University of Crete

Computer Science Department

TOOL SUPPORT FOR DESIGNING AND GENERATING ACCESSIBLE USER INTERFACES IN AMBIENT INTELLIGENCE ENVIRONMENTS

by

SOKRATIS KARTAKIS

MASTER'S THESIS

OCTOBER 2008

To my parents Christos and Ourania,

and

my brothers George and Angelos

Acknowledgements

For their assistance, encouragement and support, I am grateful to my supervisor Prof. Constantine Stephanidis.

I would also like to thank all my colleagues at the HCI Laboratory of ICS-FORTH, and in particular Dr. Margherita Antona, Dimitris Grammenos, John Georgalis, Anthony Katzourakis and Dionisia Aravantinou.

Abstract

In the next few years, it will be feasible to control a large number of standard electrical and electronic devices residing in the home environment through computing technologies. The vision of Ambient Intelligence (AmI) promises to deliver more natural ways for controlling these devices (e.g., by using hand gestures and speech) or even offer proactive, non user-initiated, controls (e.g., a “smart” room knows when and how to change the state of each device in it). However, the need will continue to be felt for highly usable graphical user interfaces (GUIs) that will allow direct and universally accessible user control, as well as quick and intuitive overviews of the current state of the computer-controllable environment. Additionally, the provision of appropriate development tools to facilitate the easy and quick deployment of smart home technologies will be of paramount importance. This thesis proposes two tools, the AmI Designer and the AmI Player, that have been specifically created to address this need through automatic generation of accessible GUIs in AmI environments without the need of programming.

The AmI Designer is a graphical editor which creates descriptions of a designed application in XML. The Designer includes different widgets available to design a user interface. These widgets are detected dynamically from the editor, and developers can program their own widgets. The AmI Designer communicates with the Middleware of an AmI environment, is continuously informed about the available services of a room and can find the Middleware signals. Therefore, it supports defining how the application manipulates the devices in the room and how the external environment interacts with the GUI.

The AmI Player uses the export XML description of the AmI Designer to generate GUIs ready for run-time. It sends and receives messages and signals from the middleware, calls services and uses devices. The produced GUIs can be manipulated through several input devices.

These tools can be extended with additional interaction styles and services, as well as interaction platforms. The generated user interfaces integrate by design non-visual feedback and a scanning mechanism. Therefore, they are accessible and usable for all end users, including hand-motor and visual impaired users, children, elders, and inexperienced users. A case study is also presented in this thesis where the AmI Designer and the AmI Player have been used in the design of a light control application in a smart environment. The results of the conducted evaluation confirm the usefulness and usability of the tools themselves, as well as the accessibility and usability of the produced user interfaces.

Περίληψη

Τα επόμενα χρόνια, θα είναι εφικτός ο χειρισμός ενός μεγάλου αριθμού τυποποιημένων ηλεκτρικών και ηλεκτρονικών συσκευών που βρίσκονται στο οικιακό περιβάλλον μέσω υπολογιστικών τεχνολογιών. Το όραμα της Διάχυτης Νοημοσύνης υπόσχεται να δώσει φυσικότερους τρόπους για τον έλεγχο αυτών των συσκευών (π.χ., με την αναγνώριση χειρονομιών και ομιλίας) ή ακόμα και να προσφέρει «έξυπνες» λειτουργίες (π.χ., ένα «έξυπνο» δωμάτιο ξέρει πως και πότε να αλλάξει την κατάσταση κάθε συσκευής που βρίσκεται μέσα σε αυτό). Εντούτοις, η ανάγκη θα είναι εξίσου αισθητή για υψηλής ευχρηστίας γραφικές διεπαφές που θα επιτρέπουν τον άμεσο και καθολικά προσβάσιμο έλεγχο από τον χρήστη, καθώς επίσης και γρήγορες και πιο διαισθητικές επισκοπήσεις της τρέχουσας κατάστασης του ελεγχόμενου περιβάλλοντος από υπολογιστή. Επιπλέον, η παροχή κατάλληλων εργαλείων ανάπτυξης, για τη διευκόλυνση της εύκολης και γρήγορης ανάπτυξης τεχνολογιών έξυπνου χώρου, θα είναι ύψιστης σημασίας. Αυτή η διατριβή προτείνει δύο εργαλεία, το AmI Designer και το AmI Player, τα οποία έχουν δημιουργηθεί για να καλύψουν αυτή τη συγκεκριμένη ανάγκη μέσω της αυτόματης παραγωγής καθολικά προσβάσιμων γραφικών διεπαφών για περιβάλλοντα διάχυτης νοημοσύνης χωρίς τη χρήση προγραμματισμού.

