degree of interdependence between the various software components; such information can significantly contribute to any reasoning components that monitor AmI-Solertis aiming to improve its operation (i.e., performance, robustness, fault tolerance).

5.1.3 Usage and Metrics

AmI environments consist of many heterogenous computational resources with different capabilities (i.e., more powerful stationary devices vs. less powerful mobile devices). As with any distributed system or a cloud environment, load balancing can help: (i) distribute the dynamic workload across multiple nodes to ensure that no single node is overloaded, (ii) proper utilization of resources, (iii) and significantly improve the system's overall performance and throughput [189]. But efficient load balancing can only be achieved if utilization-related information (e.g., CPU usage, Memory usage, Network traffic, Storage usage) are collected for every participating host. AmI-Solertis has introduced a relevant Usage and Metrics collection service that can be used by any host to transfer such information to the core system; currently this library is integrated into the Host Manager component that runs in every AmI-Solertis compliant host and automatically sends such data that developers can view in the relevant section of the AmI-Solertis Studio (Chapter 6).

[2] See http://zipkin.io and https://github.com/jaegertracing/jaeger

5.1.4 Health Recorder

Resource metrics are low-level data that provide useful information regarding the physical components of resource (i.e., host). Nevertheless, high-level components such as a database or a geolocation microservice or an AmI script, are also be considered resources if another system requires that component to produce work. AmI-Solertis, in an attempt to reconstruct a detailed picture of a system's state that will be especially valuable for investigation and diagnosis of problems, exposes the Health Recorder service that collect for each AmI component of the ecosystem metrics that cover four key areas:

- **Utilization** is the percentage of time that the resource is busy, or the percentage of the resource's capacity that is in use.
- **Saturation** is a measure of the amount of requested work that the resource cannot yet service, often queued.
- **Errors** represent internal errors that may not be observable in the work the resource produces.
- **Availability** represents the percentage of time that the resource responded to requests. This metric is computed by regularly checking every resource for availability (i.e., watchdog [105]).

All these metrics are used to compile the live health report for each AmI component at real-time and subsequently permit its profiling and metering; this report is very useful for the core components that apply fault-tolerance strategies (see below). Example metrics for an AmI script could be:

Utilization: average % time each request-servicing thread was busy

Saturation: enqueued requests

Errors: internal errors such as caught exceptions

Availability: % time service is reachable.

5.2 Fault Tolerance policies

When deployed in a full-fledged AmI environment (e.g., an Intelligent Home), AmI-Solertis has to manage dozens -if not hundreds- of AmI artifacts and scripts, which inevitable will fail at some point or under-operate, more often that their creators think [71]. Even though microservices limit the side-effects of any error from affecting the overall AmI environment, AmI-Solertis has introduced a number of libraries and services that aid developers in creating and installing defense mechanisms and exception-handling policies for addressing abnormal operation, stop cascading failures and prevent the ripple effect.

5.2.1 Runtime Argument Checker

As mentioned earlier, the REST protocol by itself neither specifies the methods that are offered by a service nor the data types and valid values that these method expect. Therefore, developers consuming REST services might, either by mistake or by ignorance, provide incorrect parameters (e.g., missing parameter, wrong data type, value is out of range) to the remote service which will result into a failure. If the remote service has been careful engineered (i.e., adopting defensive programming strategies [386]) that failure will only affect the incorrect call; if not, it may result into a crash thus affecting any subsequent calls. To address that shortcoming, AmI-Solertis embeds the Runtime Argument Checker library in every AmI-Solertis Proxy, that developers can optionally activate in order to validate the arguments of a remote method call. Internally, the checker consults the formal OAS specification to ensure that all arguments are of the right type, and if the remote service provides information regarding the accepted value ranges it examines that aspect as well. If the conditions are met, the calls is forwarded to the remote service, otherwise the call gets blocked (to secure the remote service) and a runtime exception is raised.

5.2.2 Policy Enforcer

The Policy Enforcer is a service that AmI-Solertis has introduced to let developers make their software more resilient by adding robust transient fault handling logic through well-known exception-handling policies [76]. Transient faults are errors that occur because of some temporary condition

such as network connectivity issues or service unavailability. Typically, if the operation that resulted in a transient error is retried a short time later, the error might probably disappeared [44].

Different services can have different transient faults, and different applications require different fault handling strategies. Thus, the Policy Enforcer enables developers to express their own policies and includes by default (using the hystrixjs library[3]) the following ones:

- **Retry**: it retries a call up to a number of times (three by default) until it fails with an exception.
- **Circuit Breaker**: it trips a circuit-breaker to stop all requests to a particular service for a period of time, if the error percentage for this service passes a configured threshold.
- **Timeout**: it ensures that the caller will never have to wait beyond the configured timeout.
- **Bulkhead Isolation**: it acts both as an isolation unit, and (intentionally) as a load-shedder. To preserve the health of the underlying machine, the bulkhead intentionally sheds load when its capacity and queue are exhausted.
- **Fallback**: it provides a substitute value in the event of failure.

Policy enforcer has pre-installed a simple decision-making model that permits it to offer basic functionality, but that logic can be easily updated (via external services) to include more sophisticated algorithms (e.g., Artificial Intelligence, Ontologies and Semantic Reasoning).

5.2.3 Module Replacement Library

AmI-Solertis decouples the various components by introducing the concept of proxies that intercept the communications between the remote ends. That practice facilitates Continuous Integration by permitting the replacement of services in real-time if necessary (e.g., bug fix, introduction of new feature) as depicted in Figure 5.1. Moreover, the same technique can be applied to introduce new instances of a service to accommodate load balancing. In practice, the Module Replacement Library is an AmI script that is deployed in the ecosystem and applies its strategies by utilizing the APIs exposed by the central Deployer and Executor modules. With respect to the actual

[3] See https://bitbucket.org/igor_sechyn/hystrixjs

replacement process, the library relies on contextual knowledge to determine the exact update strategy; for example, if a service is inactive it can be immediately replaced, whereas if it executes an operation the update should be postponed. Similarly, if during update a request for that service occurs, then the library can append it into a queue and upon successful restart forward them appropriately (Figure 5.1).

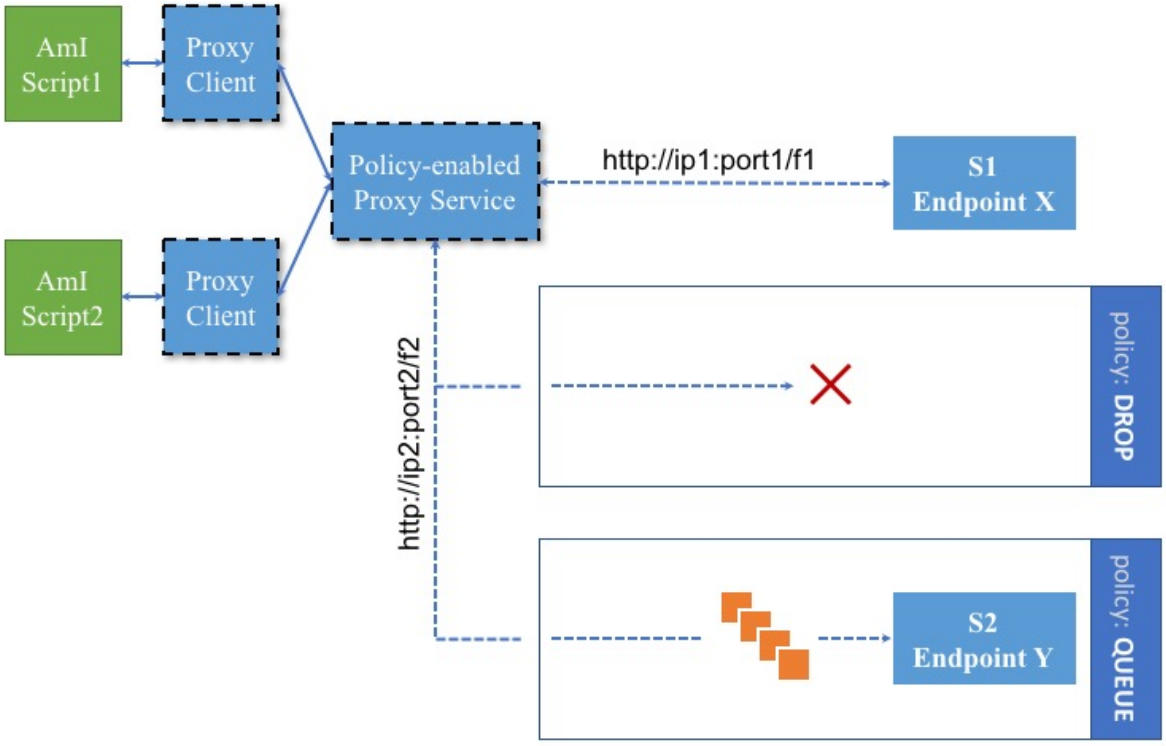

Figure 5.1: Service replacement at real-time.

5.2.4 Service Profiler

The Service Profiler module communicates with various AmI-Solertis components in order to collect information about the currently available AmI artifacts and scripts, and create their "operational profile" that provides useful insights to the Policy Enforcer module. This module is an AmI script as well, thus its functionality can be easily enhanced through external services that introduce advanced profiling models (e.g., based on Ontologies [281]) or more complex techniques to assess and predict [411] the performance of a remote service. Moreover, the Service Profiler

provides appropriate mechanism that other components can use in order to classify the available services (to facilitate exploration through the AmI-Solertis Studio) or to discover compatible equivalents if necessary (a process known as matchmaking [414]); for instance, that need might arise in case a service's performance degrades and should be substituted by an equivalent one as presented in Figure 5.2).

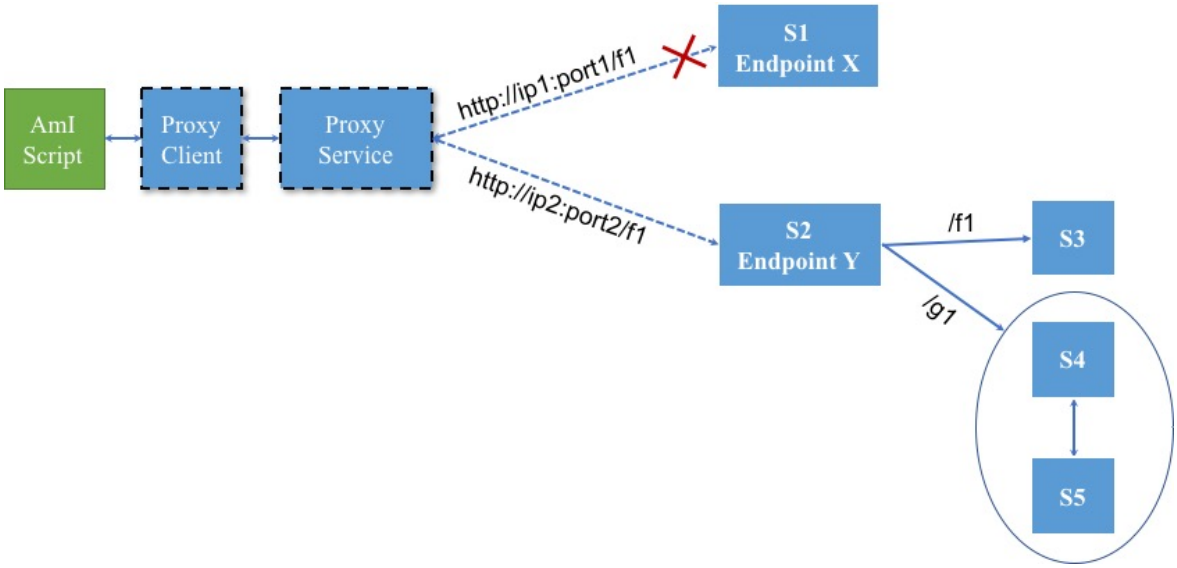

Figure 5.2: Service Adaptation is facilitated by the relevant Proxy.

5.3 Storage Management

One common requirement that is shared across the majority of ubiquitous environments is the need to exchange content [207] that goes beyond a simple textual message. AmI-Solertis offers a collection of utilities that AmI components can use in order to store their content either for their own private use or for sharing with other components.

5.3.1 Memorystore

The Memorystore is implemented as an AmI artifact that communicates with a persistence layer (currently MongoDB™[4]) and exposes an API to the system through which the other components

[4]See https://www.mongodb.com

can perform any Create-Retrieve-Update-Delete operations on their data. With respect to the supported data types, currently the implemented schema enable the storage <key : value> pairs, where the key is kind of string and a value is either a JSON object or any POJO object that can be converted to JSON. Nevertheless, any component that wishes to utilize this module, can encode its data using its own proprietary format and simply serialize them in a string format. The Memorystore will insert that data and upon retrieval the components should deserialize them to get the actual values.

5.3.2 Datastore

The AmI-Solertis Datastore enable applications to easily exchange binary content (e.g, documents, images, videos). It is build on top of the Minio[5] object storage server. Object storage (also known as object-based storage [254]) is a computer data storage architecture that manages data as objects, as opposed to other storage architectures like file systems which manage data as a file hierarchy and block storage which manages data as blocks within sectors and tracks. Each object typically includes the data itself, a variable amount of metadata, and a globally unique identifier.

The Datastore uses the Memorystore to store the various object identifier in order to facilate advanced content sharing. In particular, it supports authentication and offers three different levels of access, namely private, limited and public. Private files are only accessible by their owners. Files with limited access can be shared only with the components that the uploaded has granted access. Public files are accessible by everyone as long as they know how to retrieve them (i.e., using their unique identifier). Finally, Datastore offers a temporary storage area, where files that have a limited life-span are stored. Currently, the Datastore permits the component that uploaded the object to set its availability duration or the maximum number of permitted downloads; in both cases, as soon as the threshold is crossed, the object is immediately removed from the storage.

5.3.3 AmI-Context

Contextual-Awareness is the cornerstone of Ambient Intelligence, since without it an environment is unable to sense and understand its surroundings, let alone to proactively react to human needs

[5] https://www.minio.io

or adapt to their needs. Over the years many alternatives have been proposed [356, 390], while four main representation alternatives exist according to [295]: (i) **binary**: small, portable but difficult to extend, (ii) **object-based**: complex data structures support, (iii) **attribute-value pairs**: limited complexity compared to object-based solutions, language- and platform- independent, and (iv) **XML-based**: more complex structures, substantial overhead in term of network communications and processing.

Given the various alternatives, AmI-Solertis introduces a mechanism towards managing contextual data (including the addition of different components when necessary), rather than creating yet another model for describing context. Internally, it is built on top of MemoryStore and provides an extended API that developers can use on the one hand to manipulate and share context information from device to device or situation to situation, thus supporting continuity and mobility, and on the other hand explore for contextual information through the AmI-Solertis Studio.

5.4 Utilities

AmI-Solertis also offers a set of programming tools and libraries that can help developers build compatible applications or integrate popular AmI-oriented external systems in this platform.

5.4.1 AmI-Solertis Installer

In order to simplify the preparation of a node to become an AmI-Solertis compliant host that will be able to accommodate AmI-Solertis artifacts or scripts, platform-specific installers have been implemented that automate that process. In particular, these tools download and install the core components (i.e., CHEF-client, AmI-Solertis Host Manager) and their dependencies (e.g, .NET CLR, NodeJS, PM2) that are mandatory for remote management through the AmI-Solertis platform and configures basic settings (e.g., enable remote access/winrm[6], add an exception to the firewall, configure Host Manager to run at startup). As soon as everything is in place, then the node is registered as an AmI-Solertis enabled host and starts is operation.

[6]See https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/enable-psremoting?view=powershell-5.1

5.4.2 AmI Configuration Service

Given that every AmI-Solertis component (i.e., AmI artifact, AmI script, Core component) is automatically managed by the platform and is deployed in a dynamic, distributed manner, some configuration settings should be loaded dynamically as well to facilitate that process. Thus, AmI-Solertis offers the AmI Configuration Service that components can use in order to load program-critical information dynamically at runtime (e.g., the context name that will be used, the port on which they will be launched, the connection details to the central Event Federator).

These information are provided during the deployment phase, where the AmI-Solertis Studio enables developers to supply such kind of data through the AmI Configuration Service. Every component can optionally define its own settings that require a-priori configuration and ask the AmI-Solertis Deployer to also ask the user to fill them in. Eventually, these user-oriented information are stored locally and from that point onward they are solely managed by the Host Manager via the local configuration service.

5.4.3 Event Federator Client

AmI-Solertis has introduced a central Event Federator through which all the AmI components (i.e., AmI artifacts, AmI scripts, Core components) can communicate in an event-based manner (Figure 5.3). To simplify its use, the Event Federator Client library wraps the boilerplate code necessary to configure, register and communicate with that main server (implemented using Redis PUB/SUB feature[7]). It exposes the bare minimum methods through which a component can (i) send events to any channel, or (ii) specify the channel that should be monitored for any broadcasted events and register the function that will handles the event's payload; the payload is automatically transformed to a Plain Old Java Object (POJO). As regards data marshalling [8] and unmarshalling [9] [369],

[7] https://redis.io

[8] Marshalling or marshaling is the process of transforming the memory representation of an object to a data format suitable for storage or transmission, and it is typically used when data must be moved between different parts of a computer program or from one program to another. Marshalling is similar to serialization and is used to communicate to remote objects with an object, in this case a serialized object.

[9] Unmarshalling or unmarshaling refers to the process of transforming a representation of an object that was used for storage or transmission to a representation of the object that is executable. A serialized object which was used for communication can not be processed by a computer program. An unmarshalling interface takes the serialized object and transforms it into an executable form.

this is automatically handled by the library, thus the consumer only has to supply the desired data.

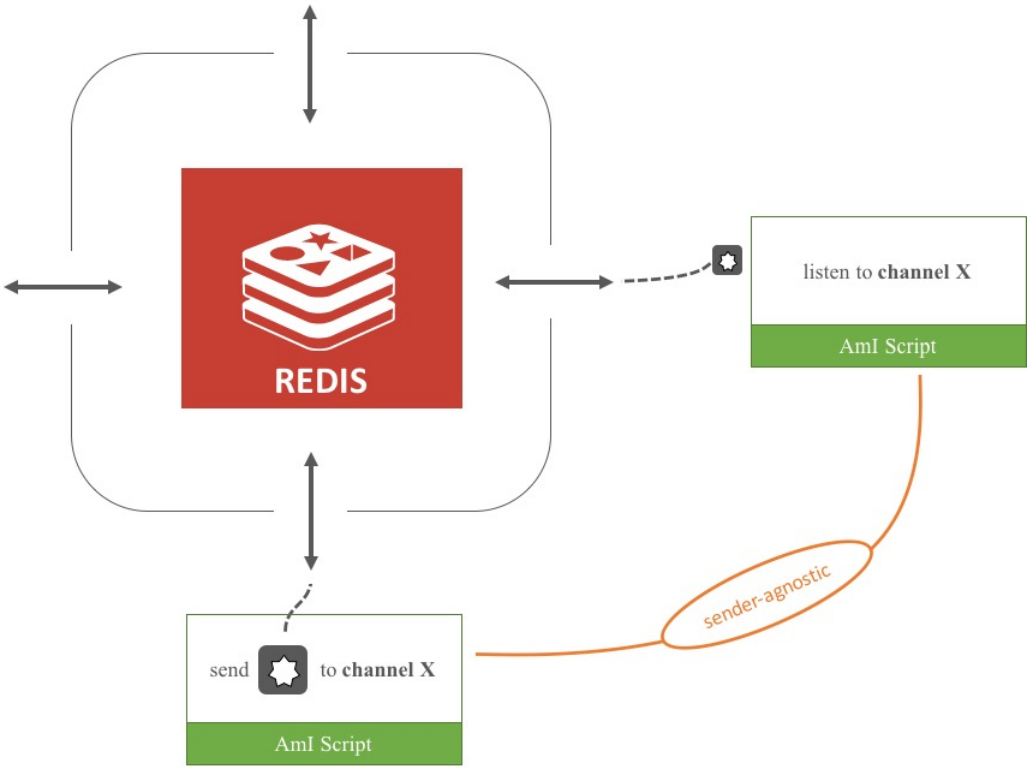

Figure 5.3: The AmI-Solertis Event Federator.

5.4.4 Application Runtime Registry

In an AmI environment, more than one instances of the same front-end application may be active at any time (e.g., Museum Guide [187], PUPIL [198], Home Game [219]). Therefore, an AmI script or an AmI artifact that controls them should be able to determine which exact instance to target. Depending on the application type different approaches exist. In particular, applications targeting smartphones or desktop environments can generate a unique identifier at their first launch and use that to recognize different instances. Web application instances on the other hand can be identified by the different websocket [10] identifiers that they have established with the server to receive new data without having to constantly poll the server. AmI-Solertis provides the Appli-

[10] WebSockets [262] is an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.

cation Runtime Registry Service (that internally relies on the AmI-Solertis Memorystore) which exposes a straightforward API for storing these unique identifiers in an always accessible location and retrieve them on demand.

5.4.5 AmITest

The AmITest framework [230] facilitates testing and validation of the functionality of each individual AmI component. As a core component of AmI-Solertis, AmITest utilizes the available meta-information regarding the installed AmI component and behavior definitions, in order to simplify the overall testing process. More specifically, a wizard-like process enables even less-experienced users to easily compose the desired tests, while the test execution orchestration is automatically handled.

To minimize its footprint and increase its usability, AmITest optionally offers a sandboxed environment in which testing can be executed without affecting the actual system and its artifacts. Specifically, it provides AmI component impersonation capabilities (e.g., mock how a smart artifact reacts to stimuli) and local execution of the AmI script under inspection.

Finally, the internal reporting system of AmITest delivers the test execution outcome to the tester through a rich user interface in order to assess the effectiveness of the AmI script and validate the desired assertions and invariants. That information is also fed back to the AmI-Solertis system to be further examined for potential erroneous behaviors and to ensure the environment's robustness.

5.4.6 Home-Automation Plugin

AmI-Solertis is currently deployed in the ICS-FORTH AmI Facility Building, which -among others- uses the KNX[11] protocol for Home and Building Control. In order to transfer control data to all building management components, KNX ensures that all isolated devices communicate via a common language. This standard is based upon more than 24 years of experience in the market, amongst others with predecessor systems to KNX: EIB, EHS and BatiBUS. Via the KNX medium to which all bus devices are connected (twisted pair, radio frequency, power line or IP/Ethernet),

[11] https://www.knx.org/knx-en/index.php

they are able to exchange information.

Bus devices can either be sensors or actuators needed for the control of building management equipment such as: lighting, blinds / shutters, security systems, energy management, heating, ventilation and air-conditioning systems, signalling and monitoring systems, interfaces to service and building control systems, remote control, metering, audio / video control, white goods, etc. All these functions can be controlled, monitored and signalled via a uniform system without the need for extra control centers. The Engineering Tool Software (ETS)[12] is a manufacturer independent configuration software tool that enables to design and configure intelligent home and building control installations with the KNX system. Nevertheless, this tool is intended to "hard-wire" devices and events with specific actions, thus limiting the scalability and extensibility of the environment.

AmI-Solertis though uses KNX in a slightly different manner. It considers it to be yet another source of sensorial data (e.g, temperature, rain sensor, brightness controller, presence detector) accompanied by a collection of devices that can be remotely manipulated (e.g., shutter actuators, heating actuator, fan coil actuator). To that end, it has introduced the KNX Bridge artifact in order to encapsulate the KNX-specific infrastructure and expose an equivalent REST API. A KNX Bridge artifact is automatically generated using the KNX installation details from the ETS tool along with some additional user-provided metadata (i.e., details about the physical installation). The AmI-Solertis KNX Integration Assistant has been implemented to aid KNX experts in incorporating the necessary metadata that streamline the creation of the bridge when configuring their installation via the ETS.

5.4.7 Artificial Intelligence and Cognitive Robotics Linkage

An AmI environment according to its formal definition has to proactively react in order to address its user needs. Multiple reasoning approaches have been proposed [35, 114, 155, 193, 264, 321, 399], but as [295] highlights, it is important to employ multiple reasoning techniques such as Bayesian networks, probabilistic and fuzzy logic, as each technique performs well in different situations [276].

[12] See https://www.knx.org/knx-en/software/ets/ets-professional/index.php

In particular, AmI-Solertis foresees to integrate technological solutions coming from the domain of Cognitive Robotics [288, 289] and Semantic Web [46] which target dynamic environments (i.e., AmI space, IoT, CPS). Using such technologies developers can: (i) define the preconditions that need to be met before proceeding with the execution of a certain action„ (ii) specify the events of interest and which contextual parameters they can potentially affect, (iii) model the physical surroundings, the devices and the available services, (iv) address the frame problem, and (v) partially (via semi-decidable languages such as ASP) or fully (using always decidable languages such as OWL-DL) address the ramification problem. Moreover, such technologies empower developers to tackle harder problems (especially as their systems scale and become more complicated) such as: (i) validation and verification of their services, (ii) synthesis of a strategy or an action sequence that is guaranteed (when applied to any of the initial states) to generate a state which contains the desired goal (i.e., planning), and (iii) the generation of new knowledge by observing the AmI environment.

To that end, AmI-Solertis acknowledges that AI is a key ingredient of AmI environments [240], but it does not offer any implementations of specific algorithms. On the contrary, its overall architecture has been carefully designed so as to enable the introduction of any AI technique -in the form of an AmI artifact-, which could be used by any AmI script to define its intelligent behavior.

Chapter 6

The AmI-Solertis Studio

This chapter will present in details the various frontend facilities of the AmI-Solertis Studio. Firstly it will provide an overview of the provided functionality, and then will discuss the major components through which developers interact with the AmI-Solertis framework so as to create, deploy and monitor the software that control the behavior of an AmI environment, namely: (i) Components Repository, (ii) Code Editor, (iii) Control Panel, (iv) Infrastructure Manager, and (v) Analytics Viewer. Finally, the chapter will conclude by presenting AmI-Solertis chat-bot, the virtual agent in the form of a chat-bot, that can communicate with the end-users via a natural language textual interface in order to help them accomplish numerous orchestration-related tasks.

6.1 Overview

The AmI-Solertis Studio aims to assist developers in creating, exploring, deploying, and optimizing the AmI scripts (i.e., programs) that control the behavior of the AmI environment by combining and orchestrating various AmI artifacts or other AmI scripts that reside in it. To that end, it offers various exploration facilities that accelerate the discovery of AmI components (i.e., artifacts and scripts), a web-based code editor with enhanced support features through which developers can author their programs, a control panel to deploy and manage (i.e., monitor at real-time, start/stop, test) the existing components, the infrastructure manager through which the developers can administer the AmI-Solertis compliant hosts and the analytics section to get detailed information about the operational state of the AmI environment. To do so, it offers an intuitive user

interface that categorizes the available functionality under four (4) main menu items (see Figure 6.1), namely: (i) **Components**, (ii) **Control Panel**, (iii) **Infrastructure**, (iv) **Analytics**, (v) and a fifth auxiliary item, called **Various**, that encapsulate less frequently used features (e.g., the downloads section).

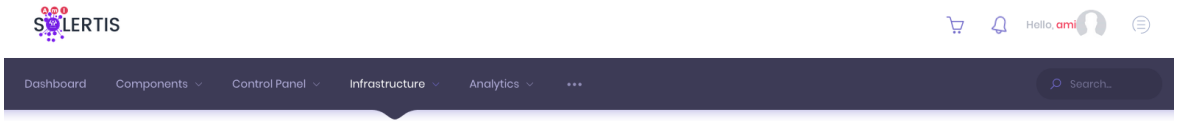

Figure 6.1: The AmI-Solertis Main Menu.