Ο AmI Designer είναι ένας γραφικός σχεδιαστής ο οποίος δημιουργεί την περιγραφή μιας σχεδιασμένης εφαρμογής σε XML. Ο σχεδιαστής περιέχει διάφορα γραφικά αντικείμενα, τα οποία είναι διαθέσιμα για τη σχεδίαση της διεπαφής. Αυτά τα γραφικά αντικείμενα ανακαλύπτονται δυναμικά από το σύστημα, δίνοντας τη δυνατότητα σε προγραμματιστές να φτιάξουν νέα αντικείμενα. Ο AmI Designer επικοινωνεί με το ενδιάμεσο επίπεδο ενός περιβάλλοντος Διάχυτης Νοημοσύνης και πληροφορείται συνεχώς για τις διαθέσιμες υπηρεσίες κάποιου δωματίου, καθώς και για τα σήματα που υπάρχουν σε αυτό. Άρα, ορίζει πως η εφαρμογή θα διαχειρίζεται τις συσκευές στο δωμάτιο και πως το εξωτερικό περιβάλλον θα αλληλεπιδρά με τη γραφική διεπαφή.

Ο AmI Player χρησιμοποιεί την παραγόμενη σε XML περιγραφή από τον AmI Designer και αναπαράγει τη γραφική διεπαφή που είναι έτοιμη για χρήση. Αυτή στέλνει και λαμβάνει μηνύματα και σήματα από το ενδιάμεσο επίπεδο, καλεί υπηρεσίες και διαχειρίζεται τις συσκευές. Η παραγόμενη γραφική διεπαφή μπορεί να χρησιμοποιηθεί με διάφορες συσκευές εισόδου.

Τα εργαλεία που αναπτυχθήκαν στα πλαίσια αυτής της διατριβής μπορούν να επεκταθούν με επιπρόσθετα αλληλεπιδραστικά χαρακτηριστικά και υπηρεσίες, καθώς επίσης επιπρόσθετες αλληλεπιδραστικές πλατφόρμες. Οι παραγόμενες διεπαφές ενσωματώνουν από το σχεδιασμό τους μη-οπτικά ερεθίσματα, καθώς και ένα μηχανισμό διαδοχικής σάρωσης. Επομένως, είναι προσβάσιμες και χρηστικές από όλους τους τελικούς χρήστες, συμπεριλαμβανομένων των χρηστών με κινησιακές αναπηρίες και προβλήματα όρασης, παιδιών, ηλικιωμένων, καθώς επίσης και μη έμπειρων χρηστών.

Επίσης, σε αυτή τη διατριβή παρουσιάζεται μια εφαρμογή όπου ο AmI Designer και ο AmI Player έχουν χρησιμοποιηθεί για τον σχεδιασμό μιας εφαρμογής ελέγχου των φώτων σε ένα έξυπνο περιβάλλον.

Τα αποτελέσματα της πραγματοποιημένης αξιολόγησης επιβεβαιώνουν τη χρησιμότητα και ευχρηστία των εργαλείων, καθώς επίσης και την προσβασιμότητα και ευχρηστία των παραγόμενων διεπαφών.