One of the main features of the AmI-Solertis Studio is that it enables developers to explore the available component (i.e., AmI Scripts, AmI Artifacts, Other Libraries) and collect into a virtual "bag" those that they would like to include in a new AmI Script. In addition to browsing (see "Components Repository"), a system-wide search area permits users to quickly lookup for software components from any page. From a functionality point of view, this "bag" of components not only enables the editor to provide context sensitive support (e.g., automatic generation of code that initializes access to these proxies, auto-completion of proxy methods), but also informs the Packaging component and the Deployer about the software packages (or application bundles) that need to be downloaded and installed in the host before executing the actual AmI script.

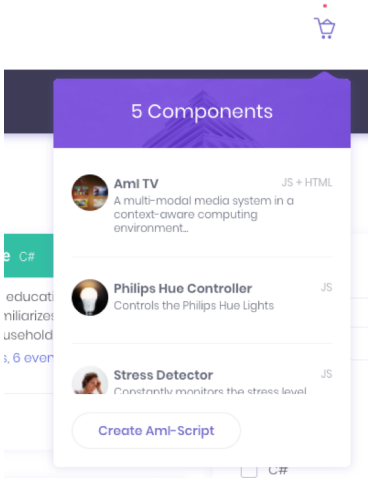

Figure 6.2: The expanded cart popup.

The shopping cart metaphor is employed to represent the components' "bag" in a user-friendly

manner. The top right corner of the AmI-Solertis Studio contains a cart icon; as soon as the user adds a component to the cart, a balloon indicator appears displaying the number of the currently added components (as displayed in Figure 6.2). Upon selection, a popup appears displaying the components, appropriately classified according to their type (i.e., AmI Script, AmI Artifact and External Library). At this point, if all necessary components have been collected, the user can select to go directly to the code editor to start editing the new AmI script.

The AmI-Solertis Studio Dashboard. It is a visual display of data used to monitor conditions and/or facilitate understanding. Moreover, in the context of AmI-Solertis Studio, it is an interactive display that allows users (i.e., developers) to explore various information about the AmI environment (e.g., health, recent logs) and perform some quick actions (Figure 6.3).

In particular, users can get some general statistics about the current state of the AmI environment whose data are presented in their **personalized** dashboard, such as: the number of: (i) active AmI-Solertis compliant hosts compared to the total number of registered host, (ii) running AmI scripts versus all the AmI script that are available in the AmI ecosystem, and (iii) finally, the respective counts of the active running and imported AmI artifacts. Below, a scrollable component with various health-related diagrams is presented where the user can get a quick overview across the key metrics about the system (e.g., performance, errors). It provides quick information at a glance, but also allows for deeper analysis by comparing the metrics over time.

Next, the user can see a list view of the recently started or created components. Every row starts with an icon that indicates the type of the component (i.e., AmI artifact or AmIscript) and its name, and then differentiates according to its type; specifically in case of recently started components the context name and exact date/time of its launch are displayed, whereas in the case of a newly created components the name of the author is presented next to the creation date.

On the right hand side an quick actions sidebar is provided to the users from where they can launch commonly used tasks without leaving the home page. For instance, a user can initiate the deployment process of an single AmI component, or view the already deployed components that are not currently running and launch them. Below the sidebar, at the bottom right corner a drop-zone permits a user to drag'n'drop an archive of an AmI artifact that will be automatically uploaded to the AmI-Solertis server. Users will be able to review it and finalize its import later.

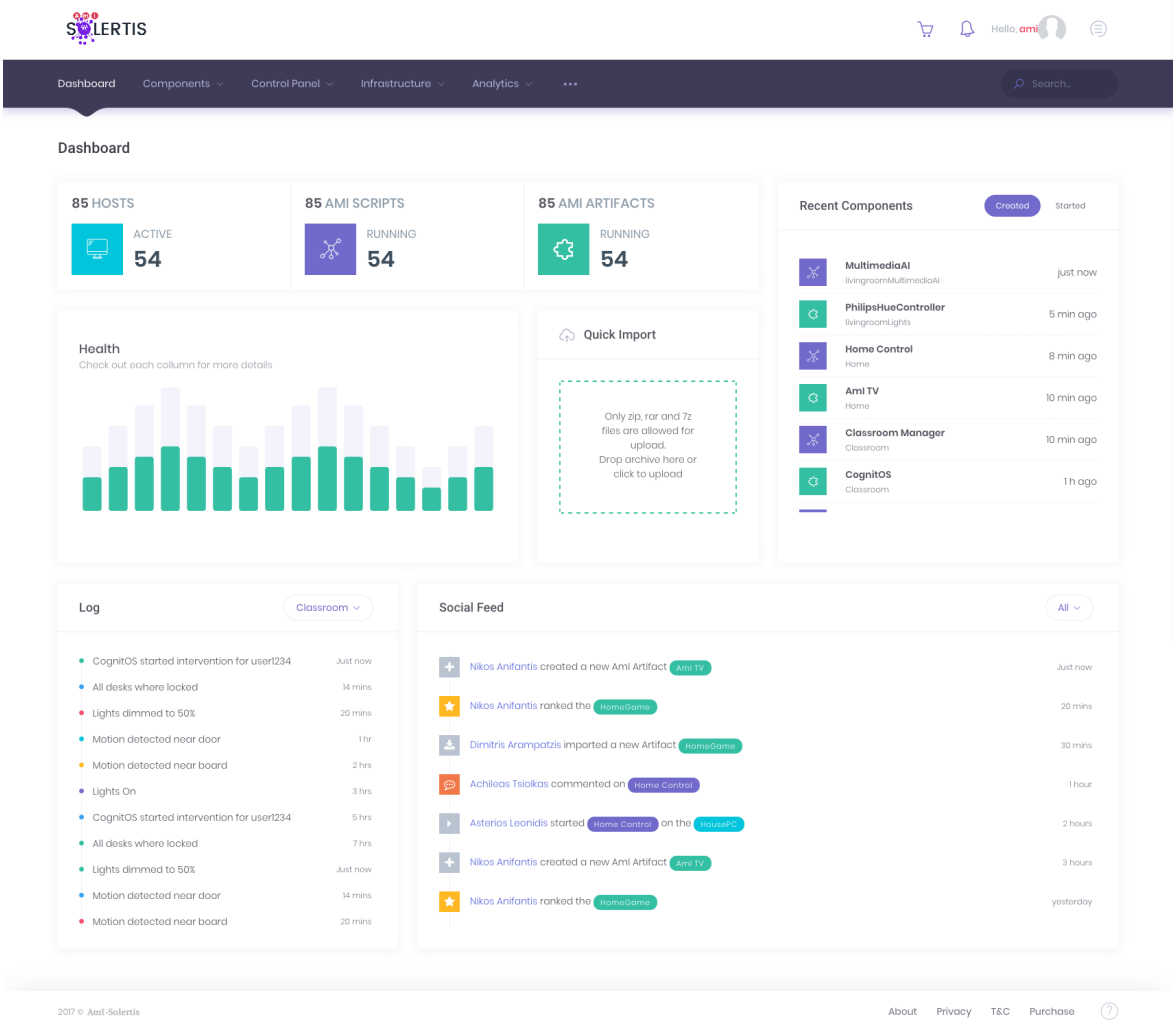

Figure 6.3: The AmI-Solertis Dashboard.

Beneath these sections, the user is presented with two infinite scrolling lists, one displaying all the messages that have been logged, and the other data from the social feed. With the respect to the logs feed, given that it can grow exponentially, an advanced filtering mechanism is available through which the user can customize that component according to her own preferences; for instance, the user can select to display only messages for AmI script X, or filter the logs to display all the messages the AmI recorded by all the components that are running with the context name Y. Finally, the social feed enables the user to get information about other users or components that she follows, view previews of comments of discussions that she is participating to (e.g., relevant to

AmI components that she has created) and ranking lists of the most popular components in the ecosystem (e.g, most used, most followers, most active instances).

6.2 The Components Repository

The AmI-Solertis Studio aims to assist developers in creating, exploring, deploying, and optimizing behavior scripts that combine and orchestrate the various technological facilities of AmI Environments. To that end, it provides a variety of exploration tools, namely the Web Explorer, the VR Explorer and the location-aware AR Explorer, that permit users to browse through the list of available components (i.e., AmI artifacts, AmI scripts, Libraries).

6.2.1 Web Explorer

Through the **Components** menu item, a user can visually explore and add to her cart AmI Scripts, AmI Artifacts and Other Libraries. The components are represented in the form of tiles, which contain useful information to ensure their discoverability (Figure 6.4).

In terms of content exploration, the most commonly used, yet highly efficient universal feature, is content classification based on explicit categories. Previous studies have shown it to simplify browsing [136] for almost 50% of targeted users [4, 305]. Content organization is further enhanced with tag annotations that can be set in a category-independent manner, on a classification-by-use basis [20]. Moreover, the AmI-Solertis system, facilitates the discovery of new and noteworthy content by promoting featured, popular and trending applications or artifacts based on their impact [3]. To assist users who prefer searching over browsing, it also provides a rich filtering mechanism and a faceted-search facility that allows users to narrow down the visible components by specifying various parameters of their query (e.g., Platform, Type, Category, Tags). Given that the overall user experience is strongly influenced by the level of personalization on the displayed content [195], AmI-Solertis delivers (i) personalized recommendations based on each user's areas of interest and (ii) context-sensitive lists of recently-viewed artifacts and components for quicker reference.

For each **AmI artifact** or **AmI script**, the user can get detailed information about their exposed

functionality, navigate among all their currently running instances in the Intelligent Environment and examine their status (e.g., running, stopped, deployed), explore their usage across the AmI-Solertis Ecosystem, and receive QoS-related information (e.g., mean response time) and extensive statistics of their use (e.g., popular methods, frequency of use, real-time logging). With respect to external Javascript libraries, which are collected by querying the official NPM repository, the tile is slightly different as it includes the package name, its version, description, any associated keywords and a few QoS-related data (i.e., quality, popularity and maintenance).

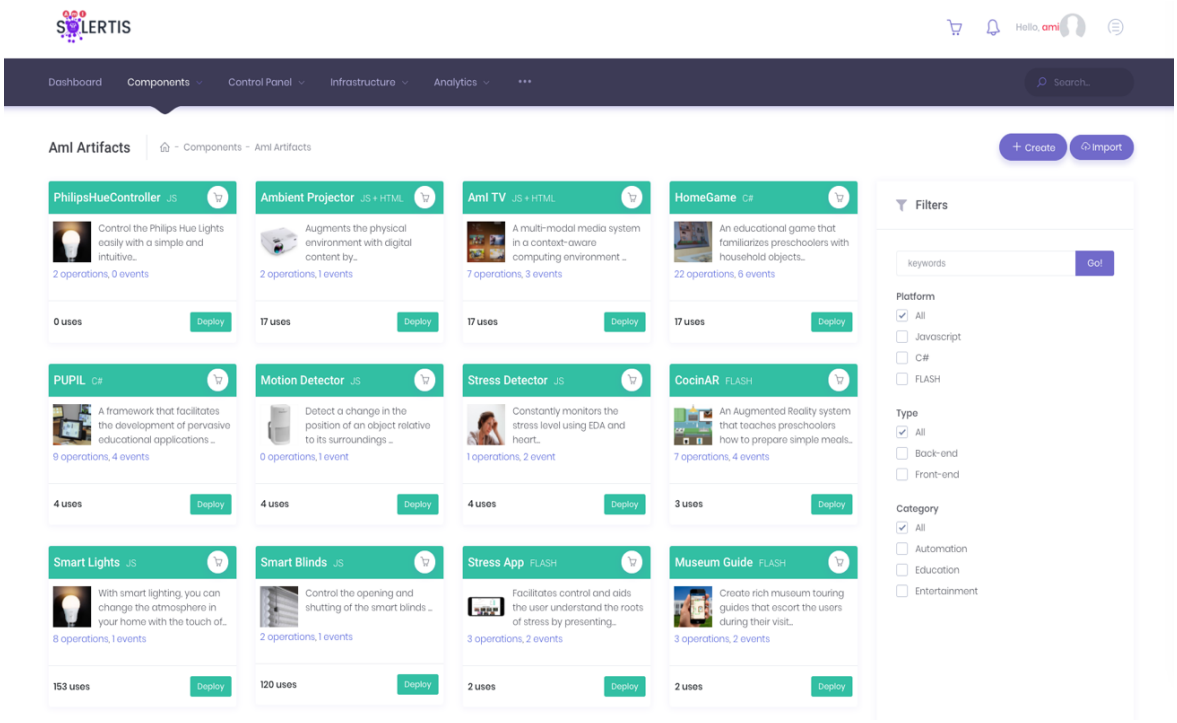

Figure 6.4: A snapshot of the Web Explorer showing the available AmI artifacts.

6.2.2 VR and AR Explorers

In addition to the web-based view, AmI-Solertis Studio offer two alternative exploration tools that developers can use in order to collect the services that they would like to use to build a new AmI script. The VR explorer permits the user to navigate in a photo-realistic 3D-model of the physical space and search for any running instances of AmI components in that virtual environment. For every component, information similar to the web-based view is presented (i.e., name, short

description, status), along with an appropriate control that adds the selected item to the user's "shopping cart", thus making it available in the web-based editor. The benefit of this approach is that the user can take into account the spatial location of the AmI components before combining them in an AmI script (e.g., to properly distribute partial UIs of a ubiquitous application in multiple displays within an AmI environment).

Similarly to the VR Explorer, the AR Explorer is delivered as a location-aware mobile application for smartphones (i.e., currently only iOS™is supported), through which users can explore the physical environment in order to discover any available facilities and collect them in their cart (Figure 6.5). Any deployed or running instances of AmI components -along with their details (e.g., name, context name, short description)- are superimposed over the physical scene that is captured by the smartphone's camera. Blending digital objects (i.e., AmI components) and information with the environment around the user enables developers to explore and use unfamiliar environments (e.g., museum) in a more natural way. It can also be useful to empower designers of AmI experiences to schedule "brainstorming session in the wild" to get a better understanding of the physical surroundings that they have to design intelligent behaviors for.

Figure 6.5: The AR Explorer as presented in an iPhone.

Both these facilities are currently used to explore the various facilities and to manage the AmI-

Solertis compliant host that are physically distributed in the AmI environment (e.g, deploy an AmI component, start/stop already deployed ones).

6.2.3 The Artifact

The details page of an AmI artifact or script collects various data about it from the entire AmI ecosystem and presents them to the user in a concise manner (Figure 6.6). In particular, the user firstly can get information about its exposed functionality (i.e., offered operations and events that it emits), its score based on users' ranking, and the list of its contributors (i.e., individual developers that have aided in its Implementation). Moreover, a user can get detailed information regarding its actual use in the AmI ecosystem. Specifically, the number of AmI scripts that include it, the total number of hosts where it has been already deployed along with the number of currently running instances. In addition to these quantitative data, the user can graphically see, information about the detailed use of each running instance present: total operation calls, total events emitted, and number of errors occurred per host, thus enabling the user to overview its performance.

Next, a list of all active deployments is provided. For every stopped instance, the user is able to start its execution (i.e., similar to how a user launches a locally installed application in her computer), whereas for any running instance, its console output and/or log messages are presented so that the user can view the functional state of it (i.e., running properly, it has errors, it is not responding to calls), can stop it or quickly evaluate its actual health in real-time (i.e., simulate a call, query its health API).

AmI-Solertis among others enable collaboration between its users; AmI components are the prevalent way through which developers can contribute to the AmI ecosystem. Therefore detailed information about the "social impact" of a components are provided, including the number of followers, the overall activity (e.g., use in an new AmI script, new deployment at host X, user A added a new comment), whereas all user can discuss and comment its functionality or share their experience while using it. Finally, a few quick actions enable users to: (i) update this component by uploading a new version (if it is an AmI artifact) or edit via the built-in editor (if it is an AmI script), (ii) delete it from the AmI ecosystem, (iii) add it to cart, or (iv) deploy the current version of the AmI component in a new host.

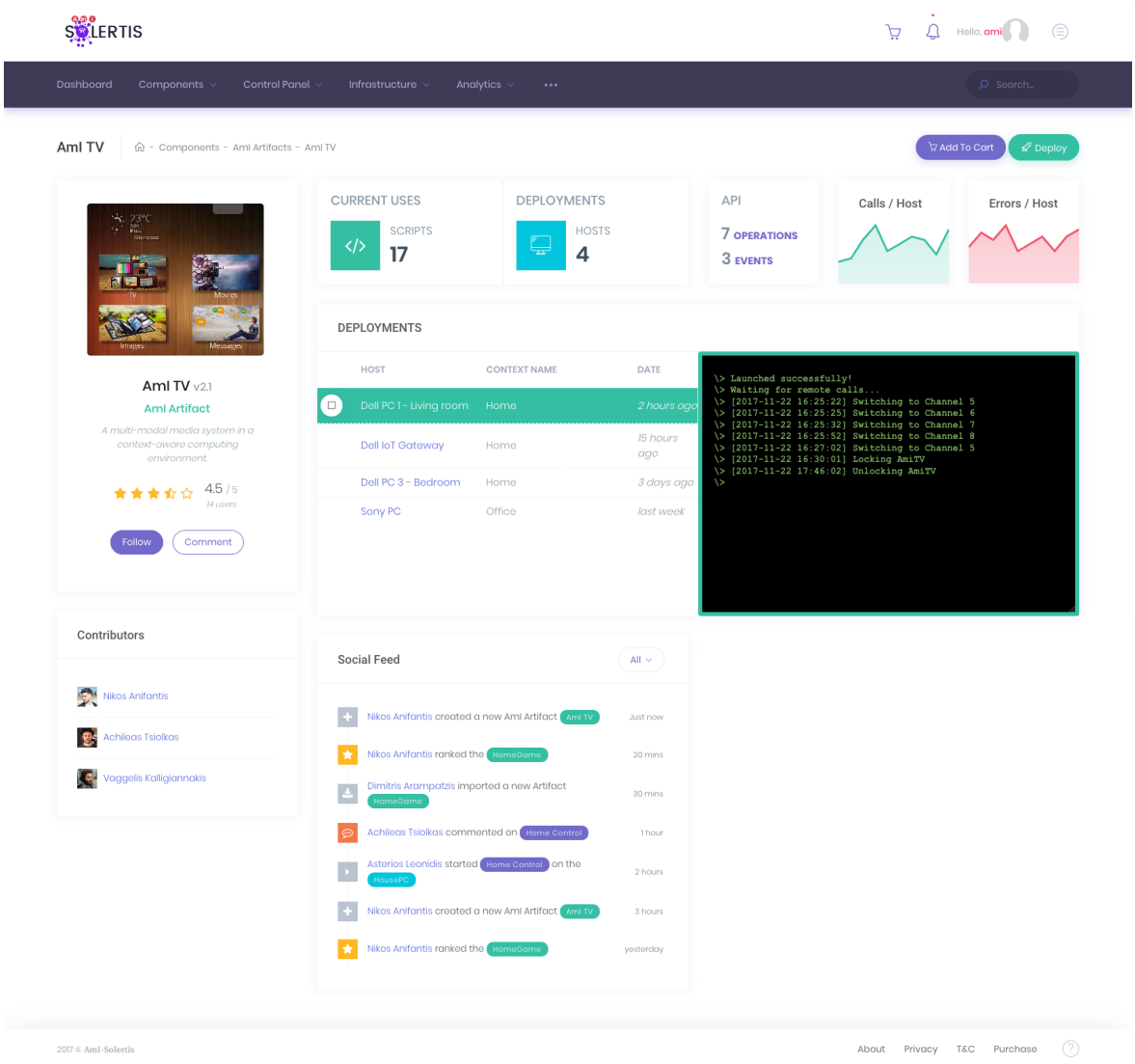

Figure 6.6: The AmI component's details screen.

6.2.4 Service Importer

New AmI artifacts that introduce innovative services can be easily registered to AmI-Solertis via a dedicated graphical interface, the Service Importer, whose role is two fold: enable file uploading and fine-tune the relevant proxy generation process. During the transfer phase, the user is asked to upload a compressed file (i.e., .ZIP, .RAR,. 7z) that contains either the distribution-ready[1] version

[1] A production build of a system focuses on minifying bundles, reducing the weight of source maps, and optimizing assets to improve load time.

of the AmI artifact (which was scaffolded by any of the AmI-Solertis project generators) or all the necessary binary files for it (Figure. 6.7).

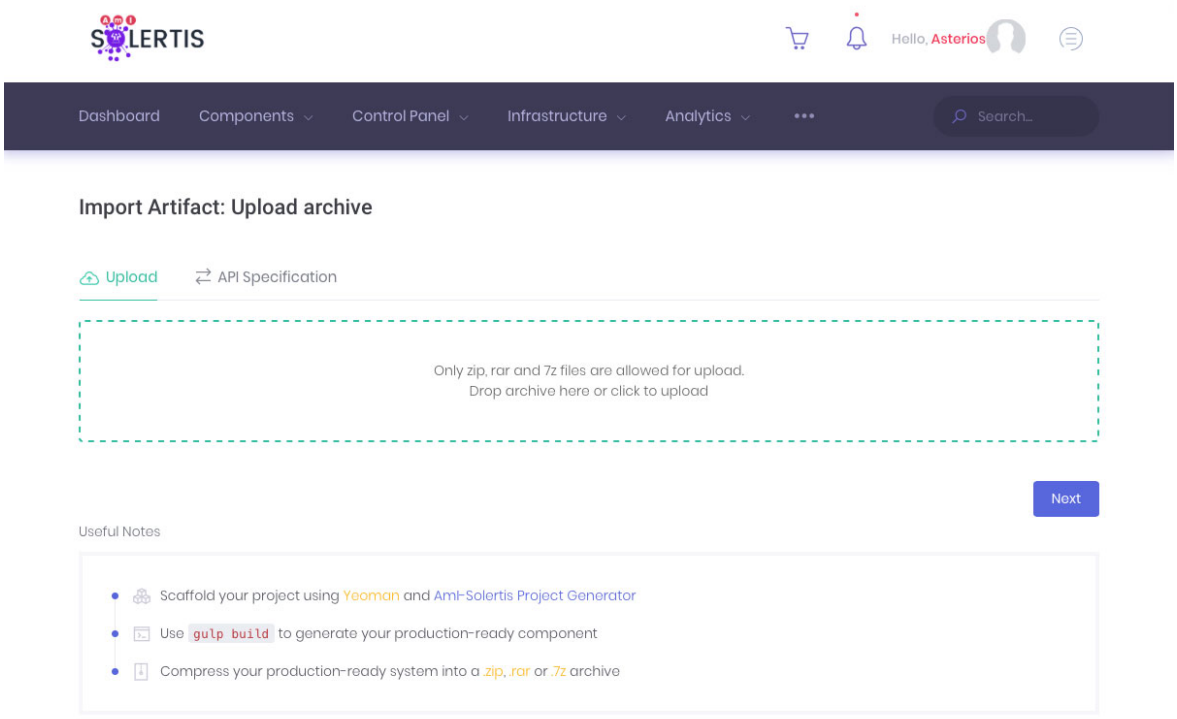

Figure 6.7: User-friendly service import process.

Upon successful receipt, the generation process starts. If the uploaded file is in a compatible project format, then the OAS is automatically extracted; otherwise, the user is asked to upload a valid OAS specification. Using the OAS specification the service importer pre-fills the various fields (e.g., operation names, URLs, parameter names and types) and asks the user to review and verify or modify them accordingly ((Figure. 6.8). If no specification is provided, the user is asked to manually complete all the required fields to ensure optimal integration.

As soon as this fine-tuning process is completed, the Service Importer instructs the back-end modules (i.e., Artifacts and Scripts Importer, Packaging component) to build the necessary proxies and software bridges (if needed) and create the respective software package. Upon completion, the new AmI artifact gets installed in the Universal Repository and gets published to the AmI-Solertis ecosystem.

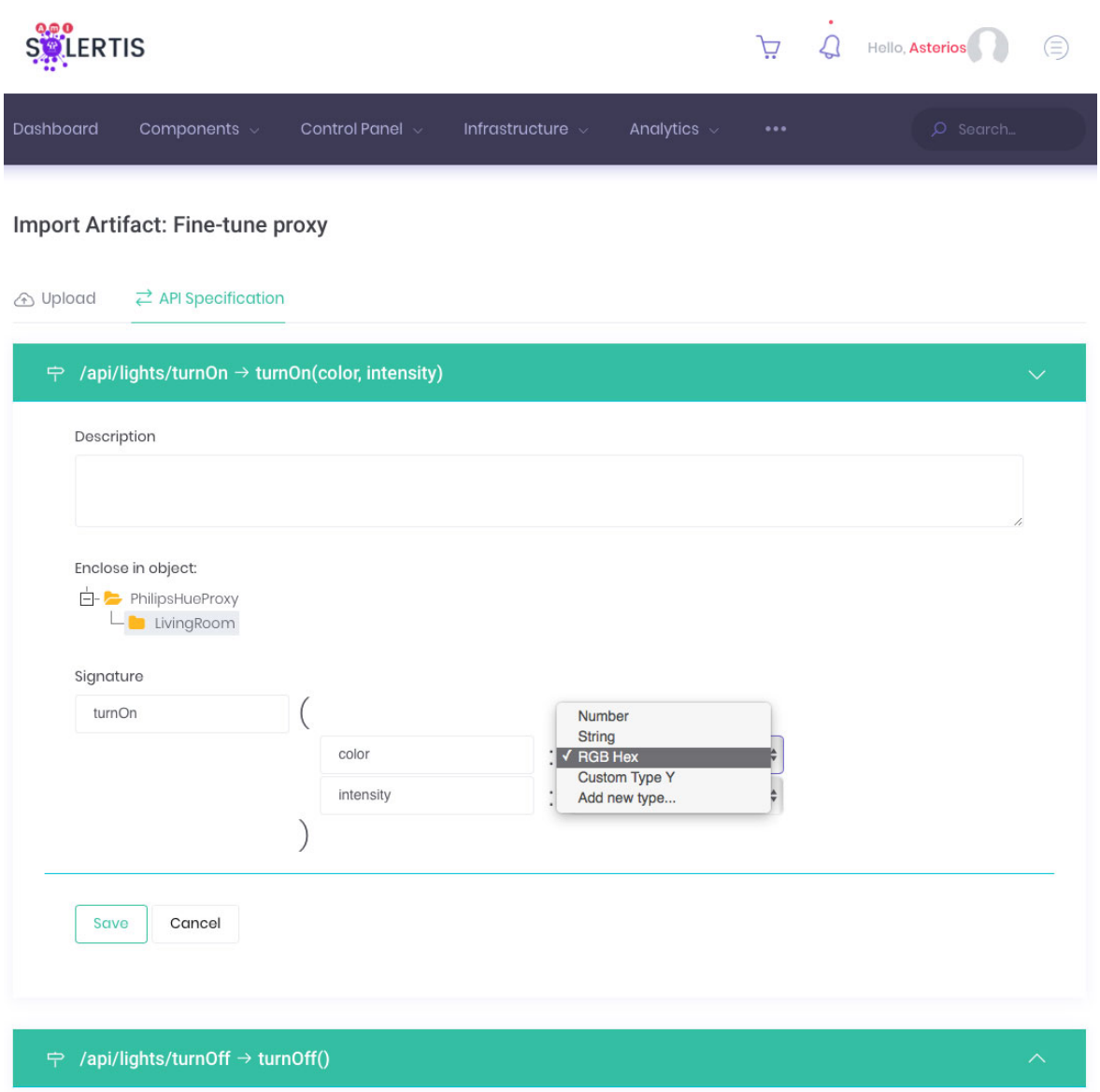

Figure 6.8: The fine-tuning process offered by the service importer.