Table of Contents

List of Figures

List of Tables

1. Introduction

This thesis addresses the development of accessible user interfaces in Ambient Intelligence environments, and proposes tool-based support towards achieving:

- easy and quick automatic generation of user interfaces for smart home applications
- automatic integration of the developed interfaces in smart home environments without the need of further programming
- accessibility and usability of the developed user interfaces for all final users, including disabled and elderly people without the need of any further adaptation, thus providing a convenient way for final users to manipulate diverse devices in the environment through consistent interaction
- extensibility of the tools themselves to additional interaction styles and additional services, as well as potentially additional interaction platforms.

Towards setting the background of the performed work and of the underlying approach, this Chapter introduces the main technological dimensions and characteristics of Ambient Intelligence, and provides an overview of smart home environments and related applications. Subsequently, is discusses the social dimensions of AmI, and outlines current challenges related to user interface development in AmI environments.

The remaining Chapters of this thesis are organized as follows:

Chapter 2 discusses related work in the domain of graphical user interface editors and generators. While a very wide variety of UI design and generation tools is currently available, no tool currently satisfies the requirements posed by the development of accessible user interfaces in an AmI environment.

Chapter 3 presents in details the design of the proposed tools, and explains their rationale and functionality.

Chapter 4 presents relevant aspects of the tool's implementation.

Chapter 5 reports a detailed case study where the tools have been used in the development of an accessible light control application in a smart home environment.

Chapter 6 reports on the usability evaluation of both the design tool and the light control application, confirm the benefits and added value of the proposed approach in terms of both (i) support to the design and development process, and (ii) accessibility and usability of the deriving user interfaces for a diverse population of end-users.

Finally, Chapter 7 offers some conclusions and discusses future work.

1.1 Ambient Intelligence

Ambient intelligence (AmI) is a burgeoning research field that has potential for great impact in the future. The term "ambient" is defined in the Merriam-Webster's dictionary (Mish and Morse 1999 [34]) as "existing or present on all sides". The term Ambient Intelligence is defined by the Advisory Group to the European Community's Information Society Technology Program (ISTAG [46]) as "the convergence of ubiquitous computing, ubiquitous communication, and interfaces adapting to the user" (Gupta 2003 [30]). The term “ubiquitous” should also be defined, as the concept of ubiquity is intrinsic in AmI. Ubiquity involves the idea that something exists or is everywhere at the same time on a constant level, for example, hundreds of sensors placed throughout a household. This concept is important when trying to understand the future implications that AmI will have on the human environment. Indeed, it is anticipated that the onset of AmI will revolutionize business, government, and everyday life in a manner similar to the personal computing revolution of the 1980s and 1990s. As more and more attention and effort is directed towards developing AmI to its full potential, the question of how people will all be affected by it, both positively and negatively, requires consideration.

The objective of AmI is to broaden the interaction between human beings and digital information technology through the use of ubiquitous computing devices. Conventional computing primarily involves user interfaces (UIs) such as keyboard, mouse, and visual display unit, while the large ambient space that encompasses the user is not utilized. AmI, on the other hand, uses this space in the form of, for example, shape, movement, scent and sound recognition or output. Again the example of the all-encompassing sensors in households is pertinent. These information media become possible through new types of interfaces and will allow drastically simplified and more intuitive use of devices. Wireless networks will be the dominant technology for communication between these devices. The combination of simplified use and the ability to communicate will eventually result in increased efficiency for users and will, therefore, create value, leading to a higher degree of ubiquity of computing devices. Examples of such devices range from common items such as pens, watches, and household appliances to sophisticated computers and production equipment.

This Chapter examines the digital information technology behind AmI, its business value and potential weaknesses, as well as the changes that AmI introduces to personal and professional lives.

AmI is comprised of three main components: ubiquitous computing, ubiquitous communication, and user adaptive interfaces.

1.1.1 Ubiquitous computing

Weiser (1991 [37]) coined the term "ubiquitous computing", referring to omnipresent computers that serve people in their everyday lives at home and at work, functioning invisibly and unobtrusively in the background and freeing people to a large extent from tedious routine tasks. The general working definition of ubiquitous computing technology is any computing technology that permits human interaction away from a single workstation. This includes pen-based technology, hand-held or portable devices, large-scale interactive

screens, wireless networking infrastructure, and voice or vision technology (Abowd 2004 [20]).

In its ultimate form, ubiquitous computing means that any computing device, while moving with the user, can build incrementally dynamic models of its various environments and configure its services accordingly. The devices will be able to either "remember" past environments they operated in, or proactively build up services in new environments (Lyytinen and Yoo 2002 [32]). In its 1999 vision statement, the European Union's Information Society Technologies Program Advisory Group (ISTAG) used the term "ambient intelligence" in a similar fashion to describe a vision where "people will be surrounded by intelligent and intuitive interfaces embedded in everyday objects around us and an environment recognizing and responding to the presence of individuals in an invisible way" (Ahola 2001 [21]).

One of the most significant challenges in AmI/ubiquitous computing technologies is to create user-friendly interfaces. Developing interfaces for ubiquitous computing is a rather new field. Human computer interaction designers are currently struggling to establish usable standards for these technologies (Bohn et al. 2004, Davis 2002 [25], [28]).

1.1.2 Ubiquitous communication

Today, numerous objects are equipped with computers, i.e., the environment already exhibits a relatively high level of ubiquitous computing. However, in most cases, these computers do not operate at their full potential, since they are unable to communicate with each other. A major change in the corporate and home environments that will promote ubiquitous communication and, thereby, ubiquitous computing, is the introduction and expansion of wireless network technology, which enables flexible communication between interlinked devices that can be stationed in various locations or can be portable. To implement wireless technology on a wide level, however, the wireless hardware itself must meet several criteria on the one hand, while, on the other hand, easy integration and administration as well as security of the network must be ensured.

The following list presents different wireless technologies that are available on the market or currently under development. Middleware use such technologies to make the communication between the smart devices and applications. The main factors to be considered when selecting the technology are physical size, range of operation, data transfer rate, and price. Physical size may become a critical factor due to space constraints, especially in mobile applications. Following Moore's law, wireless hardware has been decreasing in size considerably and can nowadays be placed on small microchips. The University of California, Berkeley demonstrated this by building a wireless sensor only 2.0 by 2.5 mm in size that can transmit data at a speed of 19,200 kbit/s over a range of up to 40 feet (Yang 2003 [40]).

- Wireless LAN (W-LAN)
- Bluetooth technology
- High rate W-PANs.
- Low power W-PANs
- Wireless body area networks (BANs [42])
- Radio frequency identification (RFID

As indicated earlier, the mere existence of wireless technology does not suffice to promote ubiquitous communication and computing. Network integration and administration must be made easier and network security guaranteed. To combine computers and networks efficiently and effectively, it is crucial that they can communicate without need for data conversion or translation. This ability is referred to as network interoperability and is imperative for the success of AmI. Companies engaged in AmI research are, therefore, making efforts to agree on communication standards that will ensure compatibility of different objects in the network (Ailisto *et al*. 2003 [22]).

Network administration is facilitated by minimizing the effort required for setting up networks. The introduction of mobile *ad hoc* networks (MANETs) is an important step in this direction. A MANET uses the wireless technologies described above, but is more flexible than conventional networks, since the routers are included in the mobile nodes instead of being fixed and have the ability to configure themselves. This provides the network with great flexibility due to its ability to adapt automatically to a changing network environment (Corson and Macker 1999 [27]).

1.1.3 Adaptive and multimodal interfaces

AmI will profoundly change human interaction with technology. Two basic features will charaterise user interfaces in AmI environments: multimodality and adaptivity.