6.3 The Code Editor

In recent years there is a trending norm of providing tools for professional over the Internet such as Document Processors (e.g., Google Documents), File Managers (e.g., DropBox), Design Tools (e.g., Figma) instead of requiring users to install them locally; even specific OSs (i.e., ChromeOS) have been introduced that target full-fledged computers (i.e., Google PixelBook) which execute most of

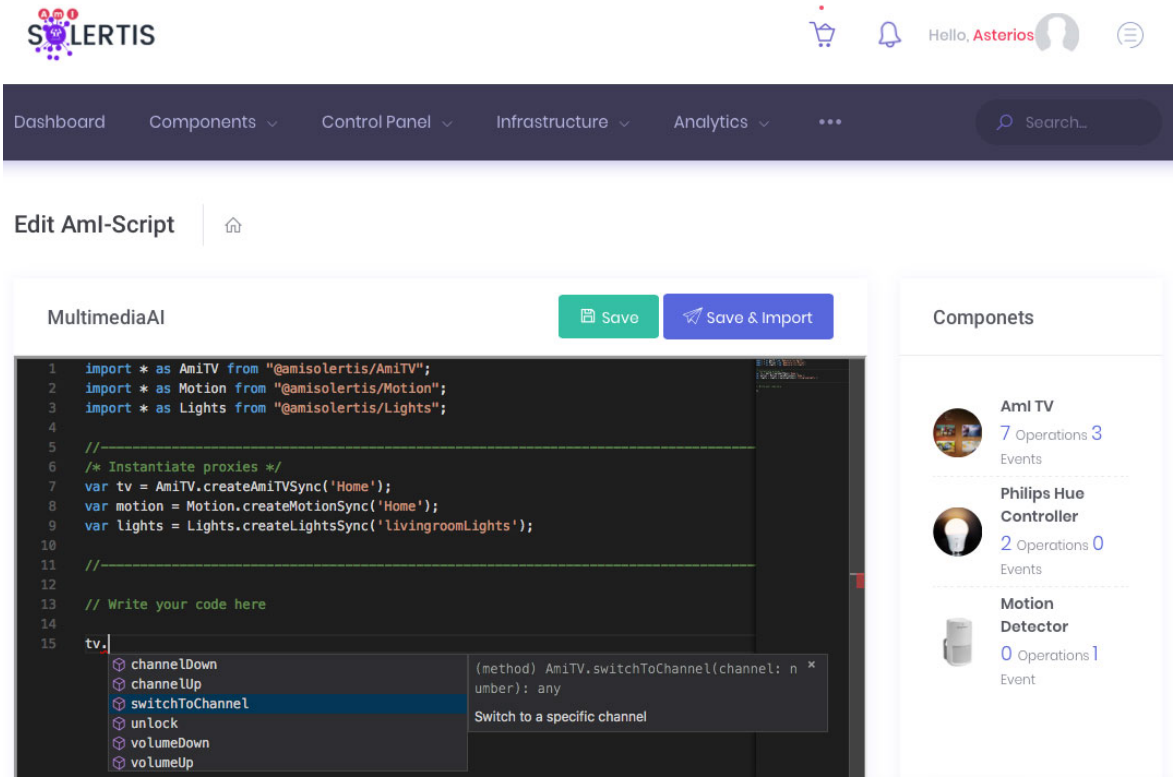

Figure 6.9: The AmI-Solertis Textual Code Editor

its tasks over the Internet, thus in practice converting them into very powerful web browsers. The benefit of providing such tools through the Internet is that they relieve developers from having to build their software using pre-configured computers, but instead they can fire up any modern browser from any device (e.g., computer, tablet, smartphone) and be able to continue with their work.

The AmI-Solertis Studio integrates both a text and a visual code editor to accommodate both novice and experienced developers. The textual editor (Figure 6.9) is built on Monaco editor [2] and offers advanced editing facilities such as “Go to Definition”, “Rename Symbol”, “Basic Syntax Colorization”, “Hints and Recommendations”. Most importantly though, it supports an extensible “Auto-completion” mechanism that AmI-Solertis populates based on the AmI-Solertis compliant services on which the current behavior scripts depends, and further enhances it with contextual suggestions regarding the expected values that are passed as arguments to proxies’ methods. To

[2] See https://microsoft.github.io/monaco-editor/

facilitate programming by novice developers, the visual editor (Figure 6.10) enables them to view and modify the existing scripts or create new. Users combine graphical blocks [132] that correspond either to basic programming structures (e.g., loops, variable definitions, arithmetic expressions, etc.) or functions stemming from service proxies, to define the script's logic.

Listing 6.1: An AmI script that was automatically generated by the code editor based on the collection of components that the user has added in her cart.

```
import * as HES from '@ami/HomeEntertainmentSystem';
import * as HOME from '@ami/HomeControl';

var contextName = '__PLACEHOLDER__';
// Instantiate proxies to access the remote AmI artifacts
var entertainment = await HES.createEntertainmentSystem(contextName);
var myHome = await HOME.createHomeControl(contextName);
class MusicFollowMe {
  constructor() {}
  async startUp() {
    // Register for events
    this.registerToEventFederator();
  }
  registerToEventFederator() {
    // add registration code here
  }

  // Write your business logic here

  // Sample of a function exposed to the AmI Ecosystem
  /**
  * What does this function do?
  * @export - /sampleURL
  * @param {String} arg - description => VALUES: [v1, v2]
  * @returns {String} description
  */
  sampleFunctionToBeExported(arg) {
    return 'value';
  }
}
module.exports = MusicFollowMe;
```

Event-based programming is supported by connecting the appropriate event callbacks [65] to the available hooks using the respective proxies' registration methods. To address the asynchronous nature of remote calls, AmI-Solertis behind the scenes statically analyzes the script and based on the user's experience either rewrites the remote calls in a synchronous manner using

the “await” keyword for novice programmers, or in case of experienced programmers it visually annotates the source code to target the user’s attention to that part of the code.

Both editors when launched are provided with the collection of AmI artifacts or scripts that the new AmI script will use, which allow them to further assist the users by automatically generating the necessary boilerplate code to use them (Listing 6.1). Based on the provided list, the editors import the relevant proxies, declare and initialize one instance for each, thus facilitating the users to focus on writing the business logic that makes use of these variables. Moreover, as already mentioned in chapter 4, users can annotate (using JSDoc in case of text editor or the equivalent control in the visual editor) that a method will be exposed to the overall AmI ecosystem and the editors automatically generate the necessary Express project to accomplish that.

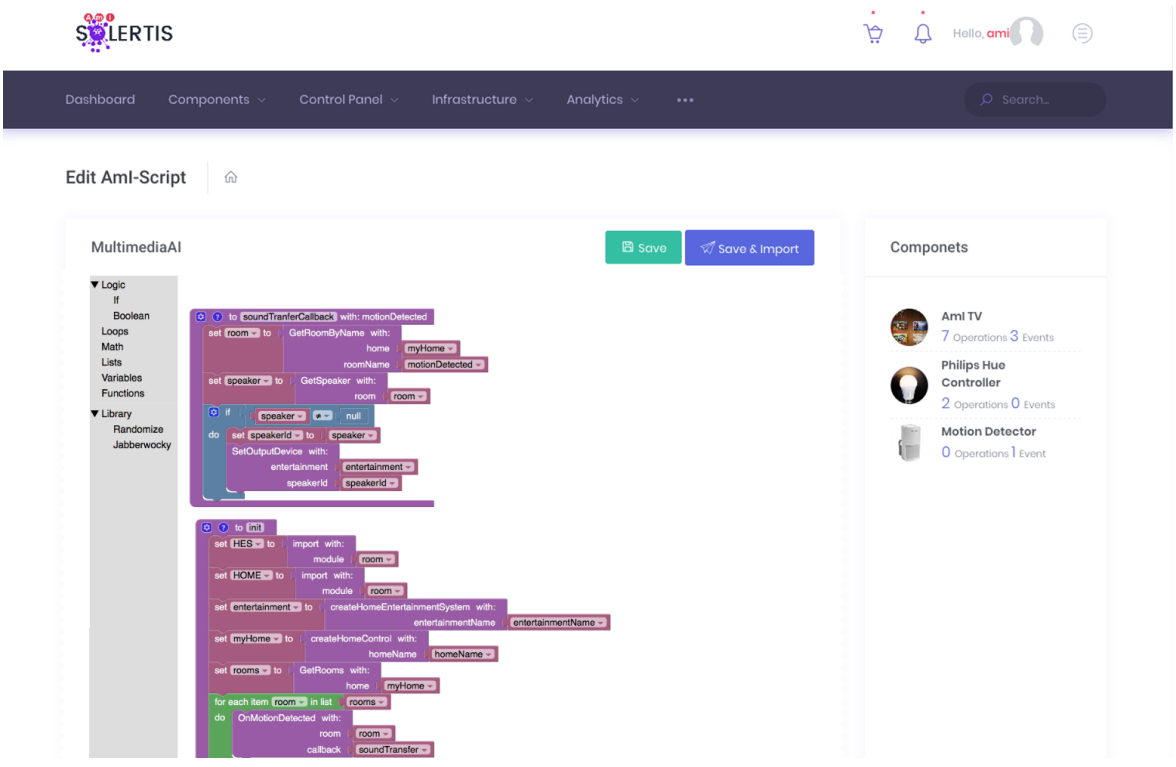

Figure 6.10: The AmI-Solertis Visual Code Editor

Finally, upon script creation, the user can bundle together the behavior script and its dependencies and use the Service Importer and the Deployer to execute it immediately on multiple hosts or schedule its later deployment. An illustrative example is depicted in Figure, which provides the textual and graphical equivalent of the same script, which instructs the Home Entertainment Sys-

tem to transfer the sound output to any speaker that is closer to the user.

6.4 The Control Panel

The Control Panel enable users to monitor and control the overall AmI environment, from a functional perspective in real-time. Cumulative statistics present the overall performance of the AmI ecosystem in a macroscopic scale (e.g., number of active components, total number of events sent, average Central Processing Unit (CPU) usage per host, host with really high usage) and facilitate high-level management. Nevertheless, the user can get detailed information about every aspect of the environment. Specifically, the user can see all the currently running components organized either by host or by component and for each one: (i) get a summary of its operational state in real-time (i.e., console output, logs, operation calls, events emitted) , and (ii) perform quick management action (e.g., stop, restart, query health, start an inactive deployed instance). The most important facility of this component though is the ability to compile custom monitoring consoles. In particular, a user can select a set of AmI components to create and save a custom view that will display information about their performace, real-time console output and logs, and a live trace of their various interactions (e.g., operation calls, event emission and handling) similar to a stack trace. That way the user can monitor certain execution scenarios to ensure that the AmI environment functions properly according to its specification and requirements.

6.5 The Infrastructure Manager

The Infrastructure Manager enables AmI-Solertis users to manage the environment from a hardware perspective (Figure 6.11). In particular, the user can manage all the registered AmI-Solertis compliant hosts or discover and install new ones; the local AmI-Solertis Host Manager instance enable this component to collect general information about every host (e.g., name, group, role, installed AmI components) and execute administrative tasks (i.e., restart, shutdown, delete). Moreover, through this interface the administrators (i.e., users with elevated privileges) of the overall AmI ecosystem can install new runtime environments that are currently not installed (e.g., JRE, Python), update the local Executor component (which is part of the local Host Manager) with the

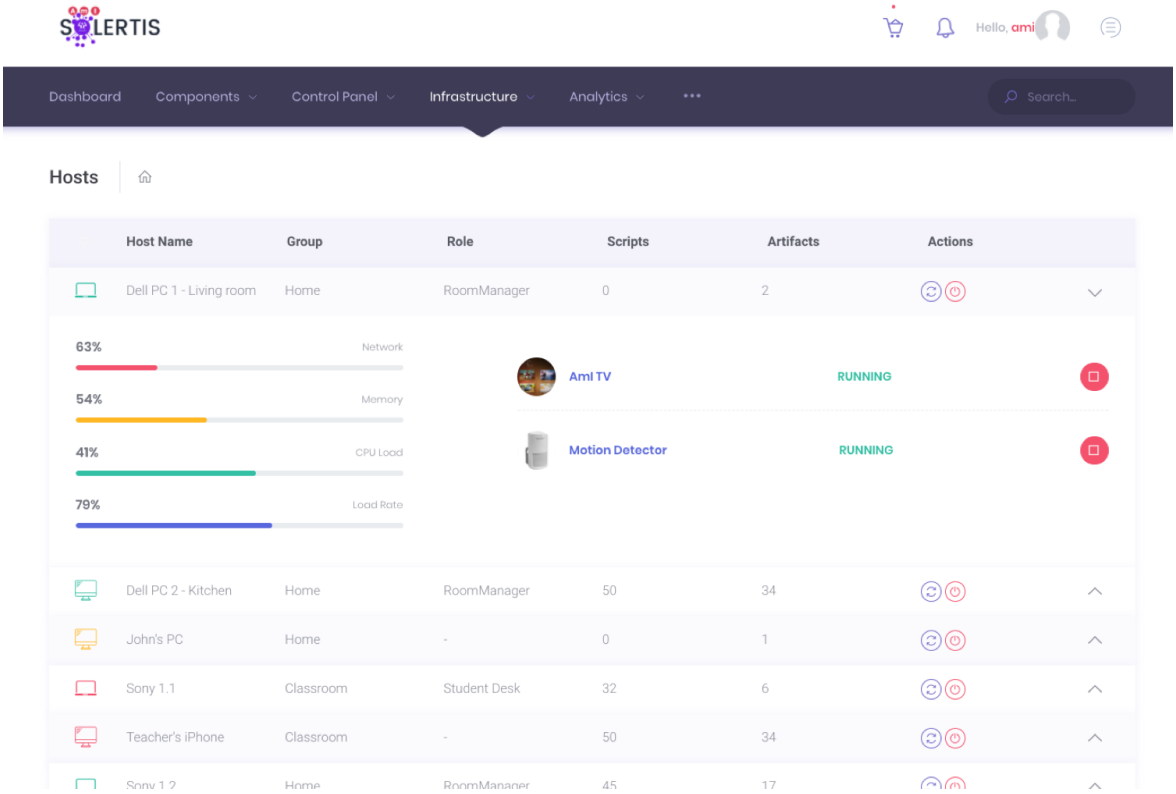

Figure 6.11: The AmI-Solertis Infrastructure Manager Frontend.

respective plugin and update the respective dependency rules. Via the Infrastructure Manager a administrator can also manage (i.e., create, view or modify) the various roles that are available in the ecosystem to simplify batch host management. Finally, this interface permits the creation of new application bundles that combine multiple software packages in a single "box" which streamlines the deployment of multiple AmI components.

6.6 Secondary facilities

Finally, AmI-Solertis Studio aims at assisting developers in creating their own full-fledged external applications or systems that integrate its exposed facilities. To that end, a dedicated "Downloads Area" permit developers to search the documentation or download the respective libraries (e.g., AmI-Solertis Mobile Proxy for iOS library, AmI-Solertis Code Editor, Web Explorer).

6.7 The AmI-Solertis virtual agent

The emerging paradigm of End-User Development (EUD), along with the rapid development of Intelligent Environments that intensively use IoT technologies suggest that in the near future end-users will have to be able to modify the behavior of the software artifacts they possess and the environments in which they live in [170, 223], in order to maximize their efficiency, extensibility and adaptation. Therefore, AmI systems need to be user-programmable. In fact, their programmers are not expected to be only professional developers, but also inexperienced end-users who can either modify the behavior of the system based on their current needs or extend its intelligence to address future necessities. The latter, in combination with the fact that programming such environments is inherently difficult due to their high architectural and computational complexity, further complicates the overall process.

Towards this objective, various alternatives have been proposed, with the visual programming paradigm being the prominent choice since it facilitates inexperienced users to quickly learn how to build simple programs [166]. Nevertheless, the rise of chatbots [340] and the fusion of conversational interfaces into Intelligent Environments enables the provision of more intuitive interaction paradigms (i.e., natural language dialogues) between users and intelligent virtual agents (i.e., technological artifacts). The AmI Solertis Studio empowers users to create behaviors scenarios by reviewing and modifying the high-level "business logic" of an intelligent environment in a user-friendly manner through both a textual and visual programming platform (as already mentioned), as well as an accompanying chat-bot agent (Figure 6.12).

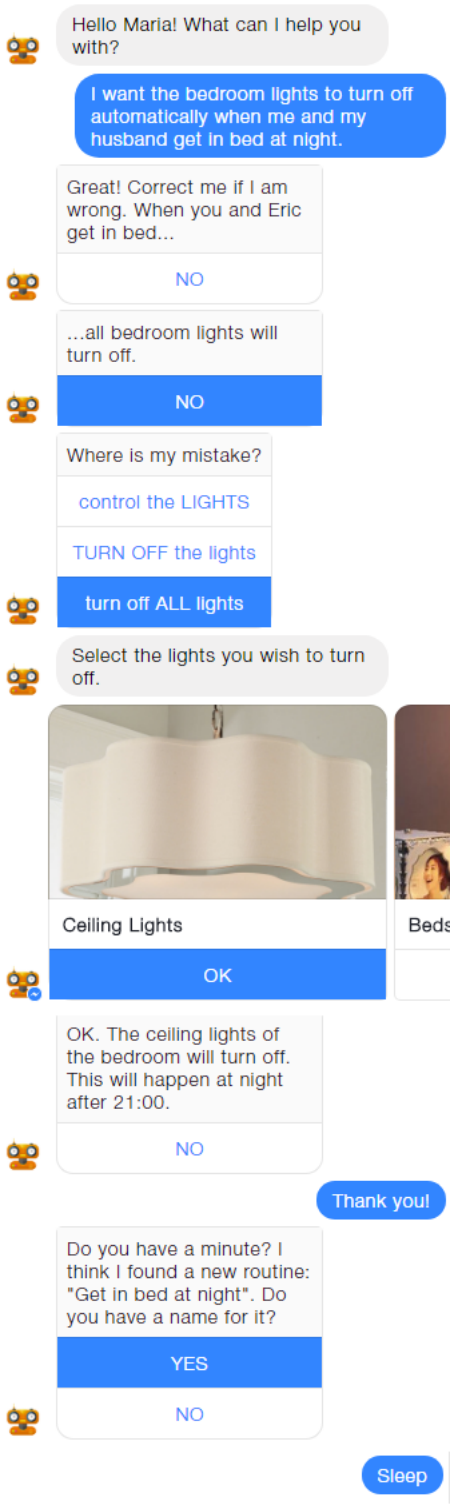

Figure 6.12: A sample conversation flow.

6.7.1 Implementation Details

AmI-Solertis offers a virtual agent in the form of a chat-bot, to facilitate management of the available technological artifacts of an Intelligent Environment by its occupants. AmI-Solertis chat-bot can communicate with the end-users via a natural language textual interface in order to help them accomplish numerous orchestration-related tasks (e.g., inquiries about services' status, definition of new behaviors, validation of existing ones).

AmI-Solertis chat-bot belongs to the category of conversational interfaces [416], as it constitutes a UI that mimics chatting with a real human. The underlying concept is that instead of communicating with a computer on its own terms by clicking on icons and entering syntax-specific commands, the user can naturally interact with it, by just telling it what to do. Nowadays, companies are interested in conversational interfaces as they are truly cross-platform. They work well everywhere, including smartphones, desktops, smartwatches, and even devices without screens at all (e.g., Amazon Echo™) [249]. Another interesting aspect of chatbots is that they make functions and commands immediately available, without the need to navigate in a menu.

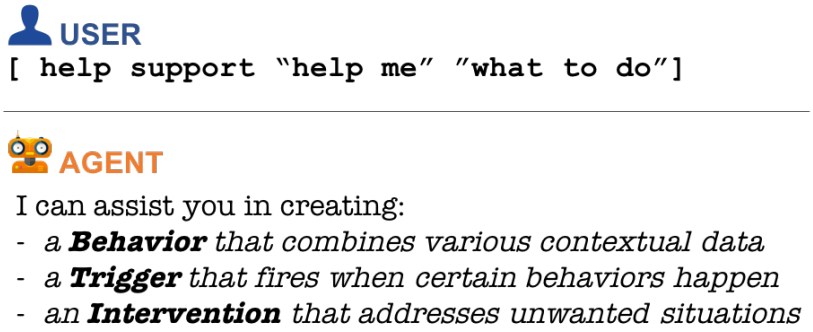

Figure 6.13: Dialog definition using Chatscript.

Chatbots have a certain preprocessing pipeline [100, 126, 185, 244] that they follow in order to progress with a conversation, which (almost always) includes the following stages: (i) **Spellcheck**: fix spelling errors, (ii) **Split into sentences**: enable analysis of every single one separately, (iii) **Split into words**: facilitate hardcoded rules that operate with words, (iv) **Part-of-Speech (POS) Tagging**: assigns parts of speech to each word, such as noun, verb, adjective, etc., (v) **Word lemmatization**: remove inflectional endings and return the base or dictionary form of a word, which is known as

the lemma, (vi) **Entity recognition**: identify dates, number, proper nouns, etc., and (vii) **Synomys/-Concepts identification**: properly annotate items that can be searched when applying a rule (e.g., starting point when booking a flight). After preprocessing is complete, the chatbot has a clean list of sentences and lists of words inside each sentence. Each word is lemmatized and marked with a part of speech. The next step is to identity the user intent by trying to match the defined patterns and proceed with the dialog by responding accordingly.

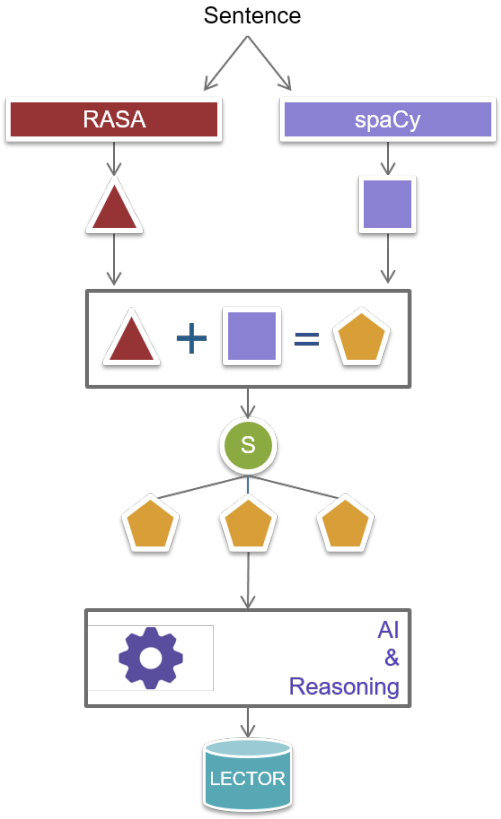

Figure 6.14: The AmI-Solertis chat-bot to understand what is the meaning of a sentence.

AmI-Solertis chat-bot is built using the **Chatscript** engine[3] [395], which is a pattern-based heuristics engine that uses a set of rules with pattern-based conditions. When AmI-Solertis chat-bot receives a message, it goes through all the patterns until finding a pattern that matches the user message; if a match is found, then AmI-Solertis chat-bot uses the corresponding template to generate a response. A sample pattern (defined using Chatscript's rule-based language) that aims

[3]https://github.com/bwilcox-1234/ChatScript

to identify a user intention to create a new AmI script is presented below in Figure 6.13.

ChatScript is able to keep track of dialogs and support long scripts which cover different topics. However, it does not run machine learning algorithms and by default does not access external knowledge pools or 3^{rd} party APIs. Therefore, AmI-Solertis chat-bot also uses the **spaCy** Natural Language Processing Library [4] and the RASA NLU[5] toolkit to apply machine learning for intent classification by utilizing information regarding the available AmI components (e.g., operations offered by the smart oven AmI artifact). As a result, a combination of patterns and machine learning classification algorithms help AmI-Solertis chat-bot to understand what is the meaning of the user message and generate the appropriate response (as depicted in Figure 6.14).

6.7.2 Context-Aware Interruption Handling

As expected, within such environments interruptions and task-switching while a human actor is engaged in a conversation with the chatbot agent (e.g., trying to deploy a new script that dictates the behavior of the intelligent environment) are quite frequent. In general, a great amount of research [265] has been conducted on how to handle interruptions that occur during human-agent conversations that originate from the human participant (e.g., the human explicitly or implicitly asks the agent to stop talking). However, in the context of this work, the term conversational interruption refers to the unexpected termination of the user's interaction with one of the intelligent agents due to external stimuli. The employed handling strategy is inspired by the HCI approaches that handle task switching (e.g., breadcrumbs to help the user identify at what step a multi-step process was stopped) [208] and the established guidelines on how to design mobile experiences for partial attention and interruption (e.g., data segmentation, glanceability) [166].

AmI-Solertis resides on contextual knowledge to smoothly resume the conversation. In principle, the key aspect is to assist the user via relevant visual information (i.e., context); in this case though, where conversation dialogues are used to build structured programs (i.e., AmI scripts), visual information is not available. Therefore, AmI-Solertis chat-bot artificially generates such information by combining data regarding the conversation itself (e.g., what the user was saying) with contextual information (e.g., what was happening in the surroundings and might be relevant).

[4] https://spacy.io
[5] http://rasa.ai

For that to be achieved, conversations are modelled via appropriately annotated utterances that facilitate natural language understanding [355] and hierarchically structured dialogs [41,396] that enable progress tracking; additionally, appropriate meta-models that store domain-specific conversation information have been designed and are used by AmI-Solertis chat-bot's Conversational Interruption Handler (CIH) to attach the contextual information that will offer mental cues during conversation resume.

Conversation Model

As aforementioned, in the context of this work, certain types of conversational dialogs exist that are directly mapped to the available back-end facilities of the AmI-Solertis framework, namely: (i) Activation and Deactivation Requests, (ii) Exploration Inquiries, (iii) Monitoring Inquiries, (iv) Recommendation Inquiries, (v) Creation and Modification Commands, and (vi) Help and Training Dialogs. Given that AmI-Solertis chat-bot supports certain user tasks (e.g., define a new behavior), every dialog corresponds to a micro Finite-State Machine (FSM) [344] and populates the respective data structures as the dialog with the user progresses and the FSM transits from one state to the next. An illustrative example of such a state machine is presented in Figure 6.15; the model of "Create a new behavior" is decomposed into its inner states and the three alternative dialogues are provided to demonstrate AmI-Solertis chat-bot's in-order (dialogue 3) and out-of-order (dialogues 1 and 2) data collection.

In addition to the FSM-specific meta-model that stores the behavior related parameters, the AmI-Solertis chat-bot's Dialog Manager, which orchestrates the overall process, stores any active conversations with their closure [212]. In particular, it persists the entire environment of a conversation including: (i) its complete stack of utterances of the current session, (ii) the identified user intents (i.e., what is the task that the user aims to accomplish via the current dialog), and (iii) any entities that have been successfully recognized and will be forwarded to the services of the AmI-Solertis framework to execute the actual task (e.g., instantiate and deploy a new script that encodes the desired behavior).