Multimodality goes beyond the traditional keyboard and mouse to improve human interaction with technology by making it more intuitive, efficient, and secure. Multimodal interfaces allow the computer to know and sense far more about a person, the situation the person is in, the environment, and related objects than traditional interfaces can. They encompass interfaces that create a perceptive computer environment rather than one that relies solely on active and comprehensive user input. Multimodal interfaces can include the following modalities:

- Visual recognition (e.g., face, 3D gesture, and location) and output
- Sound recognition (e.g., speech, melody) and output
- Scent recognition and output
- Tactile recognition and output
- Other sensor technologies

Traditional user interfaces like PC-controlled touch screens in a company environment and user interfaces in portable units such as PDAs or cellular phones can also become multimodal. Key to multimodality is the ease of use, and in particular the ability to personalize and adapt automatically to particular user behavior patterns (profiling) and different situations (context awareness) by means of intelligent algorithms. With increasing amounts of information being transferred across networks, greater concern arises regarding security and privacy, especially in the light of recent reports on intrusion by viruses and worms, which demonstrate the vulnerability of network technology. A weak spot in networks regarding privacy has been password queries. People with infected machines can be 'watched' by others while typing in passwords, use passwords that are too simple, or use passwords that are too easily relatable to the person, such as names of children, pets, and

the like. in AmI, automatic user identification by, for example, face or voice recognition, can make network usage more secure.

It is important for the individual computer in a network to be sure that a message actually comes from the source it is believed to come from. Authentication technologies can ensure this, for example, by means of digital signatures, which ascertain the source as well as the content of a data packet. Finally, to protect critical data, encryption technologies are used which provide only the sender and the selected recipient of a message with the ability to decipher it.

1.2 Ambient intelligence in the home environment

The following scenario illustrates potential applications of AmI in the home environment and in everyday activities more in general.

A young mother is on her way home, driving together with her 8-month old daughter who is sleeping in her child seat on the passenger side of the car. Arriving at home, a surveillance camera recognizes the young mother, automatically disables the alarm, unlocks the front door as she approaches it, and turns on the lights to a level of brightness that the home control system has learned she likes. After dropping off her daughter, the young mother gets ready for grocery shopping. The intelligent refrigerator has studied the family's food consumption over time and knows their preferences, as well as what has been consumed since the last time she went shopping. This information has been recorded by an internal tracking system and wireless communication with the intelligent kitchen cabinets. Based on this information, the refrigerator automatically composes a shopping list, retrieves quotations for the items on the list from five different supermarkets in the neighborhood through an Internet link, sends an order to the one with the lowest offer and directs the young mother there. When arriving at the supermarket, the shopping cart has already been filled with the items on her shopping list. Spontaneously, she decides to add three more items to her cart and walks to the check-out. Instead of putting the goods on a belt, the entire cart gets checked out simply by running it past an RFID transponder that detects all items in the cart at once and sends that information to the cash register for processing.

This scenario illustrates some anticipated aspects of the future.

What sounds like a science fiction story is an illustration of possibilities AmI can offer to people in their private lives and of the main driving forces towards AmI for home use:

- convenience
- cost savings
- time savings
- security
- safety
- entertainment

If companies succeed in offering AmI technology at a price that people are willing and able to pay for the associated benefits, AmI can turn into an entirely new market. Currently, some large companies are investing considerable sums on research and development of AmI

in anticipation of market expansion. For example, Philips has a facility in the Netherlands called HomeLab [49], which is specifically designed as a development and test site for in-home AmI technology, and where technologies like Easy Access are emerging. Easy Access recognizes a hook line from somebody humming automatically compares it with a song database and plays the song on the room's stereo equipment (Magalhães 2002 [33]). Philips' goal is to create a "perceptive home environment", which is able to identify and locate the people in it, as well as determine their intentions, to ultimately provide them with the best service possible (van Loenen 2003 [31]).

Electrolux and Ericsson built 59 smart condominiums in Stockholm, as part of the alliance's e2 Home project. These buildings were officially unveiled in February 2002, when the first buyers moved in. Inhabitants of these condominiums are now enjoying many of the benefits that modern AmI technology can provide. The following are some possibilities provided through use of a touch screen terminal located in the kitchen:

- Security: a front door security camera has been installed to permit easy identification of visitors.
- Convenience: the ability to perform availability checks and bookings of common areas such as laundry room and sauna without having to leave the apartment.
- Time saving: home delivery of items that can be placed in smart delivery boxes located outside the door.

Representatives of the e2 Home project [43] report that these new facilities have enjoyed significant interest from buyers.

The Georgia Institute of Technology has developed a 5040-square-foot "Aware home" [44] used as a laboratory of AmI development. Various products have been tested, providing valuable insight into the scope of applications that are available. Examples are the digital Family Portrait and the Gesture Pendant. Elderly persons are often a source of great concern for their families, especially if they are separated by a large distance. The Family Portrait [44] is unique device because it looks like a regularly framed picture, but its image is updated daily to reflect the general wellness of its subject. If the person in the picture is sick and inactive, the picture will capture this appearance in much the same way that seeing a person in real-life would. The Gesture Pendant [44] helps eliminate the stifling number of remote controls prevalent in most of homes today. Instead, simple hand gestures control multiple devices, providing convenience and minimizing technological clutter.

Besides the positive value that AmI offers in the home environment, there are also critical points that require careful consideration before launching AmI technology in the market: protection of privacy, costs, and minimum network scope. AmI systems must protect the privacy of the users. This is likely to be the main concern of customers when making a purchasing decision. For this reason, marketing needs to point out the security advantages AmI can offer, while R&D needs to continue improving network protection against intrusion to improve security reputation of technology.

In general, the cost of AmI technology must be at such a level that the price for an AmI system does not outweigh the perceived benefits to the user. AmI technology in the home environment needs to reach a critical mass. Going back to the example of the young mother and child, it becomes clear that one intelligent device alone has little value or impact. The key to success for AmI is the interaction between different computers in the household and their cooperation with outside networks. This requires considerable investment on the part of the users, and only a limited proportion of society will be available to afford this. To recover the high R&D investment costs, companies should first introduce AmI in the luxury segment, where higher profit margins can be attained and a higher demand for costly, sophisticated equipment is expected.

The introduction of AmI in a home environment will have an impact on personal lives in several ways. The time gained will allow people to spend more time with their family and friends. Convenience, money and time savings, security, safety and entertainment all contribute to reducing stress, leading to an overall higher quality of life. However, the ability to prepare or complete more and more everyday tasks such as shopping or banking at home potentially leads to reduced face-to-face interaction between people or, at least, to selective interaction restricted to mainly family and friends.

1.3 Social implications of ambient intelligence

The current phase of AmI/pervasive computing, in which computers are already being embedded in many devices, has begun to affect everyday life in various ways, which sometimes are not easily perceived :

- computing is spread throughout the environment
- users are mobile
- information appliances are becoming increasingly available
- communication is made easier-between individuals, between individuals and things, and between things.

Bio-sensing allows devices to not only be able to sense the presence of a user, but also the user's needs and goals to enhance person-to-person communication. Organizations are exploring ways to incorporate gaze tracking, emotion detection, speech recognition, and gesture recognition in "smart" computers, that in the future will often be invisible and will offer an intuitive user interface (Ark and Selker 1999 [24]).

AmI in an intelligent home environment is sensitive, personalized, adaptive, anticipatory and responsive to people. As envisioned by developers, Philips Electronics and Massachusetts Institute of Technology (MIT), ambient intelligence is a marriage between the technologies of ubiquitous computing and social user interfaces.

At the front-end of an AmI system are a variety of tiny devices that can hear, see, or feel an end-user's presence. At the back-end, wireless-based networked systems make sense of these data, identifying the end-user and understanding his/her needs (Ribauda 2003 [35]). Wireless digital assistants, network-attached refrigerators, and smart houses are examples

of pervasive or ambient computing. For example, a "virtual house" can update its contents whenever the “inhabitants” buy merchandise on the Net or in person, and can be walked around with the aid of a virtual reality headset. IBM's Pervasive Computing Lab [45], MIT's Media Laboratory [47], Accenture's Technology Labs [41] and Microsoft's Hardware Devices Group [48] are some of the organizations working with AmI/pervasive computing, based on trends in software, hardware, and networking.