Modelling Contextual Knowledge and Interruptions

Context of Use is the cornerstone of Intelligent Environments, as it constitutes the glue that

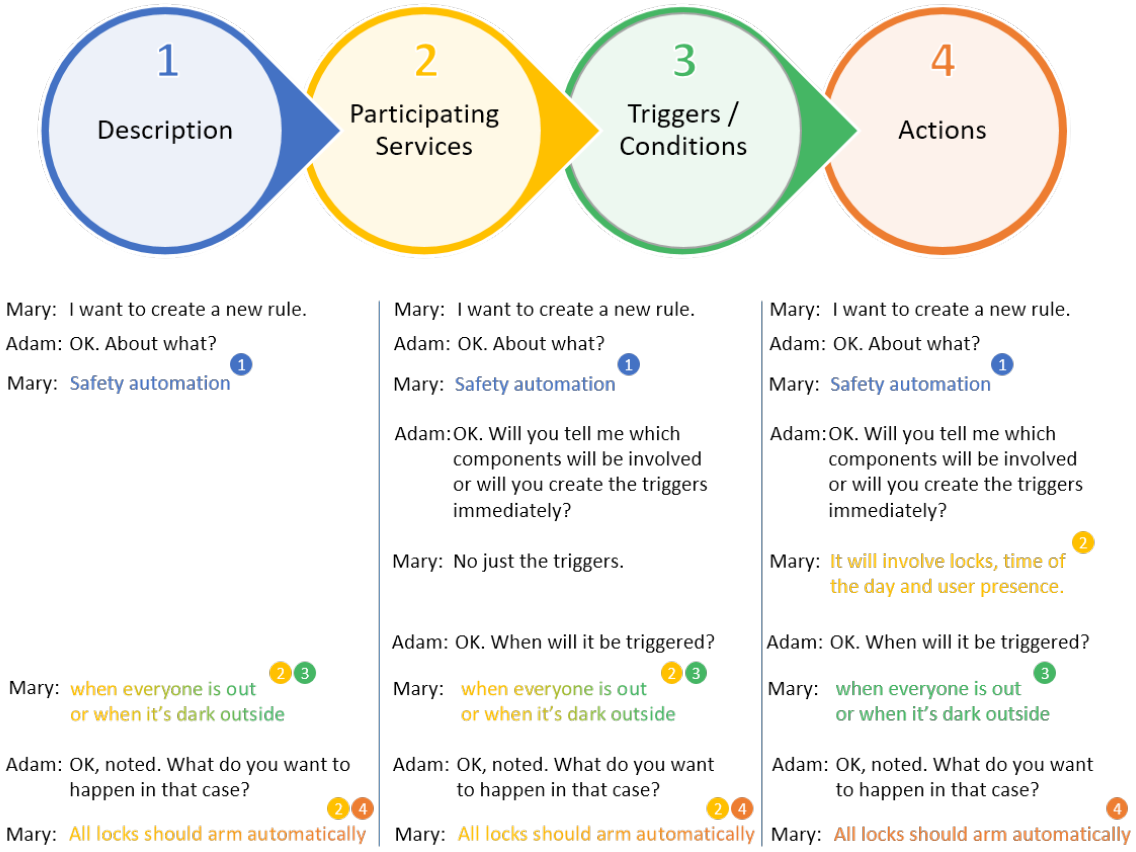

Figure 6.15: The state machine of the **Create a new rule** dialog and the out-of-order population of the mandatory fields of the relevant AmI-Solertis command.

brings together all the isolated services under a common roof and enables the environment's proactive response to user needs by providing intelligence sharing across the various services (including virtual agents). In such environments, various sensors and applications collect and process huge amount of information to distill and distribute useful insights in the form of small information chunks (e.g., users' location, activities at hand, state of interactive applications, etc.) that can be persisted and used by other agents. Such abundant contextual information is the key towards semantically annotating interruptions, as it contains background data that will be used

to provide mental cues to the user and assist conversation continuation.

As an example, consider the following scenario: Mary, while watching TV, notices the trailer of a new TV show that is about to air. The show seems interesting, so she decides to ask AmI-Solertis chat-bot to program its recording at the relevant time. At that moment, her friend Anna has decided to pay her a visit and is standing at the front door. As expected, the latter event is of higher priority, therefore interaction with AmI-Solertis chat-bot gets aborted. The dialog has not finished yet, however contextual information has been recorded (Figure 6.16); in particular, the Dialog Manager has stored that: (i) Mary has started telling AmI-Solertis chat-bot to create a new behavior, (ii) she was watching TV, (iii) the TV was on channel X, (iv) the program airing at that moment was Show-Z and (v) the trailer is about Show-W. In the future, when Mary is available, AmI-Solertis chat-bot can make a suggestion to resume their conversation and, if necessary, assist her by recalling what she was about to ask by retrieving and presenting that information.

Apparently, the amount of information available at any given moment in a fully connected intelligent environment is quite large; nevertheless, appropriate filtering techniques are used to determine which contextual information is useful to be attached to the interruption event (e.g., presence of another user, activities/applications capturing the user focus, etc.). The AmI-Solertis framework handles information storage using its internal logging mechanisms and exposes to AmI-Solertis chat-bot's Conversational Interruption Handler (CIH) only their MIME-types [55] and their reference points that facilitate their retrieval. Since AmI-Solertis chat-bot's behavior is encoded using the AmI-Solertis facilities, it can be easily extended via plugins (i.e., AmI scripts) to introduce new strategies that rely on contextual information that were not originally available; for example, in the context of the aforementioned scenario, if a TV channel can be queried to preview its content at that time through a 10-seconds video (including any commercials shown), then AmI-Solertis chat-bot could orchestrate its presentation to further assist Mary.

Resuming Conversations

Modelling by itself can be easily used by AmI-Solertis chat-bot's submodules to handle a single conversation that has been interrupted; but interruptions happen all the time in our daily environments and in many cases, they happen concurrently. The objective of employing ambient intelligence technologies is not just to limit unwanted interruptions, but mainly to minimize their

Figure 6.16: Filtered contextual information is attached to an interruption event.

handling and recovery time [186] benefitting from contextual information and collective intelligence.

Stack-based interruption handling

The approach followed by the AmI-Solertis framework regarding conversational interruption handling is inspired by the methodology followed by almost all compilers, to manage their run-time memory as a stack [13]; whenever a procedure is called, space for its local variables is pushed onto a stack, and when the procedure terminates, that space is popped off the stack.In more detail, whenever a new conversation starts, AmI-Solertis chat-bot's Dialog Manager creates a new activation record about that dialog, where the relevant references to the appropriate models are stored (Figure 6.17). In case of an interruption, the current dialog's contextual information is persisted, the overall dialog is marked as incomplete, and it gets pushed to the stack of dialogs for future reference. Additionally, while the user is occupied handling the unexpected event, AmI-Solertis chat-bot's Conversational Interruptions Handler attempts to evaluate the importance and the priority of the latest conversation and estimate the duration of the interruption, in order to determine

whether an intervention to resume the dialogue should be scheduled as soon as the user is available. For instance, if Mary has started saying *"Tonight, schedule lights to..."* or *"When my child approaches..."* and the interrupt originated from an expected event which is not supposed to take long (e.g., a courier delivering a package), then AmI-Solertis chat-bot will mark that last conversation as critical and will prompt Mary to resume at the earliest opportunity. In different cases, other appropriate closure strategies will be applied.

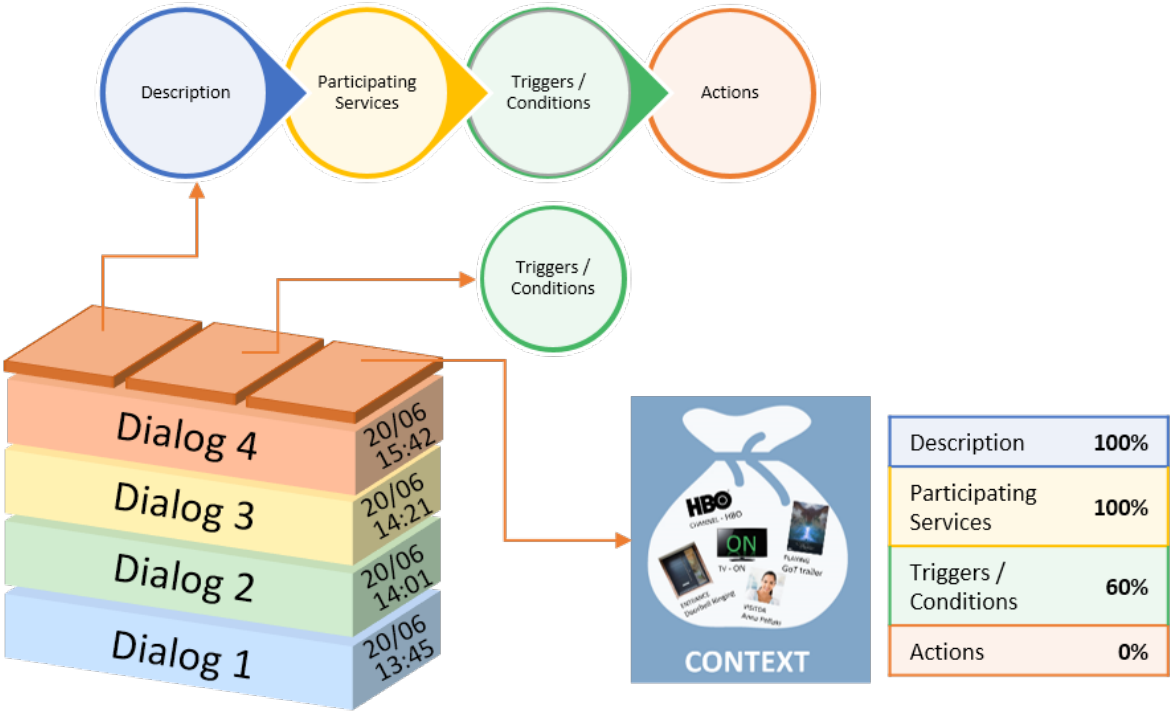

Figure 6.17: Snapshot of the Stack of Incomplete Conversations and a decomposed activation record.

Proactive Informative Interruptions. A conversation model in the context of this work refers to a single or a series of commands that will be submitted to the AmI-Solertis framework for execution. Every command specifies a set of mandatory fields (i.e., intents, entities) that must be populated using information extracted from the user's input. Therefore, if AmI-Solertis chat-bot

identifies that those values are not properly set before submission, it can proactively interrupt the conversation, by altering its natural flow, in order to get additional clarifications (e.g., user-specific jargon has been identified in a mandatory field instead of the actual service's name).

Interruption Avoidance or Fast-track. AmI-Solertis chat-bot, apart from artificially injecting interruptions in active conversations, aims to predict: (a) whether interruptions are about to happen and (b) how much time will their handling require, in order to determine how to avoid them. In case of a potential conflict, AmI-Solertis chat-bot either suggests to postpone the currently active conversation; e.g., if the user is about to get a notification saying that the cooking process needs his attention -a task that is expected to take 2 minutes to be completed-, whereas rule creation process takes 5 minutes, then AmI-Solertis chat-bot will ask for user's permission to schedule it for another time, or fast-track interruptions (e.g., asks the user to check the pot now rather than waiting for two minutes first) to ensure that the conversation will not stop from that point forward.

Conversation Abandonment. AmI-Solertis chat-bot periodically monitors the context of use in order to determine which pending conversations need to be discarded. To that end, AmI-Solertis chat-bot examines the participating services, triggers and actions of every partial activation record and evaluates whether an equivalent behavior has been already installed in AmI-Solertis from any other configuration channel (e.g., manually via the supported graphical editor, automatically by the AmI-Solertis self-assessment mechanism).

Conversation Continuation. Most importantly though, AmI-Solertis chat-bot's Interruption Handler aims to successfully complete any interrupted conversation. To that end, it monitors interaction and if a similar dialog begins or an analogous context is detected, it suggests to resume a relevant pending conversation. Priority is given to conversations that were recently interrupted (e.g., a few seconds ago), then to incomplete conversations that can be finished quickly (e.g., only the confirmation step is missing) and finally, to conversations that were classified as important or time-critical by the relevant AmI-Solertis facilities (e.g., involve sensitive family members, the user implied that the relevant behavior should start today). On resume, if the conversation was conducted some time ago, AmI-Solertis chat-bot provides a short summary to the user (e.g., *"You were saying about Safety Automation"*) to improve recollection. If necessary, AmI-Solertis chat-bot

can re-estate a more detailed context of use in usero'W•s working memory by restoring the information attached to that dialog; for instance, *"You were watching a commercial about Show-Z on Channel X, when you asked me to create a new rule to... Maybe you wanted to set a recurring notification to remind you about that show or record it if you are unavailable"*, or *"You were saying that when the night falls, for you family's safety, you want to lock..."*.

Chapter 7

Evaluation

This chapter reports evaluation findings that highlight the benefits of AmI-Solertis when used to define the intelligent behavior of AmI environments. In particular, firstly it provides the outcome of a preliminary conceptual evaluation that aimed to assess whether and how AmI-Solertis optimize the workflow that developers execute in order to perform different tasks to specify the behavior of an AmI environment. Next, it presents the findings two evaluation experiments, one expert- (i.e., Heuristic) and one user-based, regarding the usability of the implemented User Interfaces.

7.1 Conceptual Evaluation

AmI-Solertis is a multipurpose tool that facilitates the overall development process of software that facilitates the (i) integration new functionality to an AmI environment, or (ii) the control of the control the AmI facilities in an intelligent manner that satisfies user needs, or (iii) the combination of existing functionality to create a more complex "library" component; a library is a collection of implementations of behavior, written in terms of a language, that has a well-defined interface by which the behavior is invoked. For instance, developers who want to write a higher level program can use a library to make system calls instead of implementing those system calls over and over again; for instance, one can create a library for a single room that collects all relevant calls (e.g., KNX, Lights control, HVAC control) rather than using all these components in every script that somehow control the facilities of that particular room.

To the best of our knowledge, a single tool that offers similar functionality to the AmI-Solertis,

i.e., it developers in the whole development life-cycle of an AmI artifact or script, does not exist; hence, AmI-Solertis cannot be directly compared to other alternatives in a pure performance perspective. Consequently, this section will report a preliminary evaluation that aims to assess the AmI-Solertis concept and highlight the benefits of using it. To do so, a number of use cases in the context of controlling the behavior of a living room of a Smart Home will be described, through which the alternative workflows (i.e., with and without AmI-Solertis) will be presented.

Figure 7.1: The current living room setup of the Smart Home at the ICS-FORTH AmI Facility.

Objectives:

- Automatically adapt the physical and ambient light of the living room when watching a movie to match its “theme”.
- Lock the main entrance when night falls.

Facts:

- The services that control: (i) the multimedia center (named AmITV), (ii) the smart lock (named AmILock), and (iii) the blinds (named AmIBlinds) are already implemented, installed and properly configured in the AmI environment.
- The AmI environment contains two main computers that can host applications and scripts. The one is inside the cabinet under the TV and it is connected with it, whereas the other is located in the utility room of the house.

Use case- 1: Install a new AmI Artifact to control the light level.

Build, configure and deploy a program that will communicate with the official Philips Hue API[1] and control the smart lights of the room.

Table 7.1: Use case-1: Install a new AmI Artifact to control the light level.

Using AmI-Solertis	**Without AmI-Solertis**
S1.Scaffold a new empty project using the AmI-Solertis project generator.	S1.Create a new generic empty project using the platform/language of preference and manually customize it.
S2.Write the controller that will forward incoming requests to the official API.	S2.Write the controller that will forward incoming requests to the official API.
S3.Document (using JSDoc) the controllers' methods to guide the OAS extractor.	S3.Manually document a full OAS specification that describes the newly created service.
S4.Import the new service into the AmI ecosystem via AmI-Solertis Studio.	S4.Physically access the host to transfer the respective files.
S5.Configure and deploy the service to the desired host via a single-step process (i.e., AmI-Solertis Deployer).	S5.Install the new service by manually copying the files and manually editing any configuration files.
S6.Navigate to the host and start the new service via a single-step process (i.e., AmI-Solertis Executor).	S6.Launch the service using the respective OS- and platform- specific execution command.

As depicted in table 7.1, the major benefits in that simple scenario when using AmI-Solertis

[1] See https://www.developers.meethue.com/philips-hue-api

is the fact that many tasks are handled transparently by the system on behalf of the users (e.g., OAS generation, service deployment and execution). In particular, given that in both cases coding has to be done outside of AmI-Solertis, the differences mostly lay on the automation side; without doubt AmI-Solertis simplifies the overall process through its respective mechanisms compared to its the labor-intensive and error-prone alternative.

Use case- 2: Use the AmI artifact to control environment.

Dim the living room lights when watching a movie.

Table 7.2: Use case-2: Use the AmI artifact to control environment.

Using AmI-Solertis	**Without AmI-Solertis**
S1.Explore the corpus of available AmI components and add the desired ones (i.e., PhilipsLights, AmITV) to the cart.	S1.Discover the API of the AmITV service and explore the services that it offer and the events that it emits.
S2.Create a new AmI script that will contain the selected components by simply "checking-out" the cart.	S2.Create a new generic project and manually customize it.
S3.Using the automatically defined proxies connect the *"onMovieStart"* event of the AmITV proxy, with a function that calls the *"SetLightIntentityLevel"* of the PhilipsLights proxy with the value **25%** as parameter.	S3.Use any method to listen for and filter the events emitted by the AmITV artifact. When the right event is sent (i.e., *"MovieStart"*), use any method to directly make a remote HTTP request to the running PhilipsLights artifact at the right URL (i.e., that corresponds to the function *"SetLightIntentityLevel"*) and pass as a parameter an appropriately created object that contains the value **25%**.
S4.Save the new AmI script.	S4.Manually build the service's the executables.
S5.Deploy (with a single click) the new AmI script at the "HomeMainFrame" host.	S5.Physically access the "HomeMainFrame" host to transfer the respective files.
S6.Execute (again with a single click) the new AmI script.	S6.Manually copy the files and manually edit any configuration files.
	S7.Launch the service using the respective execution command.

Table 7.2 highlights that in addition to automating the overall workflow, the major benefit of using AmI-Solertis stems from the fact that a developer can create a new project that controls the behavior of an environment by simply selecting the components that she would like to use. Then using the editor and the respective proxies, the objective can be achieved with a few lines of code, as any boilerplate code (i.e, consume services or listen for their events) is handled automatically.

Use case- 3: Encapsulate room control in a single component.

Since the room includes various components that will be used in multiple scenarios, encapsulate these diverse functions in a Facade object [135] that will provide a unified interface to a set of service that control that technologically-enhanced room.

Table 7.3: Use case-3: Encapsulate room control in a single component.

Using AmI-Solertis	**Without AmI-Solertis**
S1.Explore the corpus of available AmI components and add PhilipsLights, AmITV, AmILock, AmIBlinds AmI artifacts to the cart.	S1.Discover the API of the AmITV, AmILock, AmIBlind and PhilipsLights service and identify their services and events.
S2."Check-out" to create a new project.	S2.Create a new generic project and manually customize it.
S3.Write and export (by appropriately annotating in JSDoc) the functions and events that will expose the room's functionality (e.g., the function "darken the room" will lower the blinds and turn-off all the lights).	S3.Write the controller that receives incoming requests and makes direct remote HTTP request to the respective service in order to accomplish the desired effect. To support event-based communication, implement an appropriate Pub/Sub mechanism to allow other components to express their interest to receive its events. Subscribe to events from any of the encapsulated components and upon reception, propagate that event to the subscribers of this script.
S4.Save the new AmI script.	S4.Manually document a full OAS specification that describes the newly created service.
S5.Deploy (with a single click) the new AmI script at the "HomeMainFrame" host.	S5.Physically access the host to transfer the respective files.
S6.Execute (again with a single click) the new AmI script.	S6.Install the new service by manually copying the files and manually editing any configuration files.
	S7.Launch the service using the respective OS- and platform- specific execution command.

As presented in table 7.3, building a new "service" with AmI-Solertis is as simple as creating a new module to control a specific behavior. The only difference is that in the first case the functions

or events that have to be exposed to the AmI ecosystem are annotated appropriately by appending the keywords **@export** or **@event** in their JSDoc. With respect to implementation per se, invoking a remote methods (i.e., turn-off the light X), is extremely simple, as it is accomplished via the respective proxy.

On the other hand, delivering the same functionality without AmI-Solertis requires developers to implement a new component from scratch and manually introduce it in the ecosystem. Moreover, the new component should lister for any events emitted by the component that it encapsulates and forward them to its own subscribers as well. Additionally, the calls to the various remote services are hard-coded, which implies that if the service migrates to a different server or fails, then a considerable amount of time is necessary to re-implement the new component so as to handle these failures and re-introduce to the AmI environment.

Use case- 4: Append a second AmI artifact in the above script.

In addition to the Philips Hue system, two sets of LIFX tiles [2] were also installed in the living which have to be controlled as well. The respective LIFXLights service that wraps the has been already incorporated in the AmI ecosystem.

Table 7.4: Use case-4: Append a second AmI artifact in the above script.

Using AmI-Solertis	**Without AmI-Solertis**
S1.Find the previously created component in the AmI Scripts list and select to edit it.	S1.Discover the API of the new LIFXLights service in order to identify its operations and events.
S2.On-the-fly add the new component in the AmI script. Then, modify the existing methods that control the room's ambient light to also manipulate the LIFX tiles using the newly created proxy.	S2.Modify the original source to update the respective control methods to directly communicate with the LIFX service in order to manipulate it appropriately.
S3.Save the modified AmI script.	S3.Manually build the service's the executables.
S4.Swap the running version with the new one.	S4.Physically access the host to transfer the respective files.
S5.Execute the updated room control AmI script.	S5.Replace the old service with the new one by manually overwriting the files and manually updating any configuration files.
	S6.Launch the new service using the respective OS- and platform- specific execution command.

Table 7.4 points out that the modification of any AmI script (independently of its running state) is very simple via the built-in editor. Developers only have to find it and then fire up the editor so as to modify its code. On the contrary, modifying a service that is maintained separately, requires from the developer to firstly gain access to the original source code, then modify it and re-introduce it in the AmI ecosystem. As in Use Case 3, the modified service will communicate directly with the LIFXLights service, hence limiting its flexibility and extensibility.

[2] https://eu.lifx.com/products/lifx-tile

Use case- 5: Implement the full scenario.

Build, configure and deploy a program that controls the ambient light of the room depending on the context of use (i.e., activities of its residents, current TV program) and secures the room at night.

Table 7.5: Use case-5: Implement the full scenario.

Using AmI-Solertis	**Without AmI-Solertis**
S1.Find and add to the cart the room controller AmI script which facilitates the manipulation of all the technological facilities of the room.	S1.Discover the API of the Room Control service in order to identify its operations and events.
S2."Check-out" the cart to create a new project.	S2.Create a new generic project and manually customize it.
S3.Via the built-in editor use the room's proxy to access the internal AmITV proxy and associate the *"onMovieStart"* event with a call to room's proxy function *"DarkenRoom"*. Moreover, schedule a second function to run every night at 21:00 and call the room's proxy function *"LockTheRoom"* to secure the place.	S3.Write the controller that uses some method to subscribe to the events emitted by the Room Control artifact. If the Room Control artifact does not wrap AmITV events, then the new controller should directly listen to the AmITV service. When the right event is sent (i.e., *"MovieStarted"*), use any method to directly make a remote HTTP request to the running Room Control artifact at the right URL (i.e., that corresponds to the function *"LockTheRoom"*). Moreover, schedule a second function to run every night at 21:00 and another direct to the call at the URL that corresponds to the function *"LockTheRoom"*).
S4.Save the new AmI script.	S4.Manually build the service's the executables.
S5.Deploy (with a single click) the new AmI script at the "HomeMainFrame" host.	S5.Manually copy the files and manually edit any configuration files.
S6.Execute (again with a single click) the new AmI script.	S6.Launch the service using the respective execution command.

Table 7.5 emphasizes the ability offered by the AmI-Solertis that enables wrapper components to expose -instead of redefine- the functionality of their internal components. For instance, in

the above use case, when using AmI-Solertis, the developer can access via the respective getter the proxy of the AmITV and register a handler to its event. On the contrary when building a new component from scratch, the developer is forced either to redefine all the events that its internal components emit, or ask its consumers to subscribe to directly its internal services and listen for their events, thus taking the wrapper out of the loop.

7.2 User-Interface Evaluation

7.2.1 User-based Interface Evaluation

The AmI-Solertis Studio has be assessed through a user-based evaluation experiment. During the experiment, great insights were drawn by observing the users interacting with the system and noting their comments and general opinion, while several usability errors were identified. In more detail, five (5) users of ages 25-30 years have participated in the experiment. According to Nielsen [269], 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 who have worked in Ambient Intelligence projects, and therefore familiar with the related concept and principles. Particularly, two (2) of them were junior developers, two (2) 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 AmI-Solertis framework, as well as the functionality offered through AmI-Solertis Studio, were explained to the users. This process is necessary, since the developers should get acquainted with various new concepts introduced by AmI-Solertis, such AmI artifact, AmI script, Host, etc., before being able to use the system. After the introduction of all necessary information, the users were requested to browse freely through AmI-Solertis Studio 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 has been performed the days following days 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 AmI-Solertis, while they were also encouraged to express their opinions and thoughts openly following the thinking-aloud protocol [270]. 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, and 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) [60] 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.

7.2.2 Evaluation Findings

Each user was requested to complete ten (10) tasks (Appendix C) covering the most important functionality of AmI Solertis. The findings of this process are described below categorized per system function.

Task- 1: View instructions about how to create an AmI Artifact

Evaluated through scenario task 2

Creating an AmI Artifact was the first thing that users had to do in order to complete Scenario Task 2. In more detail, the task asked them to write the code that controls a Philips Hue Lamp; to do so, a user should follow a set of instructions (i.e., execute terminal commands) that guide the scaffolding of a new AmI Solertis compliant project. After the project’s generation, the user should use an IDE or editor of her preference to simply add the lines of code required to control the lamp.

The AmI-Solertis studio featured a “Create” button that presented the guidelines to generate a project that can be used to create an AmI Artifact. However, this confused most of the users since by clicking on that button they expected to be redirected to a form or wizard within AmI-Solertis studio that guided the creation process, rather than do that online on their own computer.

Particularly, the usability issues identified were:

A. The create button is misleading to the user. The label should perhaps read “Instructions on how to create an AmI Script”.

B. After creating the AmI Artifact, a quick way to import it without leaving the instructions page should be provided.

User 1 (mid-level developer) suggested that, since a command line interface is needed to create an AmI script, the AmI-Solertis studio should incorporate either an online terminal emulator in which the necessary commands could be executed or, a “download” link that scaffolds behind the scenes a new project, compresses and returns it to the user.

Task- 2: Import an AmI Artifact to the system

Evaluated through scenario task 3

After creating an AmI Artifact the users had to import it to AmI-Solertis. This task was performed easily by the majority of the evaluators, while some minor usability issues where identified:

A. The wizard steps were not easily perceivable by the users.

B. The label “Import” of the button that finalizes the import process at the last step of the wizard might be confusing.

C. The expandable panels should be redesigned since the majority of the users did not understand that they were interactive.

D. The screen real-estate taken by each item should be minimized to facilitate overview.

Task- 3: Deploy an AmI Component on a host

Evaluated through scenario tasks 4 and 8

The majority of users deployed the component directly from the respective tiles in the list of components. This finding highlights the importance of including quick actions on the components' list, thus facilitating the developers' work by not compelling them to visit a component's page. All users completed the respective scenario tasks without facing difficulties; this is corroborated by the rapid execution time. Figure 7.2, displays the time in seconds that the users required to complete the scenario tasks 4 and 8. As it is shown, user 5 completed the fourth task in 140 secs,

however it should be noted that the user understood immediately what he had to do, but did not complete the task until he expressed his comments on the process.

The following usability issues were identified:

A. A deployment button should also be available from the host details page and the list of hosts.

B. Appropriate feedback should be provided upon deployment. The users should be informed whether the deployment was successful or not.

C. The system should offer visual feedback that the deployment was completed, guiding the user to look at the list of deployments.

D. The control for selecting the host (where a component will be deployed) should incorporate a filtering or search mechanism. For each host the user should be able to view rich information including current workload, any deployments of a previous version of the selected components, etc.

E. The system should suggest a context name combining the names of the component and/or the actual host name.

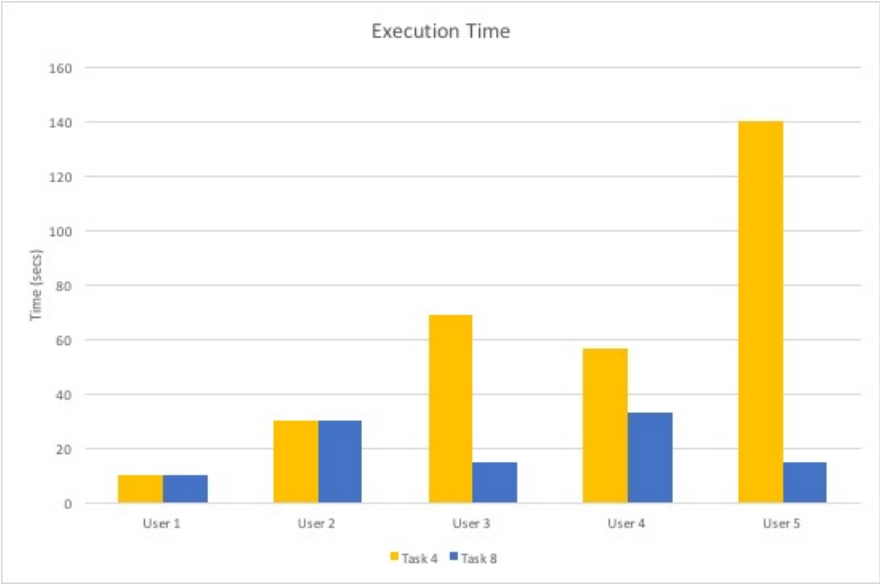

Figure 7.2: Execution time (in seconds) required to complete the scenario tasks 4 and 8.