Philips invests approximately $3.25 billion in research and development at its design headquarters to help realize its vision of ambient intelligence in which billions of tiny processors embedded in furniture, walls, and clothes will communicate wirelessly with one another and manage various domestic tasks. For example, it is developing Pogo [49], a jumble of tools to encourage 4-to-8-year-olds in the art of storytelling. A product called Nebula [49] projects scenery, an alarm clock, and games on to a bedroom ceiling. Using HomeLab and Easy Access, Cappellini-designed beds, couches, and chairs have loud-speakers, projectors, and Web screens integrated into headboards and armrests. The bathroom SmartMirror turns out to be a TV. Ambient-intelligent homes have been likened to an old English butler: they are able to make decisions on the user's behalf (Wylie 2003a [38], Wylie 2003b [39]).

According to Philips, the best way to get started is to study "the physical, social and cultural context in which technology will be used and its implications on daily life". The context here is the domicile. By design, HomeLab is set up to allow researchers and designers to work in concert with end-users "to realize a shared and tangible vision" of future in-home electronic systems. MIT's Rodney Brooks said of the HomeLab project: "Only through living with our technologies can we discover which ones really improve our lives and which ones only sound good as proposals". Since observation is a key element at this stage, HomeLab's audio- and video-based control system collects and analyzes the data provided by volunteers: the full range of human activity, including postures, facial expressions and social interactions. "By exploring people's cultural expectations with respect to functions, forms and behavior," states Stefano Marzano, CEO Philips Design, "we can ensure that Ambient Intelligence really does improve people's quality of life experience." The target date for true ambient living according to Philips is 2020 (Anon. 2002 [23], van Loenen 2003 [31]).

A brand of high-end kitchen appliances, sold in Italy, can regulate their combined consumption, and so keep the consumer from incurring fees for peak power use. In addition to power and networking, designers of these future systems will need to make them aware of their user's identity and location. In other words, not only will devices be pervasive, these devices will have to adjust to human users switching modes during the course of the day, i.e., "use a PDA, move to the car, then back to the office", without having to restart their work or connect to different back-ends. Intermediary transcoding technologies enable devices to communicate with the network, and one information network to communicate with another (Booker 1999 [26]).

The next step in the evolution of computing involves the move toward ubiquitous computing, in which computers will be embedded in natural movements and interactions with environments, both physical and social. Ubiquitous computing will help organize and

mediate social interactions wherever and whenever these situations might occur. Lyytinen and Yoo (2002) [32] predict that during the next five to ten years, ubiquitous computing will come of age and the challenge of developing ubiquitous services will shift from demonstrating the basic concept to integrating it into the existing computing infrastructure and building widely innovative mass-scale applications that will continue the computing evolution. The main challenges in ubiquitous computing originate from integrating large-scale mobility with pervasive computing functionality. In addition, Ribauda (2003) [35] states that the key technical hurdles are making computers see, hear, speak and understand language, since ambient intelligence relies on technologies that are far from fully developed, with no agreed-upon standard yet for operating systems.

The vision of a future filled with smart and interacting everyday objects offers a whole range of fascinating possibilities. For example, parents will no longer lose track of their children, even in the busiest of crowds, when location sensors and communications modules are sewn into their clothes. Similar devices attached to timetables and signposts could guide blind people in unknown environments by "talking" to them via a wireless headset. Tiny communicating computers could also play a valuable role in protecting the environment; for example, sensors the size of dust particles that detect the dispersion of oil spills or forest fires. Another interesting possibility is that of linking any sort of information to everyday objects, allowing future washing machines to query our dirty clothes for washing instructions. (Bohn *et al.* 2004 [25]) While personal gadgetry in the form of smartphones and Internet fridges continue to keep the press occupied, industry has quietly begun setting its sights on the enormous business potential that technologies such as wireless sensors, RFID tags, and positioning systems have to offer. Analysts have termed this the "real-time economy" or the "now-economy" (Siegele 2002 [36]).