Task- 4: Run an AmI Component on a host

Evaluated through scenario tasks 5 and 9

The respective scenario tasks were completed without any difficulties.

Task- 5: Browse Components

Evaluated through scenario tasks 2 - 8

Navigation through the components was performed easily by all users. The tile representation was particularly appreciated, since it offered an overview of the component details and it enabled the users to perform quick actions on them (i.e., add to cart, deploy). Despite the fact that the components list included a rich filtering mechanism to permit users narrow down the visible components according to specific criteria, one evaluator suggested that it would be beneficial to provide a categorization mechanism.

Another user pointed out that when creating or importing components, the system should offer visual feedback to help the user focus on the newly inserted item on the list. Finally, one usability issue was identified: there were not appropriate visual cues to inform the users regarding the items that have been already added to the cart.

Task- 6: View Details for a Component

Evaluated through scenario tasks 4, 5, 6, 8, 9 and 10

During the evaluation experiments the users had to visit various component pages so as to complete the requested tasks. During their interaction some minor usability issues emerged:

A. The position of the widget displaying the API is not optimal, making it difficult for the users to locate it. Some users expected to find it near the panel displaying the component's description.

B. A modal dialog is not appropriate for displaying a component's API, especially when there are many items to present.

C. The fact that a user can click on a row of the deployment's table to view the host's output console is not obvious, however some users stated that it is a useful feature.

D. When viewing the details of an AmI Artifact, personalized suggestions could be provided (e.g, other uses has also seen) based on the user's history.

E. When viewing the details of an AmI script, the system could display the AmI artifacts that are used in it.

Some useful suggestions made by the users were:

A. A structured log should also be available, in order to permit users explore the events happening in real time in the active hosts, rather than just viewing the textual output messages in the console.

B. The page presenting the details of an AmI Script should also display the components that the creator(s) utilized to build it.

Task- 7: Collect components to create an AmI Script

Evaluated through scenario task 7

Collecting the components to create an AmI Script was the task that troubled most of the users. Interestingly, none of them comprehended nor appreciated the metaphor of the shopping cart; furthermore, despite the fact that after receiving assistance from the facilitator all users added the items to the cart, no one visited it in order to reach the editor. Instead, they used the create button residing in the AmI Scripts list.

Some of the users who visited the editor without collecting items, saw the instructions provided by the system and understood the process that they had to follow. However, one user concentrated on the editor and could not understand what he had to do.

Despite the fact that with first time users the metaphor of the cart seemed inappropriate, this approach should be further examined given that the targeted developers will use the tool frequently. Furthermore, it should be investigated whether the cart metaphor is acceptable if presented differently to the users; for example, the cart icon could be replaced by a button (i.e., an icon and a label) reading "My Editor", while the "Add to cart" button could be replaced with an "Add to my editor".

A useful suggestion made by one of the evaluators, was that when the user selects to visit the components list from the respective buttons in the editor's page, the component tiles should con-

tain only the "Add to editor" functionality and no other quick actions (e.g., Deploy).

Task- 8: Create an AmI Script

Evaluated through scenario task 7

When asked to write the code of an AmI Script combining other AmI Components the users highly appreciated the functionality offered by the editor. Particularly, they liked the fact that it auto generated some of the required blocks of code, while they found the autocomplete feature really useful.

One minor usability error was identified regarding the "Save" and"Save & Import" buttons, which were available through the editor's page. Most of the users got confused since they were unaware that AmI Solertis studio enables them to save a draft of their script. This issue can easily be addressed by (i) using a different label for the respective button and (ii) adding appropriate indications to the tiles (Browse Components page) and the panel displaying the component's description (View Details page) of the draft AmI Scripts. Finally, one user (the experienced developer) asked whether a Deploy button could be added next to the two currently active ones, in order to simplify the overall process. His exact literal expression was: "Since I am now editing my script, why can't I deploy immediately, but I rather have to go back to the list to accomplish that".

Task- 9: View console output of a running AmI Script

Evaluated through scenario task 7

This task was completed successfully by four out of five evaluators. Interestingly, the remaining one was unable to answer the question asked (i.e., How many times has the **MultimediaAI** script **captured a "motion detected event"** and it instructed the **lamps to turn-on**). However, it must be examined whether this failure can be justified due to the evaluator's fatigue. That particular user was very talkative during the entire experiment, providing valuable feedback to the note-takers; however this energy-consuming behavior might have been the reason that lead him to be overwhelmed at the last task of the scenario. Finally, one user suggested that a structured log should also be available, in order to permit users explore the events happening in real time in the active hosts.

7.2.3 Discussion

As the findings suggests, the AmI-Solertis studio was well-received by the target user group (i.e., developers of software for AmI environments). All the users considered it to be overwhelming for the first use, nevertheless they all agreed that with frequent use, one gets easily familiarized with the entire concept. This observation is really important, since in practice the system aims to be used on a daily basis by developers that have been assigned specific tasks (e.g., creation of AmI script, artifact integration, system administration).

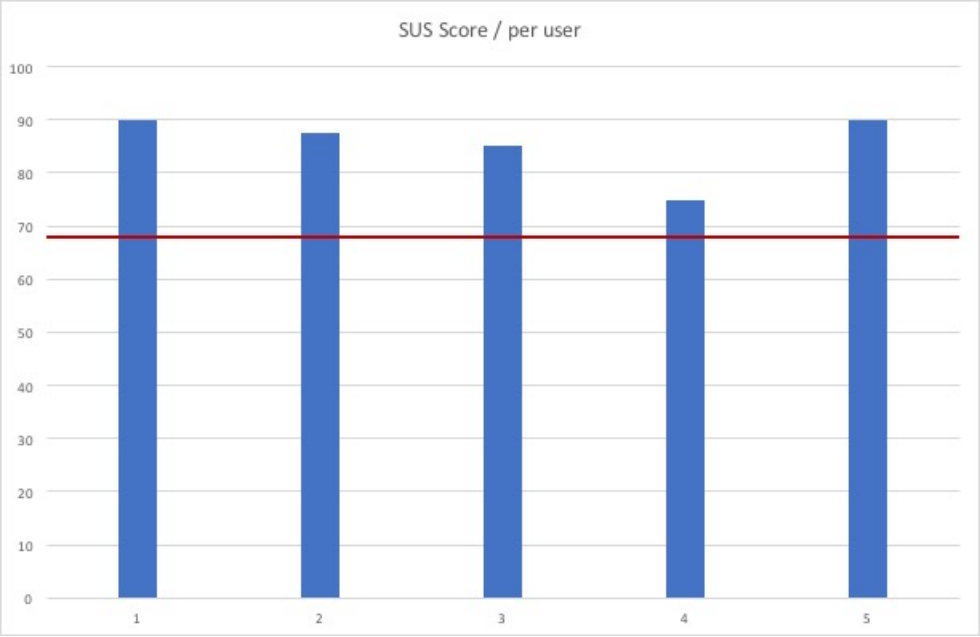

Figure 7.3: SUS score per user.

The overall opinion of the users, as extracted through the debriefing section was that the AmI Solertis studio is an intuitive tool, with a pleasant UI that they would definitely use for (i) exploring the available facilities of an AmI environment, (ii) importing new functionality either in the form of an AmI script or as an AmI artifact, and (iii) managing the overall environment (e.g., load--balancing, problem detection, performance deterioration). This is also corroborated by the SUS score (85) -as depicted in Figure 7.3- 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 7.4 shows how percentile ranks associate with SUS

scores and letter grades (from A+ to F) [330]. According to this graph, AmI Solertis studio's score (85) converts to a percentile rank of 98% and it can be interpreted as a grade of A.

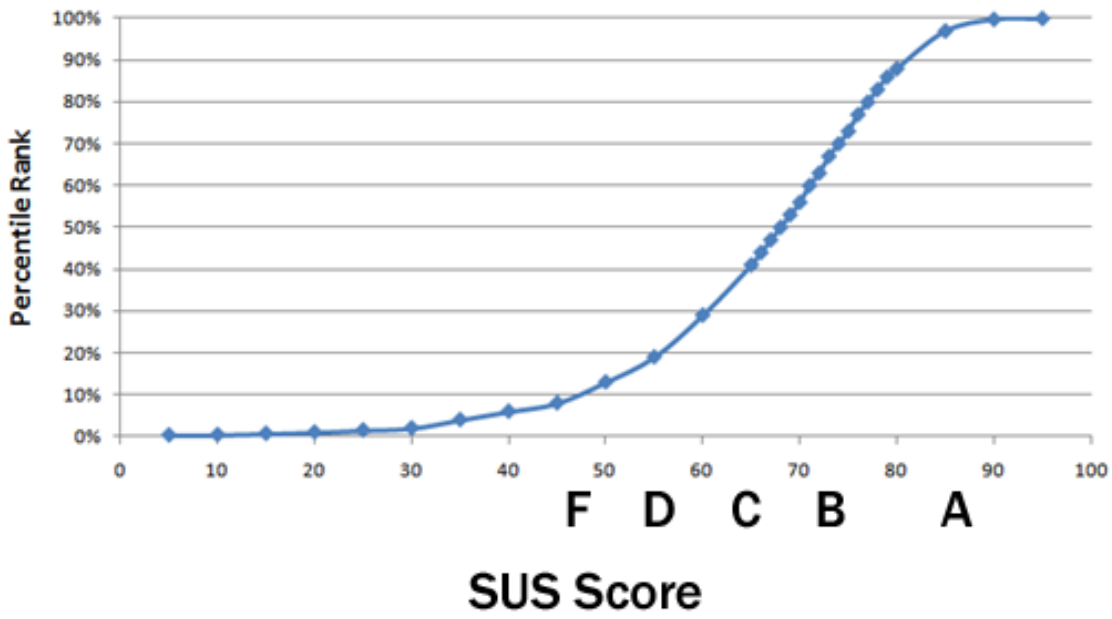

Figure 7.4: Percentile ranks associate with SUS scores and letter grades.

Despite the fact that the evaluation experiment focused mainly on the front-end, useful insights were also collected regarding the AmI Solertis framework itself. In principle, the participants consider the overall concept to be really useful and they think that it can benefit their day-to-day work. They liked the fact that they could combine the documentation of their code with the formal definition of their service, while they appreciated the ability to fine-tune the API at a later point in time. Some of the most notable user comments regarding AmI-Solertis and its Studio are presented below:

- "It is a very helpful tool especially for developers that have little experience in programming AmI environments" (user 1)
- "Very interesting tool. It can assist developers in building software for AmI environments and IoT" (user 1)
- "It is a nice and useful tool" (user 2)

- “With this tool, deploying components becomes very easy” (user 2)
- “Very interesting concept that amongst others facilitate sharing” (user 3)
- “It is very useful as it both centralizes access to various content, but also keep your things isolated” (user 3)
- “Very very nice. I like the Deploy and Run concept” (user 4)
- “I would not only use it myself, but I would recommend it to my colleagues as well” (user 4).

Chapter 8

Conclusions

8.1 Synopsis of Contributions

Ambient Intelligence (AmI) envisages that technology and information are flowing around the physical environment, objects are enhanced with computer technology to communicate, share information and collaborate with other technological devices in an intelligent fashion, thus formulating a ubiquitous and pervasive computing landscape, where implicit interaction and continuous cooperation is becoming the norm of computer supported activities. AmI forms a highly emerging market that is forecasted to exhibit exponential growth in the forthcoming years. Yet, AmI systems can only maximize their efficiency, extensibility and adaptation to the needs of their users, if they are programmable. Their programmers though, are not expected to be only professional developers, but also inexperienced end-users who can either modify the behavior of the system based on their current needs or extend its intelligence even further to address their future necessities.

To that end, this thesis has proposed the AmI-Solertis framework which empowers users to create behaviors scenarios by reviewing and modifying the "high-level business logic" of a smart environment through an advanced programming platform and an accompanying chat-bot agent. In addition to the framework, this work has introduced the AmI-Solertis Studio which offers a complete suite of tools allowing management, programming, testing and monitoring of all the individual artifacts (i.e., services, hardware modules, software components, etc.) of the overall AmI Environment.

In particular, AmI-Solertis addresses the following questions:

- Can a method unify the definition, introduction and management of new services that control devices and software components?
- Can a communication middleware also be real-time, scalable, cross-platform, rely on well-established technologies, and support multiple communication paradigms (i.e., synchronous, asynchronous and event-based)?
- Can a tool assist developers in the entire development life-cycle of an AmI system (i.e., analysis, design, implementation, testing, deployment, maintenance and disposal)?
- Can End-User Development (EUD) of AmI systems be accomplished with a user-friendly manner?

The AmI-Solertis system is built using a micro-service architecture style. A collection of loosely coupled services constitute its core, thus enabling it to be flexible and scalable to be used as a backbone and intelligent environments with diverse objectives. These autonomous components that interoperate over the network using a variety of communication protocols and channels; they expose their functionality via a public REST API in the form of stateless operations through which it can to receive incoming calls, while they can communicate their intention to the AmI ecosystem by emitting appropriate events via the AmI-Solertis Event Federator.

To further enhance interoperability and discoverability, OAS [5] is used to formally describe every API. AmI-Solertis encapsulates the complexity of configuring and performing remote calls in automatically generated proxies that ease the difficulties of distributed programming by (i) masking remote services and making them to appear as if they were local methods, (ii) enabling consumers to register their interest to events coming from a remote component without specifying any details about the underlying topology and infrastructure, and (iii) empowering AmI-Solertis to dynamically adapt and adjust the invocation process in order to address emerging requirements (e.g., re-routing a call).

AmI-Solertis enables fast, easy and error-free integration of external AmI artifacts (i.e., services) independently of their kind (e.g., pure web services, frontend applications that follow the SaaS paradigm). Moreover, it supports the creation of AmI scripts that define the behavior of

the technological facilities (i.e., business logic) towards creating pervasive, intelligent and personalized environment experiences by combining multiple components. In addition to AmI components management (i.e., AmI artifacts or scripts), AmI-Solertis also assists developers when building their AmI scripts or integrating their AmI artifacts in the ecosystem. Particularly, it automatically: (i) *generates* the necessary *auxiliary files* (e.g., service formal specification, configuration file template, service-specific methods and types definitions) (ii) *prepares the executable* files, (iii) creates the *installation packages*, (iv) *deploys* it on the designated AmI-Solertis compliant hosts, (v) orchestrates the *execution process*, and (vi) facilitates *real-time monitoring and control*.

Summing up, from an engineering perspective the AmI-Solertis framework: (i) introduces a unified **Hybrid Communication protocol** which supports synchronous, asynchronous and event-based communication, (ii) **unifies the definition and introduction** of new devices, services and software components, (iii) facilitates the **integration and usage** of heterogenous services in a standardized -yet agnostic- manner, (iv) delivers a **scripting mechanism** that can dynamically adapt the execution flow and govern the entire AmI environment'As behavior, and (v) offers a **standard library with tools** (i.e., Analytics and History, Fault Tolerance, Storage Management, Common Utilities) that developer can use.

From a user perspective, this thesis has introduced the AmI-Solertis Studio, which constitutes a web-based Integrated Development Environment (IDE) and Control Center that can be used as a creative tool for designing user experiences in Intelligent Environments. In particular, the studio: (i) supports the **entire development life-cycle** of an AmI system, (ii) **empower users** to explore and adapt software -through **user-friendly scripting environment**- to their personal needs or develop new innovative applications, (iii) simplifies service **discovery, definition, and management**, (iv) **scaffolds** typical designs. (v) offers **multiple visual representations**, (vi) provides **testing** facilities, (vii) facilitate **collaboration** between users, (viii) assists **real-time management** of the AmI environment, and (ix) delivers **AmI-Solertis virtual agent in the form of a chat-bot**, that can communicate with the end-users via a natural language textual interface in order to help them accomplish numerous orchestration-related tasks.

Finally, the AmI-Solertis Studio has be assessed through a user-based evaluation experiment that revealed a few minor issues, but in general confirmed its usability -a fact also corroborated by its achieved SUS score (85/100) which marks the tool as highly usable. The overall opinion of

the users was that the AmI Solertis studio is an intuitive tool, with a pleasant UI that they would definitely use for (i) exploring the available facilities of an AmI environment, (ii) importing new functionality either in the form of an AmI script or as an AmI artifact, and (iii) managing the overall environment (e.g., load-balancing, problem detection, performance deterioration). Moreover, the AmI Solertis framework itself was considered to be really useful and the participants mentioned that it can definitely benefit their day-to-day work.

8.2 Directions for Future Work and Research

There are several aspects that are worth further work and research in the context of the AmI-Solertis framework and the respective authoring studio. First of all, given that AmI-Solertis is expected to be extensively employed in building various AmI scenarios in-vitro in the context of Ambient Intelligence Programme of FORTH-ICS (i.e., Smart Home, Smart Classroom, Smart Greenhouse), the identified usability issues will be addressed first, yet it is foreseen that various updates and refinements of the Control Panel, Execution Tracing and Health monitoring facilities will emerge from that process and the accumulated hands-on experience.

Moreover, currently project management support is rather limited and should be further extended to support developers in building more complex systems; advanced integration of a versioning system could also contribute by aiding the creation of evolutionary services whose improvements could be properly tracked. Another potential improvement could be the deeper integration of AI and Natural Language Processing (NLP) to: (i) enable semantic service mapping for expert developers, (ii) offer personalized recommendation, (iii) empower dynamic service composition or recomposition based on various criteria (e.g., availability, QoS, compatibility), and (iv) facilitate conflict detection (i.e., the same event could trigger multiple contradictory behaviors).

Additionally, from a functional perspective, the AmI-Solertis framework should simplify the introduction of new runtime environments and deployment techniques (i.e., Docker containerization) and deliver a continuous testing and verification facility that will increase the reliability of an AmI environment by purposefully creating major failures or executing relevant verification scenarios on a regular basis. From an authoring perspective, the AmI-Solertis Studio could integrate interactive tutorials for novice users on how to introduce new or use existing AmI components to

specify the behavior of the environment. Besides the currently available programming capabilities, innovative immersive development paradigms (programming via games, AR-based or Mixed reality approaches) could further assist the end-users of such environments to build the behavior that fits their needs and preferences. Finally, tool-wise, AmI-Solertis could: (i) integrate security and authentication techniques (e.g., OAuth) to ensure that only authenticated components can use the available facilities, (ii) support other popular language by generating AmI component proxies that could be used by them, and (iii) create plugins or extensions for popular programming environments (e.g., Microsoft Visual Studio Code) through which developers interact with the AmI-Solertis framework from their preferred working environment.

Bibliography

[1] Apple App Store: number of available apps 2017 | Statistic.

[2] Google Play Store: number of apps 2009-2017 | Statistic.

[3] Hughes j (2011) iphone and ipad apps marketing: Secrets to selling your iphone and ipad apps. que publishing.

[4] Nielsen j (2001) converting search into navigation. http://www.nngroup.com/articles/search-navigation/. retrieved october 6, 2014.

[5] Swagger (OpenAPI) Specification.

[6] AARTS, E., AND GROTENHUIS, F. Ambient intelligence 2.0: Towards synergetic prosperity. *Ambient Intelligence* (2009), 1–13.

[7] AARTS, E., AND WICHERT, R. Ambient intelligence. In *Technology Guide*. Springer, 2009, pp. 244–249.

[8] AAZAM, M., AND HUH, E.-N. Fog Computing and Smart Gateway Based Communication for Cloud of Things. IEEE, pp. 464–470.

[9] ABERER, K., HAUSWIRTH, M., AND SALEHI, A. Infrastructure for Data Processing in Large-Scale Interconnected Sensor Networks. Conference and Custom Publishing, pp. 198–205.

[10] ACAMPORA, G., COOK, D. J., RASHIDI, P., AND VASILAKOS, A. V. A survey on ambient intelligence in healthcare. *Proceedings of the IEEE 101*, 12 (2013), 2470–2494.

[11] ADAMS, R., AND GRANIC, A. Creating smart and accessible ubiquitous knowledge environments. *Universal Access in Human-Computer Interaction. Ambient Interaction* (2007), 3–12.

[12] AGHAJAN, H., AUGUSTO, J. C., AND DELGADO, R. L.-C. *Human-centric interfaces for ambient intelligence*. Academic Press, 2009.

[13] AHO, A. V., SETHI, R., AND ULLMAN, J. D. *Compilers: principles, techniques, and tools*, vol. 2. Addison-wesley Reading, 2007.

[14] AILISTO, H., KOTILA, A., AND STROMMER, E. Ubicom applications and technologies. *VTT TIEDOTTEITA* (2003).

[15] AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., ALEDHARI, M., AND AYYASH, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials 17*, 4 (2015), 2347–2376.

[16] ALI, Z. Rfid based smart shopping and billing.

[17] ALOI, G., CALICIURI, G., FORTINO, G., GRAVINA, R., PACE, P., RUSSO, W., AND SAVAGLIO, C. A mobile multi-technology gateway to enable iot interoperability. In *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on* (2016), IEEE, pp. 259–264.

[18] ALOI, G., CALICIURI, G., FORTINO, G., GRAVINA, R., PACE, P., RUSSO, W., AND SAVAGLIO, C. Enabling iot interoperability through opportunistic smartphone-based mobile gateways. *Journal of Network and Computer Applications 81* (2017), 74–84.

[19] AMARAL, L. A., TIBURSKI, R. T., DE MATOS, E., AND HESSEL, F. Cooperative middleware platform as a service for internet of things applications. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (2015), ACM, pp. 488–493.

[20] AMES, M., AND NAAMAN, M. Why we tag: Motivations for annotation in mobile and online media, 2007.

[21] ANASTASOPOULOS, M., KLUS, H., KOCH, J., NIEBUHR, D., AND WERKMAN, E. Doami-a middleware platform facilitating (re-) configuration in ubiquitous systems. In *System Support for Ubiquitous Computing Workshop. At the 8th Annual Conference on Ubiquitous Computing (Ubicomp 2006)* (2006).

[22] ANTONA, M., MARGETIS, G., NTOA, S., LEONIDIS, A., KOROZI, M., PAPAROULIS, G., AND STEPHANIDIS, C. Ambient intelligence in the classroom: an augmented school desk, 2010.

[23] APPLE. Apple (2017) Apple special event September 12th 2017, release of the iPhone X., Sept. 2017.

[24] APPLE INC. Local and Remote Notification Programming Guide: APNs Overview, 2017.

[25] ASÍN, A., AND GASCÓN, D. 50 sensor applications for a smarter world: Libelium white paper. *Libelium* (2012).

[26] AUGUSTO, J., AND SHAPIRO, D. Ambiance: A mobile agent platform for end-user programmable ambient systems. *Advances in Ambient Intelligence 164* (2007), 81.

[27] BAGCI, F., PETZOLD, J., TRUMLER, W., AND UNGERER, T. Ubiquitous mobile agent system in a p2p-network. In *UbiSys-Workshop at the Fifth Annual Conference on Ubiquitous Computing* (2003), Citeseer, pp. 118–130.

[28] BAHETI, R., AND GILL, H. Cyber-physical systems. *The impact of control technology 12* (2011), 161–166.

[29] BAL, B., AND DUREJA, G. Hawk Eye: A Logical Innovative Technology Use in Sports for Effective Decision Making. *Sport Science Review XXI*, 1-2 (Jan. 2012).

[30] BALLIAUW, M., AND DECOSTER, X. Automated delivery. In *Pro NuGet*. Springer, 2013, pp. 179–214.

[31] BANDINI, S., AND SIMONE, C. Eud as integration of components off-the-shelf: the role of software professionals knowledge artifacts. *End user development* (2006), 347–369.

[32] BANDYOPADHYAY, D., AND SEN, J. Internet of things: Applications and challenges in technology and standardization. *Wireless Personal Communications 58*, 1 (2011), 49–69.

[33] BANDYOPADHYAY, S., SENGUPTA, M., MAITI, S., AND DUTTA, S. A survey of middleware for internet of things. In *Recent trends in wireless and mobile networks*. Springer, 2011, pp. 288–296.

[34] BANERJEE, P., FRIEDRICH, R., BASH, C., GOLDSACK, P., HUBERMAN, B., MANLEY, J., PATEL, C., RANGANATHAN, P., AND VEITCH, A. Everything as a Service: Powering the New Information Economy. *Computer 44*, 3 (Mar. 2011), 36–43.

[35] BARBERO, C., DAL ZOVO, P., AND GOBBI, B. A flexible context aware reasoning approach for iot applications. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on* (2011), vol. 1, IEEE, pp. 266–275.

[36] BARBOSA, R., NUNES, D., FIGUEIRA, A., AGUIAR, H., SILVA, J. S., GONZALEZ, F., HERRERA, C., AND SINCHE, S. An architecture for emotional smartphones in internet of things. In *Ecuador Technical Chapters Meeting (ETCM), IEEE* (2016), vol. 1, IEEE, pp. 1–5.

[37] BARHAM, P., ISAACS, R., MORTIER, R., AND NARAYANAN, D. Magpie: Online modelling and performance-aware systems. In *HotOS* (2003), pp. 85–90.

[38] BARRICELLI, B. R., AND VALTOLINA, S. Designing for end-user development in the internet of things. In *International Symposium on End User Development* (2015), Springer, pp. 9–24.

[39] BASS, L., WEBER, I., AND ZHU, L. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[40] BATTY, M., AXHAUSEN, K. W., GIANNOTTI, F., POZDNOUKHOV, A., BAZZANI, A., WACHOWICZ, M., OUZOUNIS, G., AND PORTUGALI, Y. Smart cities of the future. *The European Physical Journal Special Topics 214*, 1 (2012), 481–518.

[41] BELLEGARDA, J. R. Spoken language understanding for natural interaction: The siri experience. In *Natural Interaction with Robots, Knowbots and Smartphones*. Springer, 2014, pp. 3–14.

[42] BELLUCCI, A., VIANELLO, A., FLORACK, Y., AND JACUCCI, G. Supporting the serendipitous use of domestic technologies. *IEEE Pervasive Computing 15*, 2 (2016), 16–25.

[43] BEN-KIKI, O., EVANS, C., AND INGERSON, B. Yaml ain't markup language (yaml) version 1.1. *yaml. org, Tech. Rep* (2005).

[44] BETTS, D., DOMINGUEZ, J., DE LAHITTE, H., MELNIK, G., SIMONAZZI, F., AND SUBRAMANIAN, M. Developer's guide to microsoft enterprise library.

[45] BIBRI, S. E. Ambient Intelligence: A New Computing Paradigm and a Vision of a Next Wave in ICT. In *The Human Face of Ambient Intelligence*, vol. 9. Atlantis Press, Paris, 2015, pp. 23–66. DOI: 10.2991/978-94-6239-130-7_2.

[46] BIKAKIS, A., PATKOS, T., ANTONIOU, G., AND PLEXOUSAKIS, D. A survey of semantics-based approaches for context reasoning in ambient intelligence. In *European Conference on Ambient Intelligence* (2007), Springer, pp. 14–23.

[47] BIRLIRAKI, C., MARGETIS, G., PATSIOURAS, N., DROSSIS, G., AND STEPHANIDIS, C. Enhancing the Customers' Experience Using an Augmented Reality Mirror. In *HCI International 2016 - Posters' Extended Abstracts*, C. Stephanidis, Ed., vol. 618. Springer International Publishing, Cham, 2016, pp. 479–484. DOI: 10.1007/978-3-319-40542-1_77.

[48] BOHN, J., COROAMA, V., LANGHEINRICH, M., MATTERN, F., AND ROHS, M. Social, economic, and ethical implications of ambient intelligence and ubiquitous computing. In *Ambient intelligence*. Springer, 2005, pp. 5–29.

[49] BOLIN, M. *Closure: The Definitive Guide: Google Tools to Add Power to Your JavaScript.* " O'Reilly Media, Inc.", 2010.

[50] BOMAN, J., TAYLOR, J., AND NGU, A. H. Flexible iot middleware for integration of things and applications. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on* (2014), IEEE, pp. 481–488.

[51] BONGIOVANNI, R., AND LOWENBERG-DEBOER, J. Precision agriculture and sustainability. *Precision agriculture 5*, 4 (2004), 359–387.

[52] BONOMI, F. The smart and Connected Vehicle and the Internet of Things. In *Workshop on Synchronization in Telecommunication Systems (WSTS)* (2013).

[53] BONOMI, F., MILITO, R., ZHU, J., AND ADDEPALLI, S. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* (2012), ACM, pp. 13–16.

[54] BOOTH, T., AND STUMPF, S. End-user experiences of visual and textual programming environments for arduino. In *International Symposium on End User Development* (2013), Springer, pp. 25–39.

[55] BORENSTEIN, N. S., AND FREED, N. Multipurpose internet mail extensions (mime) part two: Media types.

[56] BORMANN, C., CASTELLANI, A. P., AND SHELBY, Z. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing 16*, 2 (2012), 62–67.

[57] BOTTS, M., PERCIVALL, G., REED, C., AND DAVIDSON, J. Ogc® sensor web enablement: Overview and high level architecture. *GeoSensor networks* (2008), 175–190.

[58] BRABHAM, D. C. *Crowdsourcing*. Wiley Online Library, 2013.

[59] BREIVOLD, H. P., AND SANDSTRÖM, K. Internet of things for industrial automation–challenges and technical solutions. In *Data Science and Data Intensive Systems (DSDIS), 2015 IEEE International Conference on* (2015), IEEE, pp. 532–539.

[60] BROOKE, J., ET AL. Sus-a quick and dirty usability scale. *Usability evaluation in industry 189*, 194 (1996), 4–7.

[61] BROOKS, R. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation 2*, 1 (1986), 14–23.

[62] BROOKS, R. A. The intelligent room project. In *Cognitive Technology, 1997. Humanizing the Information Age. Proceedings., Second International Conference on* (1997), IEEE, pp. 271–278.

[63] BROOKS, R. A. *Cambrian intelligence: The early history of the new AI*, vol. 97. MIT press Cambridge, MA, 1999.

[64] BURNETT, M. M., ROTHERMEL, G., AND COOK, C. R. An integrated software engineering approach for end-user programmers. *End User Development 9* (2006), 87–113.

[65] CAHILL, V., AND HAAHR, M. Real+ virtual= clever: thoughts on programming smart environments.

[66] CAMPBELL-KELLY, M., ASPRAY, W., SNOWMAN, D. P., MCKAY, S. R., CHRISTIAN, W., ET AL. Computer a history of the information machine. *Computers in Physics 11*, 3 (1997), 256–257.

[67] CANTONI, R. 11 bodyarchitecture: the evolution of interface towards ambient intelligence.

[68] CAPORUSCIO, M., RAVERDY, P.-G., AND ISSARNY, V. ubisoap: A service-oriented middleware for ubiquitous networking. *IEEE Transactions on Services Computing 5*, 1 (2012), 86–98.

[69] CARLSON, J. L. *Redis in Action*. Manning Publications Co., 2013.

[70] CHAKRABARTI, S. K., AND KUMAR, P. Test-the-rest: An approach to testing restful web-services. In *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD'09. Computation World:* (2009), IEEE, pp. 302–308.

[71] CHARETTE, R. N. Why software fails [software failure]. *Ieee Spectrum 42*, 9 (2005), 42–49.

[72] CHARLES GOLVIN, LIZZY FOO KUNE, NOAH ELKIN, ANDREW FRANK, AND JAKE SOROFMAN. Predicts 2017: Marketers, Expect the Unexpected, Nov. 2016.

[73] CHATZOPOULOS, D., BERMEJO, C., HUANG, Z., AND HUI, P. Mobile Augmented Reality Survey: From Where We Are to Where We Go. *IEEE Access 5* (2017), 6917–6950.

[74] CHEN, H., PERICH, F., FININ, T., AND JOSHI, A. Soupa: Standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on* (2004), IEEE, pp. 258–267.

[75] CHIANG, M. Fog networking: An overview on research opportunities. *arXiv preprint arXiv:1601.00835* (2016).

[76] CHRISTENSEN, B. Introducing hystrix for resilience engineering. *Netflix Tech Blog*. (2012).

[77] CISCO. *the Internet of Things: Extend the Cloud to Where the Things Are*. Cisco Syst., San Jose, CA, USA, 2016.

[78] COLUMBUS, L. Roundup of cloud computing forecasts and market estimates, 2015. *Forbes Magazine* (2015).

[79] COLUMBUS, L. Roundup Of Internet Of Things Forecasts And Market Estimates, 2016, Nov. 2016.

[80] CONSORTIUM, U., ET AL. *The Unicode Standard, Version 2.0*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[81] CONZON, D., BOLOGNESI, T., BRIZZI, P., LOTITO, A., TOMASI, R., AND SPIRITO, M. A. The virtus middleware: An xmpp based architecture for secure iot communications. In *Computer communications and networks (ICCCN), 2012 21st international conference on* (2012), IEEE, pp. 1–6.

[82] COOK, D. J., AUGUSTO, J. C., AND JAKKULA, V. R. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing 5*, 4 (Aug. 2009), 277–298.

[83] COOK, D. J., AND DAS, S. K. How smart are our environments? An updated look at the state of the art. *Pervasive and Mobile Computing 3*, 2 (Mar. 2007), 53–73.

[84] COUTAZ, J., AND CROWLEY, J. L. A first-person experience with end-user development for smart homes. *IEEE Pervasive Computing 15*, 2 (2016), 26–39.

[85] COUTAZ, J., CROWLEY, J. L., DOBSON, S., AND GARLAN, D. Context is key. *Communications of the ACM 48*, 3 (2005), 49–53.

[86] CRAIG, J. Automating for Digital TransformationTools-Driven DevOps and Continuous Software Delivery in the Enterprise - Report Summary, Dec. 2015.

[87] CRANG, M., AND GRAHAM, S. Sentient cities ambient intelligence and the politics of urban space. *Information, Communication & Society 10*, 6 (2007), 789–817.

[88] CROCKFORD, D. The application/json media type for javascript object notation (json).

[89] DA ROCHA, R. C. A., AND ENDLER, M. Middleware: Context management in heterogeneous, evolving ubiquitous environments. *IEEE Distributed Systems Online 7*, 4 (2006), 1–1.

[90] DA XU, L., HE, W., AND LI, S. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics 10*, 4 (2014), 2233–2243.

[91] DAN, Y. Digital Integrated Circuits.

[92] DASTJERDI, A. V., AND BUYYA, R. Fog computing: Helping the Internet of Things realize its potential. *Computer 49*, 8 (2016), 112–116.

[93] DASTJERDI, A. V., GUPTA, H., CALHEIROS, R. N., GHOSH, S. K., AND BUYYA, R. Fog computing: Principles, architectures, and applications. *arXiv preprint arXiv:1601.02752* (2016).

[94] DE, B. Api documentation. In *API Management*. Springer, 2017, pp. 59–80.

[95] DE ROECK, D., SLEGERS, K., CRIEL, J., GODON, M., CLAEYS, L., KILPI, K., AND JACOBS, A. I would diyse for it!: a manifesto for do-it-yourself internet-of-things creation. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design* (2012), ACM, pp. 170–179.

[96] DE RUYTER, B., AND AARTS, E. Ambient intelligence: visualizing the future. In *Proceedings of the working conference on Advanced visual interfaces* (2004), ACM, pp. 203–208.

[97] DE SILVA, L. C., MORIKAWA, C., AND PETRA, I. M. State of the art of smart homes. *Engineering Applications of Artificial Intelligence 25*, 7 (2012), 1313–1321.

[98] DELICATO, F. C., PIRES, P. F., BATISTA, T., CAVALCANTE, E., COSTA, B., AND BARROS, T. Towards an iot ecosystem. In *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems* (2013), ACM, pp. 25–28.

[99] DENG, Y., CHURCHER, C., ABELL, W., AND MCCALLUM, J. Designing a framework for end user applications. *End-User Development* (2011), 67–75.

[100] DESHPANDE, A., SHAHANE, A., GADRE, D., DESHPANDE, M., AND JOSHI, P. M. A survey of various chatbot implementation techniques.

[101] DESOLDA, G., ARDITO, C., AND MATERA, M. Empowering end users to customize their smart environments: Model, composition paradigms, and domain-specific tools. *ACM Transactions on Computer-Human Interaction (TOCHI) 24*, 2 (2017), 12.

[102] DI RIENZO, A., GARZOTTO, F., CREMONESI, P., FRG•, C., AND VALLA, M. Towards a smart retail environment. ACM Press, pp. 779–782.

[103] DING, L., ZHOU, C., DENG, Q., LUO, H., YE, X., NI, Y., AND GUO, P. Real-time safety early warning system for cross passage construction in Yangtze Riverbed Metro Tunnel based on the internet of things. *Automation in Construction 36* (Dec. 2013), 25–37.

[104] DIX, A. Human-computer interaction. In *Encyclopedia of database systems*. Springer, 2009, pp. 1327–1331.

[105] DJURDJANOVIC, D., LEE, J., AND NI, J. Watchdog agent-an infotronics-based prognostics approach for product performance degradation assessment and prediction. *Advanced Engineering Informatics 17*, 3 (2003), 109–125.

[106] DRINKWATER, D. 10 stellar real-life examples of IoT taking flight in aviation, Aug. 2016.

[107] DU, C., AND ZHU, S. Research on Urban Public Safety Emergency Management Early Warning System based on Technologies for the Internet of Things. *Procedia Engineering 45* (2012), 748–754.

[108] DUCATEL, K., BOGDANOWICZ, M., SCAPOLO, F., LEIJTEN, J., AND BURGELMAN, J.-C. *Scenarios for ambient intelligence in 2010*. Office for official publications of the European Communities Luxembourg, 2001.

[109] DUCATEL, K., BOGDANOWICZ, M., SCAPOLO, F., LEIJTEN, J., AND BURGELMAN, J.-C. Ambient intelligence: From vision to reality. *IST Advisory Group Draft Report, European Commission* (2003).

[110] DUVALL, P. M., MATYAS, S., AND GLOVER, A. *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[111] EARLE, R. H., ROSSO, M. A., AND ALEXANDER, K. E. User preferences of software documentation genres. In *Proceedings of the 33rd Annual International Conference on the Design of Communication* (2015), ACM, p. 46.

[112] ECMA, E. 262: Ecmascript language specification. *ECMA (European Association for Standardizing Information and Communication Systems), pub-ECMA: adr,* (1999).

[113] ECMA, S. 404: The json data interchange format. *Available via DIALOG http://b•• www.b•• ecma-international.b•• org/b•• publications/b•• files/b•• ECMA-ST/b•• ECMA-404.b•• pdf. Cited on 9* (2015).

[114] EJIGU, D., SCUTURICI, M., AND BRUNIE, L. Semantic approach to context management and reasoning in ubiquitous context-aware systems. In *Digital Information Management, 2007. ICDIM'07. 2nd International Conference on* (2007), vol. 1, IEEE, pp. 500–505.

[115] ELEFTHERAKIS, G., PAPPAS, D., LAGKAS, T., ROUSIS, K., AND PAUNOVSKI, O. Architecting the iot paradigm: a middleware for autonomous distributed sensor networks. *International Journal of Distributed Sensor Networks 11*, 12 (2015), 139735.

[116] ENDRES, C., BUTZ, A., AND MACWILLIAMS, A. A survey of software infrastructures and frameworks for ubiquitous computing. *Mobile Information Systems 1*, 1 (2005), 41–80.

[117] ERIKSSON, M., AND HALLBERG, V. Comparison between json and yaml for data serialization. *The School of Computer Science and Engineering Royal Institute of Technology* (2011).

[118] ERL, T., COPE, R., AND NASERPOUR, A. *Cloud computing design patterns*. Prentice Hall Press, 2015.

[119] FERSCHA, A., RESMERITA, S., AND HOLZMANN, C. Human computer confluence. *Universal Access in Ambient Intelligence Environments* (2007), 14–27.

[120] FERSI, G. Middleware for internet of things: A study. In *Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference on* (2015), IEEE, pp. 230–235.

[121] FIELDING, R. Representational state transfer. *Architectural Styles and the Design of Netowork-based Software Architecture* (2000), 76–85.

[122] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. Hypertext transfer protocol–http/1.1. Tech. rep., 1999.

[123] FIELDING, R. T., AND TAYLOR, R. N. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[124] FLECK, M., FRID, M., KINDBERG, T., O'BRIEN-STRAIN, E., RAJANI, R., AND SPASOJEVIC, M. Rememberer: A tool for capturing museum visits. In *International Conference on Ubiquitous Computing* (2002), Springer, pp. 48–55.

[125] FOGLI, D., LANZILOTTI, R., AND PICCINNO, A. End-user development tools for the smart home: a systematic literature review. In *International Conference on Distributed, Ambient, and Pervasive Interactions* (2016), Springer, pp. 69–79.

[126] FORSYTHAND, E. N., AND MARTELL, C. H. Lexical and discourse analysis of online chat dialog. In *Semantic Computing, 2007. ICSC 2007. International Conference on* (2007), IEEE, pp. 19–26.

[127] FORTINO, G., GUERRIERI, A., LACOPO, M., LUCIA, M., AND RUSSO, W. An agent-based middleware for cooperating smart objects. In *International Conference on Practical Applications of Agents and Multi-Agent Systems* (2013), Springer, pp. 387–398.

[128] FORTINO, G., GUERRIERI, A., RUSSO, W., AND SAVAGLIO, C. Middlewares for smart objects and smart environments: overview and comparison. In *Internet of Things Based on Smart Objects*. Springer, 2014, pp. 1–27.

[129] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. FIPA ACL Message Structure Specification, Dec. 2002.

[130] FOWLER, M., AND FOEMMEL, M. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf 122* (2006).

[131] FRANKE, M., SEIDL, C., AND SCHLEGEL, T. A seamless integration, semantic middleware for cyber-physical systems. In *Networking, Sensing and Control (ICNSC), 2013 10th IEEE International Conference on* (2013), IEEE, pp. 627–632.

[132] FRASER, N., ET AL. Blockly: A visual programming editor. *URL: https://code. google. com/p/blockly* (2013).

[133] FREEMAN, A. Working with visual studio code. In *Pro ASP. NET Core MVC*. Springer, 2016, pp. 343–369.

[134] FRIEDEWALD, M., DA COSTA, O., PUNIE, Y., ALAHUHTA, P., AND HEINONEN, S. Perspectives of ambient intelligence in the home environment. *Telematics and informatics 22*, 3 (2005), 221–238.

[135] GAMMA, E. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[136] GARRETT, J. J. *The Elements of User Experience : User-centered Design for the Web and Beyond. Voices that matter*. Pearson Education, Berkeley, Calif. New Riders London, 2011.

[137] GARRETT, J. J., ET AL. Ajax: A new approach to web applications.

[138] GARSKE, J., LULLA, S., MATTHEW, M., AND THUDIUM, S. Connected A&D: IOT driving new revenue in the aftermarket. Tech. rep., Sept. 2016.

[139] GARTNER. Top TrendsB•in the Gartner Hype Cycle for Emerging Technologies, 2017 - Smarter With Gartner, July 2017.

[140] GEISZ, L. Zepp sensor for golf review. *the gadgeteer, downloaded from the internet at http://the-gadgeteer. com/2013/12/13/zepp-sensor-for-golf-review/on Apr 2* (2014), 19.

[141] GEORGAKOPOULOS, D., JAYARAMAN, P. P., FAZIA, M., VILLARI, M., AND RANJAN, R. Internet of things and edge cloud computing roadmap for manufacturing. *IEEE Cloud Computing 3*, 4 (2016), 66–73.

[142] GEORGALIS, Y., GRAMMENOS, D., AND STEPHANIDIS, C. Middleware for ambient intelligence environments: Reviewing requirements and communication technologies. *Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments* (2009).

[143] GHIANI, G., MANCA, M., PATERNÒ, F., AND SANTORO, C. Personalization of context-dependent applications through trigger-action rules. *ACM Transactions on Computer-Human Interaction (TOCHI) 24*, 2 (2017), 14.

[144] GIL, D., FERRANDEZ, A., MORA-MORA, H., AND PERAL, J. Internet of things: A review of surveys based on context aware intelligent services. *Sensors 16*, 7 (2016), 1069.

[145] GIORDANO, A., AND SPEZZANO, G. Service-oriented middleware for the cooperation of smart objects and web services. In *Internet of Things Based on Smart Objects*. Springer, 2014, pp. 49–68.

[146] GONG, L. Jxta: A network programming environment. *IEEE Internet Computing 5*, 3 (2001), 88–95.

[147] GOOGLE. Firebase.

[148] GOSLING, J., JOY, B., STEELE, G. L., BRACHA, G., AND BUCKLEY, A. *The Java language specification*. Pearson Education, 2014.

[149] GRAMMENOS, D., ZABULIS, X., MICHEL, D., PADELERIS, P., SARMIS, T., GEORGALIS, G., KOUTLEMANIS, P., TZEVANIDIS, K., ARGYROS, A., SIFAKIS, M., ET AL. A prototypical interactive exhibition for the archaeological museum of thessaloniki. *International Journal of Heritage in the Digital Era 2*, 1 (2013), 75–99.

[150] GROUP, C. P. S. P. W. Framework for Cyber-Physical Systems Release 1.0, May 2016.

[151] GS1. Internet of Things (IoT), Nov. 2016.

[152] GUBBI, J., BUYYA, R., MARUSIC, S., AND PALANISWAMI, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems 29*, 7 (2013), 1645–1660.

[153] GUINARD, D., AND TRIFA, V. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain* (2009), vol. 15.

[154] GUSMEROLI, S., PICCIONE, S., AND ROTONDI, D. Iot@work automation middleware system design and architecture. In *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on* (2012), IEEE, pp. 1–8.

[155] GYRARD, A., BONNET, C., AND BOUDAOUD, K. Demo paper: Helping iot application developers with sensor-based linked open rules. In *TC/SSN@ ISWC* (2014), pp. 105–108.

[156] GYRARD, A., DATTA, S. K., BONNET, C., AND BOUDAOUD, K. Cross-domain internet of things application development: M3 framework and evaluation. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on* (2015), IEEE, pp. 9–16.

[157] HAGRAS, H., CALLAGHAN, V., COLLEY, M., CLARKE, G., POUNDS-CORNISH, A., AND DUMAN, H. Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems 19*, 6 (2004), 12–20.

[158] HAHN, E. *Express in Action: Node Applications with Express and Its Companion Tools*, 1st ed. Manning Publications Co., Greenwich, CT, USA, 2015.

[159] HALPERN, M., ZHU, Y., AND REDDI, V. J. Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on* (2016), IEEE, pp. 64–76.

[160] HASAN, S., AND CURRY, E. Thingsonomy: Tackling variety in internet of things events. *IEEE Internet Computing 19*, 2 (2015), 10–18.

[161] HELMER, R. J., MESTROVIC, M. A., FARROW, D., LUCAS, S., AND SPRATFORD, W. Smart Textiles: Position and Motion Sensing for Sport, Entertainment and Rehabilitation. *Advances in Science and Technology 60* (2008), 144–153.

[162] HENNESSY, J. L., AND PATTERSON, D. A. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[163] HENRICKSEN, K., AND INDULSKA, J. A software engineering framework for context-aware pervasive computing. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on* (2004), IEEE, pp. 77–86.

[164] HERBST, N. R., KOUNEV, S., AND REUSSNER, R. H. Elasticity in Cloud Computing: What It Is, and What It Is Not. In *ICAC* (2013), vol. 13, pp. 23–27.

[165] HILKEN, T., DE RUYTER, K., CHYLINSKI, M., MAHR, D., AND KEELING, D. I. Augmenting the eye of the beholder: exploring the strategic potential of augmented reality to enhance online service experiences. *Journal of the Academy of Marketing Science* (May 2017).

[166] HINMAN, R. *The mobile frontier: a guide for designing mobile experiences*. Rosenfeld Media, 2012.

[167] HOANG, D. D., AND KIM, H.-Y. P. C.-K. Service-oriented middleware architectures for cyber-physical systems.

[168] HOAREAU, D., AND MAHÉO, Y. Middleware support for the deployment of ubiquitous software components. *Personal and ubiquitous computing 12*, 2 (2008), 167–178.

[169] HOHPE, G., AND WOOLF, B. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[170] HOLLOWAY, S., AND JULIEN, C. The case for end-user programming of ubiquitous computing environments. In *Proceedings of the FSE/SDP workshop on Future of software engineering research* (2010), ACM, pp. 167–172.

[171] HONG, Y. A resource-oriented middleware framework for heterogeneous internet of things. In *Cloud and Service Computing (CSC), 2012 International Conference on* (2012), IEEE, pp. 12–16.

[172] HUEBSCHER, M. C., AND MCCANN, J. A. Adaptive middleware for context-aware applications in smart-homes. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing* (2004), ACM, pp. 111–116.

[173] HUEBSCHER, M. C., AND MCCANN, J. A. An adaptive middleware framework for context-aware applications. *Personal and Ubiquitous Computing 10*, 1 (2006), 12–20.

[174] HUMBLE, J., AND FARLEY, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.

[175] HUNT, R., AND POLTROCK, S. Boeing operations fleet support: A case study in integrated workplace design. *Cooperative Buildings. Integrating Information, Organizations, and Architecture* (1999), 2–11.

[176] HUTTERMANN, M. *DevOps for developers*. Apress, 2012.

[177] HWANGBO, H., KIM, Y. S., AND CHA, K. J. Use of the Smart Store for Persuasive Marketing and Immersive Customer Experiences: A Case Study of Korean Apparel Enterprise. *Mobile Information Systems 2017* (2017), 1–17.

[178] HYNES, G., REYNOLDS, V., AND HAUSWIRTH, M. A Context Lifecycle for Web-Based Context Management Services. In *EuroSSC* (2009), Springer, pp. 51–65.

[179] ISSARNY, V., CAPORUSCIO, M., AND GEORGANTAS, N. A perspective on the future of middleware-based software engineering. In *2007 Future of Software Engineering* (2007), IEEE Computer Society, pp. 244–258.

[180] JADEJA, Y., AND MODI, K. Cloud computing-concepts, architecture and challenges. In *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on* (2012), IEEE, pp. 877–880.

[181] JING, J., HELAL, A. S., AND ELMAGARMID, A. Client-server computing in mobile environments. *ACM computing surveys (CSUR) 31*, 2 (1999), 117–157.

[182] JOHNS, H. Understanding zerocof and multicast dns. *O'Reilly Wireless DevCenter, Dec 20* (2002), 1–5.

[183] JUNG, H.-S., JEONG, C.-S., LEE, Y.-W., AND HONG, P.-D. An intelligent ubiquitous middleware for u-city: Smartum. *Journal of Information Science & Engineering 25*, 2 (2009).

[184] KAMEAS, A., MAVROMMATI, I., AND MARKOPOULOS, P. Computing in tangible: using artifacts as components of ambient intelligence environments. *Ambient Intelligence* (2005), 121–142.

[185] KAMM, C. A., AND WALKER, M. A. Design and evaluation of spoken dialog systems. In *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on* (1997), IEEE, pp. 11–18.

[186] KANNER, A. D., COYNE, J. C., SCHAEFER, C., AND LAZARUS, R. S. Comparison of two modes of stress measurement: Daily hassles and uplifts versus major life events. *Journal of behavioral medicine 4*, 1 (1981), 1–39.

[187] KAPNAS, G., LEONIDIS, A., KOROZI, M., NTOA, S., MARGETIS, G., AND STEPHANIDIS, C. A museum guide application for deployment on user-owned mobile devices. In *International Conference on Human-Computer Interaction* (2013), Springer, pp. 253–257.

[188] KARLSON, A. K., IQBAL, S. T., MEYERS, B., RAMOS, G., LEE, K., AND TANG, J. C. Mobile taskflow in context: a screenshot study of smartphone usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), ACM, pp. 2009–2018.

[189] KAUR, R., AND LUTHRA, P. Load balancing in cloud computing. In *Proceedings of International Conference on Recent Trends in Information, Telecommunication and Computing, ITC* (2012).

[190] KENDALL, J. E., AND KENDALL, K. E. Information delivery systems: an exploration of web pull and push technologies. *Communications of the AIS 1*, 4es (1999), 1.

[191] KIM, G., DEBOIS, P., WILLIS, J., AND HUMBLE, J. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, 2016.

[192] KLEINBERGER, T., BECKER, M., RAS, E., HOLZINGER, A., AND MÜLLER, P. Ambient intelligence in assisted living: enable elderly people to handle future interfaces. In *International Conference on Universal Access in Human-Computer Interaction* (2007), Springer, pp. 103–112.

[193] KO, K.-E., AND SIM, K.-B. Development of context aware system based on bayesian network driven context reasoning method and ontology context modeling. In *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on* (2008), IEEE, pp. 2309–2313.

[194] KON, F., COSTA, F., BLAIR, G., AND CAMPBELL, R. H. The case for reflective middleware. *Communications of the ACM 45*, 6 (2002), 33–38.

[195] KONSTAS, I., STATHOPOULOS, V., AND JOSE, J. M. On social networks and collaborative recommendation. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (2009), ACM, pp. 195–202.

[196] KOROZI, M., ANTONA, M., NTAGIANTA, A., LEONIDIS, A., AND STEPHANIDIS, C. 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).

[197] KOROZI, M., LEONIDIS, A., ANTONA, M., AND STEPHANIDIS, C. Lector: Towards Reengaging Students in the Educational Process inside Smart Classrooms. Springer.

[198] KOROZI, M., NTOA, S., ANTONA, M., LEONIDIS, A., AND STEPHANIDIS, C. Towards building pervasive uis for the intelligent classroom: the pupil approach. In *Proceedings of the International Working Conference on Advanced Visual Interfaces* (2012), ACM, pp. 279–286.

[199] KORTUEM, G., BANDARA, A. K., SMITH, N., RICHARDS, M., AND PETRE, M. Educating the internet-of-things generation. *Computer 46*, 2 (2013), 53–61.

[200] KORTUEM, G., KAWSAR, F., SUNDRAMOORTHY, V., AND FITTON, D. Smart objects as building blocks for the internet of things. *IEEE Internet Computing 14*, 1 (2010), 44–51.

[201] KORTUEM, G., KAWSAR, F., SUNDRAMOORTHY, V., AND FITTON, D. Smart objects as building blocks for the internet of things. *IEEE Internet Computing 14*, 1 (2010), 44–51.

[202] KOSHIZUKA, N., AND SAKAMURA, K. Ubiquitous ID: Standards for Ubiquitous Computing and the Internet of Things. *IEEE Pervasive Computing 9*, 4 (2010), 98–101.

[203] KOSMATOS, E. A., TSELIKAS, N. D., AND BOUCOUVALAS, A. C. Integrating RFIDs and Smart Objects into a UnifiedInternet of Things Architecture. *Advances in Internet of Things 01*, 01 (2011), 5–12.

[204] KOSTELNIK, P., SARNOVSK, M., AND FURDIK, K. The semantic middleware for networked embedded systems applied in the internet of things and services domain. *Scalable Computing: Practice and Experience 12*, 3 (2011), 307–316.

[205] KOVATSCH, M., LANTER, M., AND DUQUENNOY, S. Actinium: A restful runtime container for scriptable internet of things applications. In *Internet of Things (IOT), 2012 3rd International Conference on the* (2012), IEEE, pp. 135–142.

[206] KRAMER, D. Api documentation from source code comments: a case study of javadoc. In *Proceedings of the 17th annual international conference on Computer documentation* (1999), ACM, pp. 147–153.

[207] KRONER, A., SCHNEIDER, M., AND MORI, J. A framework for ubiquitous content sharing. *IEEE Pervasive Computing 8*, 4 (2009).

[208] KRUG, S. *Don't make me think!: a common sense approach to Web usability*. Pearson Education India, 2000.

[209] KRYLOVSKIY, A., JAHN, M., AND PATTI, E. Designing a smart city internet of things platform with microservice architecture. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on* (2015), IEEE, pp. 25–30.

[210] LACKOVIC, M., AND TRUNFIO, P. A service-oriented discovery framework for cooperating smart objects. In *Internet of Things Based on Smart Objects*. Springer, 2014, pp. 85–105.

[211] LALIS, S., SAVIDIS, A., AND STEPHANIDES, C. Supporting distributed user interfaces in mobile and wearable device ensembles: the 2wear experience. In *Proceedings of the Ensembles of On-Body Devices workshop at Mobile HCI* (2010), vol. 2010.

[212] LANDIN, P. J. The mechanical evaluation of expressions. *The Computer Journal 6*, 4 (1964), 308–320.

[213] LANGHEINRICH, M., MATTERN, F., RÖMER, K., AND VOGT, H. First steps towards an event-based infrastructure for smart things. In *Ubiquitous Computing Workshop (PACT 2000)* (2000), p. 34.

[214] LANZA, J., SÁNCHEZ, L., GUTIÉRREZ, V., GALACHE, J. A., SANTANA, J. R., SOTRES, P., AND MUÑOZ, L. Smart city services over a future internet platform based on internet of things and cloud: The smart parking case. *Energies 9*, 9 (2016), 719.

[215] LEE, E. A. Cyber-physical systems-are computing foundations adequate. In *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap* (2006), vol. 2.

[216] LEE, E. A. Cyber physical systems: Design challenges. In *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on* (2008), IEEE, pp. 363–369.

[217] LEE, J., BAGHERI, B., AND KAO, H.-A. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters 3* (Jan. 2015), 18–23.

[218] LEIVA, L., BÖHMER, M., GEHRING, S., AND KRÜGER, A. Back to the app: the costs of mobile application interruptions. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services* (2012), ACM, pp. 291–294.

[219] LEONIDIS, A., ARAMPATZIS, D., KOROZI, M., ADAMI, I., NTOA, S., AND STEPHANIDIS, C. Home game: an educational game for children with cognitive impairments. In *Proceedings of the 2017 Conference on Interaction Design and Children* (2017), ACM, pp. 667–670.

[220] LEONIDIS, A., KOROZI, M., MARGETIS, G., GRAMMENOS, D., AND STEPHANIDIS, C. An intelligent hotel room. In *International Joint Conference on Ambient Intelligence* (2013), Springer, Cham, pp. 241–246.

[221] LEONIDIS, A., KOROZI, M., MARGETIS, G., NTOA, S., PAPAGIANNAKIS, H., ANTONA, M., AND STEPHANIDIS, C. A glimpse into the ambient classroom. *Bulletin of the IEEE Technical Committee on Learning Technology 14*, 4 (2012), 3–6.

[222] LETHBRIDGE, T. C., SINGER, J., AND FORWARD, A. How software engineers use documentation: The state of the practice. *IEEE software 20*, 6 (2003), 35–39.

[223] LIEBERMAN, H., PATERNÒ, F., KLANN, M., AND WULF, V. End-user development: An emerging paradigm. In *End user development*. Springer, 2006, pp. 1–8.

[224] LIN, B., CHEN, Y., CHEN, X., AND YU, Y. Comparison between JSON and XML in Applications Based on AJAX. IEEE, pp. 1174–1177.

[225] LINDWER, M., MARCULESCU, D., BASTEN, T., ZIMMERMANN, R., MARCULESCU, R., JUNG, S., AND CANTATORE, E. Ambient intelligence visions and achievements: Linking abstract ideas to real-world concepts. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1* (2003), IEEE Computer Society, p. 10010.

[226] LIU, Y., NGU, A. H., AND ZENG, L. Z. Qos computation and policing in dynamic web service selection. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (2004), ACM, pp. 66–73.

[227] LIU, Y., PENG, Y., WANG, B., YAO, S., AND LIU, Z. Review on cyber-physical systems. *IEEE/-CAA Journal of Automatica Sinica 4*, 1 (Jan. 2017), 27–40.

[228] LOELIGER, J., AND MCCULLOUGH, M. *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.", 2012.

[229] LOUKIDES, M. *What is DevOps?* " O'Reilly Media, Inc.", 2012.

[230] LOULOUDAKIS, N., LEONIDIS, A., AND STEPHANIDIS, C. AmITest: A Testing Framework for Ambient Intelligence Learning Applications. In *Eighth International Conference on Mobile, Hybrid, and On-line Learning* (Venice, Italy, Apr. 2016), ThinkMind.

[231] LUAN, T. H., GAO, L., LI, Z., XIANG, Y., WEI, G., AND SUN, L. Fog computing: Focusing on mobile users at the edge. *arXiv preprint arXiv:1502.01815* (2015).

[232] LUCERO, S. Iot platforms: enabling the internet of things. *HIS Technology. Retrieved from https://cdn. ihs. com/www/pdf/enabling-IOT. pdf* (2016).

[233] LUNARDI, W. T., DE MATOS, E., TIBURSKI, R., AMARAL, L. A., MARCZAK, S., AND HESSEL, F. Context-based search engine for industrial IoT: Discovery, search, selection, and usage of devices. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on* (2015), IEEE, pp. 1–8.

[234] MAGOUTIS, K., PAPOULAS, C., PAPAIOANNOU, A., KARNIAVOURA, F., AKESTORIDIS, D.-G., PAROTSIDIS, N., KOROZI, M., LEONIDIS, A., NTOA, S., AND STEPHANIDIS, C. Design and implementation of a social networking platform for cloud deployment specialists. *Journal of Internet Services and Applications 6*, 1 (2015), 19.

[235] MAINETTI, L., MIGHALI, V., AND PATRONO, L. An iot-based user-centric ecosystem for heterogeneous smart home environments. In *Communications (ICC), 2015 IEEE International Conference on* (2015), IEEE, pp. 704–709.

[236] MANRICKS, G. *Instant Handlebars. js.* Packt Publishing Ltd, 2013.

[237] MANYIKA, J., CHUI, M., BISSON, P., WOETZEL, J., DOBBS, R., BUGHIN, J., AND AHARON, D. Unlocking the Potential of the Internet of Things. *McKinsey Global Institute* (2015).

[238] MARDAN, A. *Express. js Guide: The Comprehensive Book on Express. js.* Azat Mardan, 2014.

[239] MARGETIS, G., ZABULIS, X., NTOA, S., KOUTLEMANIS, P., PAPADAKI, E., ANTONA, M., AND STEPHANIDIS, C. Enhancing education through natural interaction with physical paper. *Universal Access in the Information Society 14*, 3 (2015), 427–447.

[240] MARK JAFFE, PRELERT. IoT Wonb••t Work Without Artificial Intelligence | WIRED.

[241] MARTIN, R. C. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[242] MARX, G. T. Murky conceptual waters: The public and the private. *Ethics and Information technology 3*, 3 (2001), 157–169.

[243] MATHIOUDAKIS, G., LEONIDIS, A., KOROZI, M., MARGETIS, G., NTOA, S., ANTONA, M., AND STEPHANIDIS, C. Ami-ria: real-time teacher assistance tool for an ambient intelligence classroom. In *Proceedings of the Fifth International Conference on Mobile, Hybrid, and On-Line Lerning (eLmL 2013)* (2013), pp. 37–42.

[244] MATVELLOSO. Principles of bot design - Bot Framework.

[245] MAVROMMATI, I., AND DARZENTAS, J. End user tools for ambient intelligence environments: an overview. *Human-computer interaction. Interaction platforms and techniques* (2007), 864–872.

[246] MAYBURY, M. T., AND WAHLSTER, W. *Readings in intelligent user interfaces*. Morgan Kaufmann, 1998.

[247] MAYHEW, D. J. The usability engineering lifecycle. In *CHI'99 Extended Abstracts on Human Factors in Computing Systems* (1999), ACM, pp. 147–148.

[248] MCBRATNEY, A., WHELAN, B., ANCEV, T., AND BOUMA, J. Future directions of precision agriculture. *Precision agriculture 6*, 1 (2005), 7–23.

[249] MCTEAR, M., CALLEJAS, Z., AND GRIOL, D. The conversational interface. *New York: Springer 10* (2016), 978–3.

[250] MELL, P., AND GRANCE, T. The NIST definition of cloud computing.

[251] MENGE, F. Enterprise service bus. In *Free and open source software conference* (2007), vol. 2, pp. 1–6.

[252] MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal 2014*, 239 (2014), 2.

[253] MERZ, H., HANSEMANN, T., AND HÜBNER, C. *Building Automation: Communication Systems with EIB/KNX, LON and BACnet*. Springer Science & Business Media, 2009.

[254] MESNIER, M., GANGER, G., AND RIEDEL, E. Storage area networking - Object-based storage. *IEEE Communications Magazine 41*, 8 (Aug. 2003), 84–90.

[255] MICHELSON, B. M. Event-driven architecture overview. *Patricia Seybold Group 2* (2006).

[256] MIDDLETON, P., KOSLOWSKI, T., AND GUPTA, A. Forecast Analysis: Internet of Things b•" Endpoints, Worldwide, 2015 Update, Dec. 2015.

[257] MIRAZ, M. H., ALI, M., EXCELL, P. S., AND PICKING, R. A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT). In *Internet Technologies and Applications (ITA), 2015* (2015), IEEE, pp. 219–224.

[258] MITCHELL, J. C. *Concepts in programming languages*. Cambridge University Press, 2003.

[259] MODAHL, M., AGARWALLA, B., ABOWD, G., RAMACHANDRAN, U., AND SAPONAS, T. S. Toward a standard ubiquitous computing framework. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing* (2004), ACM, pp. 135–139.

[260] MOORE, G. E. *Cramming More Components Onto Integrated Circuits, Electronics,(38) 8*. April, 1965.

[261] MOWBRAY, T. J., AND ZAHAVI, R. *The essential CORBA: systems integration using distributed objects*. Wiley New York, 1995.

[262] MOZILLA DEVELOPER NETWORK. Websockets, 2016.

[263] MYERS, B. A. A brief history of human-computer interaction technology. *interactions 5*, 2 (1998), 44–54.

[264] MZAHM, A. M., AHMAD, M. S., AND TANG, A. Y. Agents of things (aot): An intelligent operational concept of the internet of things (iot). In *Intelligent Systems Design and Applications (ISDA), 2013 13th International Conference on* (2013), IEEE, pp. 159–164.

[265] NAKANO, M., HASEGAWA, Y., NAKADAI, K., NAKAMURA, T., TAKEUCHI, J., TORII, T., TSUJINO, H., KANDA, N., AND OKUNO, H. G. A two-layer model for behavior and dialogue planning in conversational service robots. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on* (2005), IEEE, pp. 3329–3335.

[266] NEWMAN, S. *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[267] NGO, H. Q., SHEHZAD, A., LIAQUAT, S., RIAZ, M., AND LEE, S. Developing context-aware ubiquitous computing systems with a unified middleware framework. In *International Conference on Embedded and Ubiquitous Computing* (2004), Springer, pp. 672–681.

[268] NGU, A. H., GUTIERREZ, M., METSIS, V., NEPAL, S., AND SHENG, Q. Z. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal 4*, 1 (2017), 1–20.

[269] NIELSEN, J. Why you only need to test with 5 users, 2000.

[270] NIELSEN, J. Thinking aloud: The# 1 usability tool. nielsen norman group, 2012.

[271] NOTTINGHAM, M., AND FIELDING, R. Additional http status codes.

[272] NURSEITOV, N., PAULSON, M., REYNOLDS, R., AND IZURIETA, C. Comparison of json and xml data interchange formats: a case study. *Caine 2009* (2009), 157–162.

[273] NVIDIA, C. Programming guide, 2010.

[274] NYGREN, E., SITARAMAN, R. K., AND SUN, J. The Akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review 44*, 3 (Aug. 2010), 2.

[275] OGDEN, M. Callback hell, 2015. *URL http://callbackhell. com/.[Online*.

[276] O'LEARY, D. E. Artificial intelligence and big data. *IEEE Intelligent Systems 28*, 2 (2013), 96–99.

[277] OLINER, S. D., AND SICHEL, D. E. The Resurgence of Growth in the Late 1990s: Is Information Technology the Story? SSRN Scholarly Paper ID 233139, Social Science Research Network, Rochester, NY, Mar. 2000.

[278] OB••REILLY, T. What is web 2.0, 2005.

[279] PALVIAINEN, M., KUUSIJÄRVI, J., AND OVASKA, E. Framework for end-user programming of cross-smart space applications. *Sensors 12*, 11 (2012), 14442–14466.

[280] PALVIAINEN, M., KUUSIJÄRVI, J., AND OVASKA, E. A semi-automatic end-user programming approach for smart space application development. *Pervasive and Mobile Computing 12* (2014), 17–36.

[281] PAPAIOANNOU, I. V., TSESMETZIS, D. T., ROUSSAKI, I. G., AND ANAGNOSTOU, M. E. A qos ontology language for web-services. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on* (2006), vol. 1, IEEE, pp. 6–pp.

[282] PAPAZOGLOU, M. P. Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on* (2003), IEEE, pp. 3–12.

[283] PARK, S., AND YOO, Y. Network intelligence based on network state information for connected vehicles utilizing fog computing. *Mobile Information Systems 2017* (2017).

[284] PARK, S. O., PARK, J. H., AND JEONG, Y.-S. An efficient dynamic integration middleware for cyber-physical systems in mobile environments. *Mobile Networks and Applications 18*, 1 (2013), 110–115.

[285] PARKER, D. *JavaScript with Promises: Managing Asynchronous Code*. " O'Reilly Media, Inc.", 2015.

[286] PARTARAKIS, N., ANTONA, M., AND STEPHANIDIS, C. A novel approach to painting powered by ambient intelligence. *EAI Endorsed Transactions on Creative Technologies 3*, 6 (2016), 1–11.

[287] PATIDAR, S., RANE, D., AND JAIN, P. A Survey Paper on Cloud Computing. IEEE, pp. 394–398.

[288] PATKOS, T., BIKAKIS, A., ANTONIOU, G., PAPADOPOULI, M., AND PLEXOUSAKIS, D. Distributed ai for ambient intelligence: issues and approaches. *Ambient Intelligence* (2007), 159–176.

[289] PATKOS, T., PLEXOUSAKIS, D., CHIBANI, A., AND AMIRAT, Y. An event calculus production rule system for reasoning in dynamic and uncertain domains. *Theory and Practice of Logic Programming 16*, 3 (2016), 325–352.

[290] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., ET AL. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research 12*, Oct (2011), 2825–2830.

[291] PEREIRA, C., RODRIGUES, J., PINTO, A., ROCHA, P., SANTIAGO, F., SOUSA, J., AND AGUIAR, A. Smartphones as m2m gateways in smart cities iot applications. In *Telecommunications (ICT), 2016 23rd International Conference on* (2016), IEEE, pp. 1–7.

[292] PERERA, C., JAYARAMAN, P. P., ZASLAVSKY, A., GEORGAKOPOULOS, D., AND CHRISTEN, P. Mosden: An internet of things middleware for resource constrained mobile devices. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on* (2014), IEEE, pp. 1053–1062.

[293] PERERA, C., JAYARAMAN, P. P., ZASLAVSKY, A., GEORGAKOPOULOS, D., AND CHRISTEN, P. Sensor discovery and configuration framework for the internet of things paradigm. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on* (2014), IEEE, pp. 94–99.

[294] PERERA, C., ZASLAVSKY, A., CHRISTEN, P., AND GEORGAKOPOULOS, D. Ca4iot: Context awareness for internet of things. In *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on* (2012), IEEE, pp. 775–782.

[295] PERERA, C., ZASLAVSKY, A., CHRISTEN, P., AND GEORGAKOPOULOS, D. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials 16*, 1 (2014), 414–454.

[296] PERERA, C., ZASLAVSKY, A., CHRISTEN, P., AND GEORGAKOPOULOS, D. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies 25*, 1 (2014), 81–93.

[297] PERERA, C., ZASLAVSKY, A., COMPTON, M., CHRISTEN, P., AND GEORGAKOPOULOS, D. Semantic-driven configuration of internet of things middleware. In *Semantics, Knowledge and Grids (SKG), 2013 Ninth International Conference on* (2013), IEEE, pp. 66–73.

[298] PONNEKANTI, S., LEE, B., FOX, A., HANRAHAN, P., AND WINOGRAD, T. Icrafter: A service framework for ubiquitous computing environments. In *Ubicomp 2001: Ubiquitous Computing* (2001), Springer, pp. 56–75.

[299] PREHOFER, C., VAN GURP, J., AND DI FLORA, C. Towards the web as a platform for ubiquitous applications in smart spaces. In *Second Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI), at Ubicomp* (2007), vol. 2007.

[300] PRESENT, I. Cramming more components onto integrated circuits. *Readings in computer architecture 56* (2000).

[301] PREUVENEERS, D., VAN DEN BERGH, J., WAGELAAR, D., GEORGES, A., RIGOLE, P., CLERCKX, T., BERBERS, Y., CONINX, K., JONCKERS, V., AND DE BOSSCHERE, K. Towards an extensible context ontology for ambient intelligence. In *EUSAI* (2004), vol. 3295, Springer, pp. 148–159.

[302] PRUNICKI, A. Apache thrift, 2009.

[303] QUANTILUS INC. Great Expectations: Delivering Better Service with Augmented Reality, Jan. 2017.

[304] QUIGLEY, M., CONLEY, K., GERKEY, B., FAUST, J., FOOTE, T., LEIBS, J., WHEELER, R., AND NG, A. Y. Ros: an open-source robot operating system. In *ICRA workshop on open source software* (2009), vol. 3, Kobe, p. 5.

[305] R., B. Search is not enough: Synergy between navigation and search. http://www.nngroup.com/articles/search-not-enough/. accessed 8/2015.

[306] RAGASUDHA, S., MAHESWARI, A., AND VENKATESH, T. Internet of things–a survey.

[307] RAJKUMAR, R. R., LEE, I., SHA, L., AND STANKOVIC, J. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference* (2010), ACM, pp. 731–736.

[308] RAMOS, C. Ambient intelligence-a state of the art from artificial intelligence perspective. *Progress in Artificial intelligence* (2007), 285–295.

[309] RANGANATHAN, A., AND CAMPBELL, R. H. A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware* (2003), Springer-Verlag New York, Inc., pp. 143–161.

[310] RANTANEN, P. Rest api example generation using javadoc. *Computer Science and Information Systems 14*, 2 (2017), 447–466.

[311] RAO, A. S., GEORGEFF, M. P., ET AL. Bdi agents: From theory to practice. In *ICMAS* (1995), vol. 95, pp. 312–319.

[312] RAPOLU, B. Analytics and the Internet of Things in Aerospace: How is the IoT affecting the aerospace industry? *The Innovation Enterprise Ltd 5* (2015).

[313] RASHIDI, P., AND COOK, D. J. Adapting to resident preferences in smart environments. In *AAAI Workshop on Preference Handling* (2008), pp. 78–84.

[314] RAU, P.-L. P., HUANG, E., MAO, M., GAO, Q., FENG, C., AND ZHANG, Y. Exploring interactive style and user experience design for social web of things of chinese users: A case study in beijing. *International Journal of Human-Computer Studies 80* (2015), 24–35.

[315] RAY, P. A survey on Internet of Things architectures. *Journal of King Saud University - Computer and Information Sciences* (Oct. 2016).

[316] RAZZAQUE, M. A., MILOJEVIC-JEVRIC, M., PALADE, A., AND CLARKE, S. Middleware for internet of things: a survey. *IEEE Internet of Things Journal 3*, 1 (2016), 70–95.

[317] REMAGNINO, P., ELLIS, T., AND FORESTI, G. L. *Ambient Intelligence a Novel Paradigm*. Springer Science + Business Media, Inc., New York, NY, 2005. OCLC: 300262320.

[318] REPENNING, A., AND IOANNIDOU, A. What makes end-user development tick? 13 design guidelines. *End User Development* (2006), 51–85.

[319] ROALTER, L., KRANZ, M., AND MÖLLER, A. A middleware for intelligent environments and the internet of things. *Ubiquitous Intelligence and Computing* (2010), 267–281.

[320] ROELANDS, M., CLAEYS, L., GODON, M., GEERTS, M., FEKI, M. A., AND TRAPPENIERS, L. Enabling the masses to become creative in smart spaces. In *Architecting the Internet of things*. Springer, 2011, pp. 37–64.

[321] ROMÁN, M., HESS, C., CERQUEIRA, R., RANGANATHAN, A., CAMPBELL, R. H., AND NAHRSTEDT, K. Gaia: a middleware platform for active spaces. *ACM SIGMOBILE Mobile Computing and Communications Review 6*, 4 (2002), 65–67.

[322] RUSSELL, R. M. The CRAY-1 computer system. *Communications of the ACM 21*, 1 (1978), 63–72.

[323] SACRAMENTO, V., ENDLER, M., RUBINSZTEJN, H. K., LIMA, L. S., GONCALVES, K., NASCIMENTO, F. N., AND BUENO, G. A. Moca: A middleware for developing collaborative applications for mobile users. *IEEE Distributed systems online 5*, 10 (2004), 2–2.

[324] SADRI, F. Ambient intelligence: A survey. *ACM Computing Surveys (CSUR) 43*, 4 (2011), 36.

[325] SAHA, D., AND MUKHERJEE, A. Pervasive computing: a paradigm for the 21st century. *Computer 36*, 3 (2003), 25–31.

[326] SAMOVSKIY, D. The rise of devops, 2010.

[327] SANNER, M. F., ET AL. Python: a programming language for software integration and development. *J Mol Graph Model 17*, 1 (1999), 57–61.

[328] SAREEN, P. Cloud computing: types, architecture, applications, concerns, virtualization and role of it governance in cloud. *International Journal of Advanced Research in Computer Science and Software Engineering 3*, 3 (2013).

[329] SATYANARAYANAN, M. A catalyst for mobile and ubiquitous computing. *IEEE Pervasive Computing 1*, 1 (2002), 2–5.

[330] SAURO, J. MeasuringU: Measuring Usability with the System Usability Scale (SUS), 2011.

[331] SAVIDIS, A., ANTONA, M., AND STEPHANIDIS, C. A decision-making specification language for verifiable user-interface adaptation logic. *International Journal of Software Engineering and Knowledge Engineering 15*, 06 (2005), 1063–1094.

[332] SCHLUETER, I. The node package manager and registry. *URL: https://www. npmjs. org*.

[333] SCHMIDT, A. Programming ubiquitous computing environments. In *International Symposium on End User Development* (2015), Springer, pp. 3–6.

[334] SCHNACKENBERG, A. K., AND TOMLINSON, E. C. Organizational transparency: A new perspective on managing trust in organization-stakeholder relationships. *Journal of Management 42*, 7 (2016), 1784–1810.

[335] SEARS, R., VAN INGEN, C., AND GRAY, J. To blob or not to blob: Large object storage in a database or a filesystem? *arXiv preprint cs/0701168* (2007).

[336] SELINGER, M., SEPULVEDA, A., AND BUCHAN, J. Education and the internet of everything: How ubiquitous connectedness can help transform pedagogy. *Cisco Consulting Services and Cisco EMEAR Education Team* (2013).

[337] SHA, L., GOPALAKRISHNAN, S., LIU, X., AND WANG, Q. Cyber-physical systems: A new frontier. In *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC'08. IEEE International Conference on* (2008), IEEE, pp. 1–9.

[338] SHADBOLT, N. Ambient intelligence. *IEEE intelligent Systems 18*, 4 (2003), 2–3.

[339] SHAH, S. H., AND YAQOOB, I. A survey: Internet of things (iot) technologies, applications and challenges. In *Smart Energy Grid Engineering (SEGE), 2016 IEEE* (2016), IEEE, pp. 381–385.

[340] SHAWAR, B. A., AND ATWELL, E. Chatbots: are they really useful? In *LDV Forum* (2007), vol. 22, pp. 29–49.

[341] SHENG, Z., YANG, S., YU, Y., VASILAKOS, A., MCCANN, J., AND LEUNG, K. A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities. *IEEE Wireless Communications 20*, 6 (2013), 91–98.

[342] SHKURTI, J. Node. js clustering made easy with pm2, 2015.

[343] SIGELMAN, B. H., BARROSO, L. A., BURROWS, M., STEPHENSON, P., PLAKAL, M., BEAVER, D., JASPAN, S., AND SHANBHAG, C. Dapper, a large-scale distributed systems tracing infrastructure. Tech. rep., Technical report, Google, Inc, 2010.

[344] SKLYAROV, V. Hierarchical finite-state machines and their use for digital control. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 7*, 2 (1999), 222–228.

[345] SLETTEN, B. Resource-oriented architecture: the rest of REST. *December2009. Reference Source* (2010).

[346] SOLDATOS, J., PANDIS, I., STAMATIS, K., POLYMENAKOS, L., AND CROWLEY, J. L. Agent based middleware infrastructure for autonomous context-aware ubiquitous computing services. *Computer Communications 30*, 3 (2007), 577–591.

[347] SONG, Z., CÁRDENAS, A. A., AND MASUOKA, R. Semantic middleware for the internet of things. In *Internet of Things (IOT), 2010* (2010), IEEE, pp. 1–8.

[348] SPIVAK, N. 3d printing. In *Ethical Ripples of Creativity and Innovation*. Springer, 2016, pp. 45–52.

[349] STEFANIDI, E., DOULGERAKI, M., KOROZI, M., LEONIDIS, A., AND ANTONA, M. Designing a teacher-friendly editor for configuring the attention-aware smart classroom. In *International Conference on Human-Computer Interaction* (2016), Springer, pp. 266–270.

[350] STEPHANIDIS, C., KARAGIANNIDIS, C., AND KOUMPIS, A. Decision making in intelligent user interfaces. In *Proceedings of the 2nd international conference on Intelligent user interfaces* (1997), ACM, pp. 195–202.

[351] STETTINA, C. J., HEIJSTEK, W., AND FÆGRI, T. E. Documentation work in agile teams: the role of documentation formalism in achieving a sustainable practice. In *Agile Conference (AGILE), 2012* (2012), IEEE, pp. 31–40.

[352] STOJKOSKA, B. L. R., AND TRIVODALIEV, K. V. A review of Internet of Things for smart home: Challenges and solutions. *Journal of Cleaner Production 140* (2017), 1454–1464.

[353] STOJMENOVIC, I. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian* (2014), IEEE, pp. 117–122.

[354] STOJMENOVIC, I., AND WEN, S. The fog computing paradigm: Scenarios and security issues. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on* (2014), IEEE, pp. 1–8.

[355] STOLCKE, A., RIES, K., COCCARO, N., SHRIBERG, E., BATES, R., JURAFSKY, D., TAYLOR, P., MARTIN, R., VAN ESS-DYKEMA, C., AND METEER, M. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Dialogue 26*, 3 (2006).

[356] STRANG, T., AND LINNHOFF-POPIEN, C. A context modeling survey. In *Workshop Proceedings* (2004).

[357] STREITZ, N. A., GEI\S SLER, J., HOLMER, T., KONOMI, S., MG'OLLER-TOMFELDE, C., REISCHL, W., REXROTH, P., SEITZ, P., AND STEINMETZ, R. i-LAND: an interactive landscape for creativity and innovation. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (1999), ACM, pp. 120–127.

[358] STREITZ, N. A., ROCKER, C., PRANTE, T., VAN ALPHEN, D., STENZEL, R., AND MAGERKURTH, C. Designing smart artifacts for smart environments. *Computer 38*, 3 (2005), 41–49.

[359] SULLIVAN, J., SULLIVAN, J. W., TYLER, S. W., ET AL. Intelligent user interfaces.

[360] SUMARAY, A., AND MAKKI, S. K. A comparison of data serialization formats for optimal efficiency on a mobile platform. In *Proceedings of the 6th international conference on ubiquitous information management and communication* (2012), ACM, p. 48.

[361] SUNDMAEKER, H., GUILLEMIN, P., FRIESS, P., AND WOELFFLÉ, S. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision 3*, 3 (2010), 34–36.

[362] TAN, L., AND WANG, N. Future internet: The internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on* (2010), vol. 5, IEEE, pp. V5–376–V5–380.

[363] TANAKA, H. Keynote talkb•"1: Sensing-based information system in the era of iot, cloud services and smartphones. In *Knowledge Creation and Intelligent Computing (KCIC), International Conference on* (2016), IEEE, pp. 22–22.

[364] TANDLER, P. The beach application model and software framework for synchronous collaboration in ubiquitous computing environments. *Journal of Systems and Software 69*, 3 (2004), 267–296.

[365] TANENBAUM, A. S. *Modern operating system*. Pearson Education, Inc, 2009.

[366] TANENBAUM, A. S., AND VAN STEEN, M. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[367] THEVENIN, D., AND COUTAZ, J. Plasticity of user interfaces: Framework and research agenda. In *Interact* (1999), vol. 99, pp. 110–117.

[368] THOMA, M., MEYER, S., SPERNER, K., MEISSNER, S., AND BRAUN, T. On iot-services: Survey, classification and enterprise integration. In *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on* (2012), IEEE, pp. 257–260.

[369] THOMPSON, D., EXTON, C., GARRETT, L., SAJEEV, A., AND WATKINS, D. Distributed component object model (dcom). *Monash University, Department of Software Development, Melbourne, Australien* (1997).

[370] TIGLI, J.-Y., LAVIROTTE, S., REY, G., HOURDIN, V., CHEUNG-FOO-WO, D., CALLEGARI, E., AND RIVEILL, M. Wcomp middleware for ubiquitous computing: Aspects and composite event-based web services. *annals of telecommunications-annales des télécommunications 64*, 3-4 (2009), 197–214.

[371] TILKOV, S., AND VINOSKI, S. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing 14*, 6 (2010), 80–83.

[372] TIM JONES. Understand Representational State Transfer (REST) in Ruby, Aug. 2012.

[373] TORVALDS, L., AND HAMANO, J. Git: Fast version control system. *URL http://git-scm. com* (2010).

[374] TRUMLER, W., BAGCI, F., PETZOLD, J., AND UNGERER, T. Amunb•"autonomic middleware for ubiquitous environments applied to the smart doorplate project. *Advanced Engineering Informatics 19*, 3 (2005), 243–252.

[375] TSUNG-TE LAI, T., LIN, C.-Y., SU, Y.-Y., AND CHU, H.-H. BikeTrack: Tracking stolen bikes through everyday mobile phones and participatory sensing. In *Proceedings of the 2nd International Workshop on Sensing Applications on Mobile Phones (PhoneSense). ACM* (2011).

[376] TURNER, M., BUDGEN, D., AND BRERETON, P. Turning software into a service. *Computer 36*, 10 (2003), 38–44.

[377] UR, B., PAK YONG HO, M., BRAWNER, S., LEE, J., MENNICKEN, S., PICARD, N., SCHULZE, D., AND LITTMAN, M. L. 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* (2016), ACM, pp. 3227–3231.

[378] VAHDAT-NEJAD, H., ZAMANIFAR, K., AND NEMATBAKHSH, N. Context-aware middleware architecture for smart home environment. *International journal of smart home 7*, 1 (2013), 77–86.

[379] VAN DOORN, M., VAN LOENEN, E., AND DE VRIES, A. P. Deconstructing ambient intelligence into ambient narratives: the intelligent shop window. In *Proceedings of the 1st international conference on Ambient media and systems* (2008), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), p. 8.

[380] VAN KESTEREN, A., AND JACKSON, D. The xmlhttprequest object. *World Wide Web Consortium, Working Draft WD-XMLHttpRequest-20070618 72* (2007).

[381] VARDA, K. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Available at least as early as Jul 72* (2008).

[382] VELASTIN, S. A., BOGHOSSIAN, B. A., LO, B. P., SUN, J., AND VICENCIO-SILVA, M. A. Prismatica: toward ambient intelligence in public transport environments. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 35*, 1 (2005), 164–182.

[383] VENKATESH, A. N. Connecting the dots: Internet of things and human resource management.

[384] VERMESAN, O., BLYSTAD, L.-C., ZAFALON, R., MOSCATELLI, A., KRIEGEL, K., MOCK, R., JOHN, R., OTTELLA, M., AND PERLO, P. Internet of energy–connecting energy anywhere anytime. *Advanced Microsystems for Automotive Applications 2011* (2011), 33–48.

[385] VIANELLO, A., FLORACK, Y., BELLUCCI, A., AND JACUCCI, G. T4tags 2.0: A tangible system for supporting users' needs in the domestic environment. In *Proceedings of the TEI'16: Tenth*

International Conference on Tangible, Embedded, and Embodied Interaction (2016), ACM, pp. 38–43.

[386] VIEGA, J., AND MESSIER, M. *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More*. " O'Reilly Media, Inc.", 2003.

[387] VON NEUMANN, J. *The computer and the brain*. Yale University Press, 2012.

[388] VOULODIMOS, A. S., PATRIKAKIS, C. Z., SIDERIDIS, A. B., NTAFIS, V. A., AND XYLOURI, E. M. A complete farm management system based on animal identification using rfid technology. *Computers and Electronics in Agriculture 70*, 2 (2010), 380–388.

[389] WANG, N., PARAMESWARAN, K., SCHMIDT, D. C., AND OTHMAN, O. The design and performance of meta-programming mechanisms for object request broker middleware. In *COOTS* (2001), vol. 1, pp. 103–118.

[390] WANG, X. H., ZHANG, D. Q., GU, T., AND PUNG, H. K. Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on* (2004), Ieee, pp. 18–22.

[391] WANSTRATH, C. mustache (5)–logic-less templates., 2010.

[392] WEBER, W., RABAEY, J., AND AARTS, E. H. *Ambient intelligence*. Springer Science & Business Media, 2005.

[393] WEISER, M. The Computer for the 21 st Century. *Scientific american 265*, 3 (1991), 94–105.

[394] WG01-IERC, A. Internet of Things Applications, Oct. 2015.

[395] WILCOX, B. Chatscript, 2011.

[396] WILLIAMS, J. D., KAMAL, E., ASHOUR, M., AMR, H., MILLER, J., AND ZWEIG, G. Fast and easy language understanding for dialog systems with microsoft language understanding intelligent service (luis). In *SIGDIAL Conference* (2015), pp. 159–161.

[397] WIRTZ, H., RÜTH, J., SERROR, M., BITSCH LINK, J. Á., AND WEHRLE, K. Opportunistic interaction in the challenged internet of things. In *Proceedings of the 9th ACM MobiCom workshop on Challenged networks* (2014), ACM, pp. 7–12.

[398] WULF, V., AND JARKE, M. The economics of end-user development. *Communications of the ACM 47*, 9 (2004), 41–42.

[399] XUE, W., PUNG, H., NG, W., AND GU, T. Data management for context-aware computing. In *Embedded and Ubiquitous Computing, 2008. EUC'08. IEEE/IFIP International Conference on* (2008), vol. 1, IEEE, pp. 492–498.

[400] YANG, L., YANG, S., AND PLOTNICK, L. How the internet of things technology enhances emergency response operations. *Technological Forecasting and Social Change 80*, 9 (Nov. 2013), 1854–1867.

[401] YASHIRO, T., KOBAYASHI, S., KOSHIZUKA, N., AND SAKAMURA, K. An internet of things (iot) architecture for embedded appliances. In *Humanitarian Technology Conference (R10-HTC), 2013 IEEE Region 10* (2013), IEEE, pp. 314–319.

[402] YAU, S. S., HUANG, D., GONG, H., AND SETH, S. Development and runtime support for situation-aware application software in ubiquitous computing environments. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International* (2004), IEEE, pp. 452–457.

[403] YAU, S. S., WANG, Y., AND KARIM, F. Development of situation-aware application software for ubiquitous computing environments. In *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International* (2002), IEEE, pp. 233–238.

[404] YI, S., HAO, Z., QIN, Z., AND LI, Q. Fog Computing: Platform and Applications. IEEE, pp. 73–78.

[405] YI, S., LI, C., AND LI, Q. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data* (2015), ACM, pp. 37–42.

[406] YORK, J., AND PENDHARKAR, P. C. Humanb•"computer interaction issues for mobile computing in a variable work context. *International Journal of Human-Computer Studies 60*, 5 (2004), 771–797.

[407] ZAHARIADIS, T., PAPADIMITRIOU, D., TSCHOFENIG, H., HALLER, S., DARAS, P., STAMOULIS, G. D., AND HAUSWIRTH, M. Towards a future internet architecture. In *The Future Internet Assembly* (2011), Springer, pp. 7–18.

[408] ZAMORA-IZQUIERDO, M. A., SANTA, J., AND GÓMEZ-SKARMETA, A. F. An integral and networked home automation solution for indoor ambient intelligence. *IEEE Pervasive Computing 9*, 4 (2010), 66–77.

[409] ZDRAVKOVIĆ, M., TRAJANOVIĆ, M., SARRAIPA, J., JARDIM-GONÇALVES, R., LEZOCHE, M., AUBRY, A., AND PANETTO, H. Survey of internet-of-things platforms. In *6th International Conference on Information Society and Techology, ICIST 2016* (2016), vol. 1, pp. 216–220.

[410] ZHANG, C., AND KOVACS, J. M. The application of small unmanned aerial systems for precision agriculture: a review. *Precision agriculture 13*, 6 (2012), 693–712.

[411] ZHENG, Z., MA, H., LYU, M. R., AND KING, I. Collaborative web service qos prediction via neighborhood integrated matrix factorization. *IEEE Transactions on Services Computing 6*, 3 (2013), 289–299.

[412] ZHI, J., GAROUSI-YUSIFOĞLU, V., SUN, B., GAROUSI, G., SHAHNEWAZ, S., AND RUHE, G. Cost, benefits and quality of software development documentation: A systematic mapping. *Journal of Systems and Software 99* (2015), 175–198.

[413] ZHONG, D., LV, H., HAN, J., AND WEI, Q. A Practical Application Combining Wireless Sensor Networks and Internet of Things: Safety Management System for Tower Crane Groups. *Sensors 14*, 8 (July 2014), 13794–13814.

[414] ZHOU, C., CHIA, L.-T., AND LEE, B.-S. Daml-qos ontology for web services. In *Web Services, 2004. Proceedings. IEEE International Conference on* (2004), IEEE, pp. 472–479.

[415] ZHOU, M., ZHANG, R., ZENG, D., AND QIAN, W. Services in the Cloud Computing era: A survey. IEEE, pp. 40–46.

[416] ZUE, V. W., AND GLASS, J. R. Conversational interfaces: Advances and challenges. *Proceedings of the IEEE 88*, 8 (2000), 1166–1180.

Appendix A
Publications

The research activity related to this thesis has so far produced the following publications (ordered by publication date).

(1) Margetis, G., Leonidis, A., Antona, M., & Stephanidis, C. (2011). Towards Ambient Intelligence in the Classroom. In C. Stephanidis (Ed.), Universal Access in Human-Computer Interaction. Applications and Services - Volume 8 of the combined Proceedings of the 14th International Conference on Human-Computer Interaction (HCI International 2011), Orlando, FL, USA, 9-14 July, pp. 577-586. Berlin Heidelberg: Lecture Notes in Computer Science Series of Springer (LNCS 6768, ISBN: 978-3-642-21656-5).

(2) Antona, M., Leonidis, A., Margetis, G., Korozi, M., Ntoa, S., & Stephanidis, C. (2011). A Student-Centric Intelligent Classroom. In D. Keyson et al. (Eds.), Proceedings of the 2nd International Joint Conference in Ambient Intelligence (AmI 2011), 16-18 November, Amsterdam, The Netherlands (pp. 248-252). Berlin Heidelberg, Germany: Springer [LNCS: 7040].

(3) Leonidis, A., Margetis, G., Antona, M., & Stephanidis, C. (2011). An Intelligent Task Assignment and Personalization System for Students' Online Collaboration. In C. Stephanidis (Ed.), Universal Access in Human-Computer Interaction. Applications and Services - Volume 8 of the combined Proceedings of the 14th International Conference on Human-Computer Interaction (HCI International 2011), Orlando, FL, USA, 9-14 July, pp. 548-557. Berlin Heidelberg: Lecture Notes in Computer Science Series of Springer (LNCS 6768, ISBN: 978-3-642-21656-5).

(4) Leonidis, A., Korozi, M., Margetis, G., Ntoa, S., Papagiannakis, H., Antona, M., & Stephanidis, C. (2012). A Glimpse into the Ambient Classroom. Bulletin of the IEEE Technical Committee on Learning Technology, 14 (4), 3-6.

(5) Korozi, M., Leonidis, A., Margetis, G., Koutlemanis, G., Zabulis, X., Antona, M., & Stephanidis, C. (2012). Ambient Educational Mini-games. In G. Tortora, S. Levialdi & M. Tucci (Eds.), Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI 2012), Capri Island (Naples), Italy, 21-25 May (802-803). New York: ACM Press.

(6) Kapnas, G., Leonidis, A., Korozi, M., Ntoa, S., Margetis, G., & Stephanidis, C. (2013). A Museum Guide Application for Deployment on User-Owned Mobile Devices. In C. Stephanidis

(Ed.), HCI International 2013 - Posters' Extended Abstracts, Part II - Volume 29 of the combined Proceedings of HCI International 2013 (15th International Conference on Human-Computer Interaction), Las Vegas, Nevada, USA, 21-26 July, pp. 253-257. Berlin Heidelberg: Communications in Computer and Information Science (CCIS 374, ISBN: 978-3-642-39475-1).

(7) Savvaki, C., Leonidis, A., Paparoulis, G., Antona, M., Stephanidis, C. (2013). Designing a Technology-Augmented School Desk for the Future Classroom In C. Stephanidis (Ed.), HCI International 2013 - Posters' Extended Abstracts, Part II - Volume 29 of the combined Proceedings of HCI International 2013 (15th International Conference on Human-Computer Interaction), Las Vegas, Nevada, USA, 21-26 July, pp. 681-685. Berlin Heidelberg: Communications in Computer and Information Science (CCIS 374, ISBN: 978-3-642-39475-1).

(8) Leonidis, A., Korozi, M., Margetis, G., Grammenos, D., & Stephanidis, C. (2013). An Intelligent Hotel Room. In J.C. Augusto, R. Wichert, R. Collier, D. Keyson, A.A. Salah, & A-H. Tan (Eds.), Proceedings of the 4th International Joint Conference on Ambient Intelligence (AmI-2013), Dublin, Ireland, 3-5 December (pp. 241-246). Heidelberg, Germany: Springer (LNCS 8309).

(9) Ntoa, S., Leonidis, A., Korozi, M., Papadaki, E., Margetis, G., Antona, M., & Stephanidis, C. (2015). Analysis and Design of Three Multimodal Interactive Systems to Support the Everyday Needs of Children with Cognitive Impairments. In M. Antona & C. Stephanidis (Eds.), Universal Access in Human-Computer Interaction. Access to Learning, Health and Well-Being ? Volume 9 of the combined Proceedings of the 17th International Conference on Human-Computer Interaction (HCI International 2015), Los Angeles, CA, USA, 2-7 August, pp. 637-648. Berlin Heidelberg: Lecture Notes in Computer Science Series of Springer (LNCS 9177, ISBN: 978-3-319-20684-8).

(10) Leonidis, A., Antona, M., & Stephanidis, C. (2015). Enabling Programmability of Smart Learning Environments by Teachers. In N. Streitz & P. Markopoulos (Eds.), Distributed, Ambient, and Pervasive Interactions, Volume 21 of the combined Proceedings of the 17th International Conference on Human-Computer Interaction (HCI International 2015), Los Angeles, CA, USA, 2-7 August, pp. 62-73. Berlin Heidelberg: Lecture Notes in Computer Science Series of Springer (LNCS 9189, ISBN: 978-3-319-20803-9).

(11) Louloudakis, N., Leonidis, A., & Stephanidis, C. (2016). AmITest: A Testing Framework for Ambient Intelligence Learning Applications. . In S. White, H. Mannaert & J. L. Mauri (Eds.), Proceedings of the Eighth International Conference on Mobile, Hybrid, and On-line Learning (eLmL 2016), Venice, Italy, 24-28 April (pp. 76-82). IARIA XPS Press - ISBN: 978-1-61208-471-8 [BEST PAPER AWARD]

(12) Stefanidi, E., Doulgeraki, M., Korozi, M., Leonidis, A., & Antona, M. (2016). Designing a Teacher-Friendly Editor for Configuring the Attention-Aware Smart Classroom. In C. Stephanidis (Ed.), HCI International 2016 - Posters' Extended Abstracts, Part II, Volume 28 of the combined Proceedings of HCI International 2016 (18th International Conference on Human-

Computer Interaction), Toronto, Canada, 17-22 July (pp. 266-270). Berlin Heidelberg: Communications in Computer and Information Science (CCIS 618).

(13) Barka, A., Leonidis, A., Antona, M., & Stephanidis, C. (2017). A Unified Interactive System for Controlling a Smart Home. In the Proceedings of the ACM Europe Celebration of Women in Computing (womENcourage 2017), Barcelona, Spain. [On-line]. Available at: https:womencourage.acm.orgwp-contentuploads201707womENcourage_2017_paper_23. pdf?189db0

(14) Leonidis, A., Arampatzis, D., Louloudakis, N., & Stephanidis, C. (2017). The AmI-Solertis System: Creating User Experiences in Smart Environments. 5th International Workshop on Pervasive and Context-Aware Middleware (PerCAM 17), in the context of the 13th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2017), 9-11 October, Rome, Italy.

(15) Korozi, M., Leonidis, A., Antona, M., & Stephanidis,C. (2017). Lector: Towards Reengaging Students in the Educational Process inside Smart Classrooms. 9th international conference on Intelligent Human Computer Interaction (IHCI 2017), 11-13 December, Evry, Paris, France.

(16) Leonidis, A., Antona, M., & Stephanidis, C. (2017). Using Contextual Knowledge to Resume Human-Agent Conversations when Programing the Intelligence of Smart Environments. In the Proceedings of the Workshop on Conversational Interruptions in Human-Agent Interactions (CIHAI), in the context of the 17th International Conference on Intelligent Virtual Agents (IVA 2017), 27-30 August, Stockholm, Sweden. CEUR Workshop Proceedings (CEUR-WS.org).

(17) Partarakis, N., Grammenos, D., Margetis, G., Zidianakis, E., Drossis, G., Leonidis, A., Metaxakis, G., Antona, M., & Stephanidis, C. (2017). Digital cultural heritage experience in Ambient Intelligence. In M. Ioannides, N. Magnenat-Thalmann, & G. Papagiannakis (Eds.), Mixed Reality and Gamification for Cultural Heritage (pp. 473-505). Switzerland: Springer International Publishing AG.

(18) Korozi, M., Antona, M., Ntagianta, A., Leonidis, A., & Stephanidis, C. (2017). LECTORstudio: Creating Inattention Alarms and Interventions to Reengage the Students in the Educational Process. 10th annual International Conference of Education, Research and Innovation, 16-18 November, Seville, Spain.

(19) Stefanidi, E., Korozi, E. , Leonidis, A., Doulgeraki, M., & Antona, M. (2018). Educator-Oriented Tools for Managing the Attention-Aware Intelligent Classroom. In the Proceedings of The 10th International Conference on Mobile, Hybrid, and On-line Learning (eLmL 2018), Rome.

(20) Stefanidi, E., Leonidis, A., & Antona, M. (2018). Programming Intelligent Environments in Natural Language: An Extensible Interactive Approach. In Petra 2018 (11th PErvasive Technologies Related to Assistive Environments (PETRA) Conference), Corfu, Greece, 26-29 June. (submitted)

Appendix B
Acronyms

AAL	Ambient Assisted Living
ADS	Automated Deployment Services
AI	Artificial Intelligence
AIOTI	Alliance of Internet of Things Innovation
AJAX	Asynchronous JavaScript And XML
AmI	Ambient Intelligence
AOP	Aspect-Oriented Programming
API	Applications Programming Interface
AR	Augmented Reality
ASLib	AmI-Solertis Standard Library
ASP	Answer set programming
BDI	Belief-Desire-Intention
BLE	Bluetooth Low-Energy
CAGR	Compound Annual Growth Rate
CAMUS	Context-Aware Middleware for Ubiquitous computing Systems
CI	Continuous Integration
CIH	Conversational Interruption Handler
CLR	Common Language Runtime

CoAP Constrained Application Protocol

COBASEN COntext BAsed Search ENgine

COMPaaS Cooperative Middleware Platform as a Service

CORBA Common Object Request Broker Architecture

CoT Cloud of Things

CPS Cyber-Physical System

CPU Central Processing Unit

CRUD Create-Retrieve-Update-Delete

CXaaS Context-as-a-Service

DAI Distributed Artificial Intelligence

DIM Dynamic Integration Middleware

DiY Do-it-Yourself

DLL Dynamic Link Library

DNS Domain Name System

DNS-SD DNS Service Discovery

DSL Domain Specific Language

EcoDiF Web Ecosystem of Physical Devices

EDA Electro-Dermal Activity

EEML Extended Environments Markup Language

ENS Event Notification Service

ETS Engineering Tool Software

EUD End-User Development

FAmINE FORTH AmI Network Environment

FSM Finite-State Machine

FQDN Fully Qualified Domain Name

GPS	Global Positioning System
GSMA	Groupe Speciale Mobile Association
GSN	Global Sensor Network
GUI	Graphical User Interface
HATEOAS	Hypermedia as the Engine of Application State
HVAC	Humidity-Ventilation-Air-Conditioning
HCI	Human Computer Interaction
HDS	Hot Deployment Service
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
H2H	Human-to-Human
H2T	Human-to-Technology
IaaS	Infrastructure-as-a- Service
ICT	Information and Communication Technology
IDL	Interface Definition Language
iDorm	Intelligent Dormitory
IIoT	Industrial Internet of Things
IETF	Internet Engineering Task Force
IoE	Internet of Everything
IoT	Internet of Things
ISP	Internet Service Provider
ISTAG	Information and Communication Technologies Advisory Group
IUI	Intelligent User Interface
JAC	Jadex Active Components
JADE	Java Agent Development Environment

JAX-RS Java API for RESTful Web Services

JRE Java Runtime Environment

JS Javascript

JSDoc Javascript Documentation

JSON JavaScript Object Notation

JXTA Juxtapose

LoRaWAN LoRaWAN

MAR Mobile Augmented Reality

MBaaS Mobile-Backend-as-a-Service

mDNS multicast Domain Name System

MoCA Mobile Collaboration Architecture

MOM Message-Oriented Middleware

MOSDEN Mobile Sensor Data Processing Engine

MQTT Message Queue Telemetry Transport

M2M Machine-to-Machine

NIST National Institute of Standards and Technology

NLP Natural Language Processing

NLU Natural Language Understanding

NPM Node Package Manager

OAS OpenAPI Specification

OOP Object-Oriented Programming

OS Operating System

OSGi Open Service Gateway Initiative

OWL Web Ontology Language

OWL-S Web Ontology Language - Semantic

OWL-DL Web Ontology Language - Description Logic

PaaS Platform-as-a-Service

PC Personal Computer

PDA Personal Digital Assistant

POJO Plain Old Java Object

POS Part-of-Speech

P2M Person-to-machine

P2P person-to-person

P2P Peer-to-peer

QoS Quality of Service

RCSM Reconfigurable Context-Sensitive Middleware

REST Representational State Transfer

RFID Radio-frequency Identification

ROS Robot Operating System

RPC Remote Procedure Call

SA Situation Awareness

SaaS Software-as-a-Service

SBC Single-Board Computer

SDK Software Development Kit

SDL Service Description Language

SensorML Sensor Model Language

SIoT Social Internet-of-Things

SO Smart Object

SOA Service-Oriented Architecture

SOAP Simple Object Access Protocol

SPARQL SPARQL Protocol and RDF Query Language

SWE Sensor Web Enablement

SUS Standard Usability Scale

S2aaS Sensing-as-a-Service

TCP/IP Transmission Control Protocol / Internet Protocol

T2T Technology-to-Technology

UbiComp Ubiquitous Computing

UbiMAS Ubiquitous Mobile Agent System

UI User Interface

uID Ubiquitous ID

uPnP Universal Plug and Play

URI Uniform Resource Identifier

URL Uniform Resource Locator

VR Virtual Reality

VPL Visual Programming Language

Wifi WiFi

WiMax WiMax

WPAN WPAN

WS-BPEL Web Service Business Process Execution Language

WSAN Wireless Sensors and Actuators Network

WSDL Web Service Definition Language

WSN WSN

W3C World Wide Web Consortium

XEP XMPP Extension

XHTTP XMLHttpRequest

XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
YAML	Yet Another Modelling Language
Z-Wave	Z-Wave
Zeroconf	Zero Configuration
ZigBee	Zigbee
4G	4th generation of broadband cellular network technology
5G	5th generation of broadband cellular network technology

Appendix C
Evaluation Scenario

Task 1
Login to your account using the following credentials:
- *e-mail: ami@ics.forth.gr*
- *password: 123456*

Task 2
You have been provided with a new set of Philips Hue lamps and you are asked to integrate them to the ecosystem. Create a new AmI Artifact named PhilipsHueController to manipulate lights. Write the code (in general.controller.js) and expose (through index.js) two functions that turn-on the light to a specific color and turn-off the light. Ensure that documentation is written correctly.
- *Sample code to perform a REST request can be found at file "PhilipsHueCodeSample.js"*
- *AmI-Solertis relies on code documentation to automatically extract the API of any imported artefact*

Task 3
In order to use the newly created artifact in scripts that specify the behavior of the environment, you firstly have to integrate it. Compress (i.e., zip) and import the PhilipsHueController AmI Artifact in the AmI-Solertis ecosystem.

Task 4
Before launching your bridge that controls the Philips Hue Lamps (i.e., PhilipsHueController), you have to deploy it (i.e., download and install) in a host.
- *Host: Dell PC 1 - Living room*
- *Port: 3680*
- *Context name: livingroomLights*

Task 5
Execute the PhilipsHueController on that host (Dell PC 1 - Living room).

Task 6
Explore the AmITV AmI Artifact and find how many and which operations it offers.

Task 7
You are asked create a new "intelligent" behaviour (i.e., AmI script) that will control the environment as follows: *"AmiTV will be unlocked and the Philips Hue lamps will be turned-on at color #446677 and intensity 78% as soon as motion is detected in the living room"*.

Task 8
Before launching the new behavior (i.e., MultimediaAI), you have to deploy it (i.e., download and install) in a host.
- *Host: Dell PC 1 - Living room*
- *Port: 3681*
- *Context name: livingroomMultimediaAI*

Task 9
Execute the MultimediaAI on that host (Dell PC 1 - Living room).

Task 10
How many times has the MultimediaAI script captured a "motion detected event" and it instructed the lamps to turn-on?