PriceWaterhouseCooper's (PwC) forecast of major trends in IT infrastructure over the next three years focuses on grid computing, computing as a utility, Internet protocol (IP) dial tone and pervasive computing. PwC's forecasts on pervasive computing envision IT as a universal resource that will eventually become an invisible part of daily life. Its realization, however, will require advances in user interfaces, system software, security and pervasive networks (Domingo 2002 [29]).

1.4 Current Challenges

The backend of an AmI Environment consists of devices such as sensors, TVs, PCs, cameras, projectors, etc., which have the ability to communicate with each other. This communication, as previously described, is satisfied by WiFi, LAN, Bluetooth, etc. Both when and how these devices communicate with each other, as well as the signals that are transferred to the applications, depend on a layer called Middleware (see Chapter 5). The decision that is taken from the Middleware concerning the signals is generated from another layer which is called decision engine. These two layers are not addressed in this thesis.

In order for the user to manipulate all the devices in a Smart Environment, an application needs to be used for each device individually. These applications communicate with the devices through the Middleware. However, the overhead to build application for every device individually is high. Additionally, it is of critical importance to ensure that the user

interfaces of such applications are accessible to all home inhabitants, including the elderly and people with disabilities. Furthermore, it would be clearly more convenient for the users of smart homes to be able to manage all devices through applications which exhibit consistent and compatible user interfaces, or even through a unique user interface, in order to maximize usability.

To contribute addressing these issues, this thesis reports the development of two interrelated tools: the AmI Designer and the AmI Player. AmI Designer is a tool that allows designing an accessible graphical user interface easily and producing a description of that interface. The AmI Player uses that description to generate accessible graphical user interfaces through which the user will be able to operate the various devices that can be found within the Smart Environment. In combination, these two tools provide a novel environment for the rapid prototyping and automatic generation of user interfaces of smart home applications.

The first important advantage of the proposed approach is that designers are allowed to create to create their applications in few steps, substantially reducing the programming efforts that would otherwise be necessary in terms of both time and lines of code.

Another very important achievement is that the generated GUIs are fully accessible and usable for a diverse population of end users, including people with disabilities. End users can use the generated applications with many different interaction devices. This helps motor and visual impaired users to interact with AmI environments. Accessibility is achieved through a multimodal approach integrating non visual feedback and hierarchical scanning.

Finally, the AmI Designer and the AmI Player were designed in such a way to ensure easy extensibility in the future. Potential extensions include interaction styles, services and technological platforms.

2. Related Work

User Interface Development is a complex task which is widely recognized as requiring appropriate tool support. Efforts in this direction have so far been focused on:

- the provision of graphical environments for user interface prototyping supporting a WYSIWYG ("What You See Is What You Get") design paradigm
- the provision of markup languages for the description of graphical user interfaces
- the provision of mechanisms for the automatic generation of user interfaces, intended to fill the so called design-implementation gap, thus considerably reducing development time.

This Chapter describes and discusses related work in tool support for the development of Graphical user interfaces, outlining the main characteristics of existing tools, as well as advantages and disadvantages from a developer's point of view. A schematic classification of existing tools is provided in Figure 1. In this Figure, tools are distinguished into:

- Graphical User Interface Editors
    - Editors producing UI descriptions in a markup language
    - Editors in Integrated Development Environments (IDEs)
- User Interfaces Markup Languages
- Graphical User Interface Generators which take as input the description of an application in a markup language and generate the final application
- Integrated tools including editors, markup languages and generators

The tools reported in this thesis instantiate the following user interface generation process(1) the designer creates a GUI design through an editor, (2) a description of the designed interface in a markup language is produced, and (3) such a description is used as input to a GUI Generator which provides as output a running interface.

Therefore, the AmI Designer reported in this thesis can be considered as a GUI editor and the AmI Player as a GUI Generator. So far, however, no such existing tool addressed the development of user interfaces specifically for AmI applications. In the remaining of this Chapter, existing editors, GUI Generators and markup languages are analyzed. A discussion of how AmI Designer and AmI Player differ from existing tools and of the approach followed in filling current gaps is also provided.

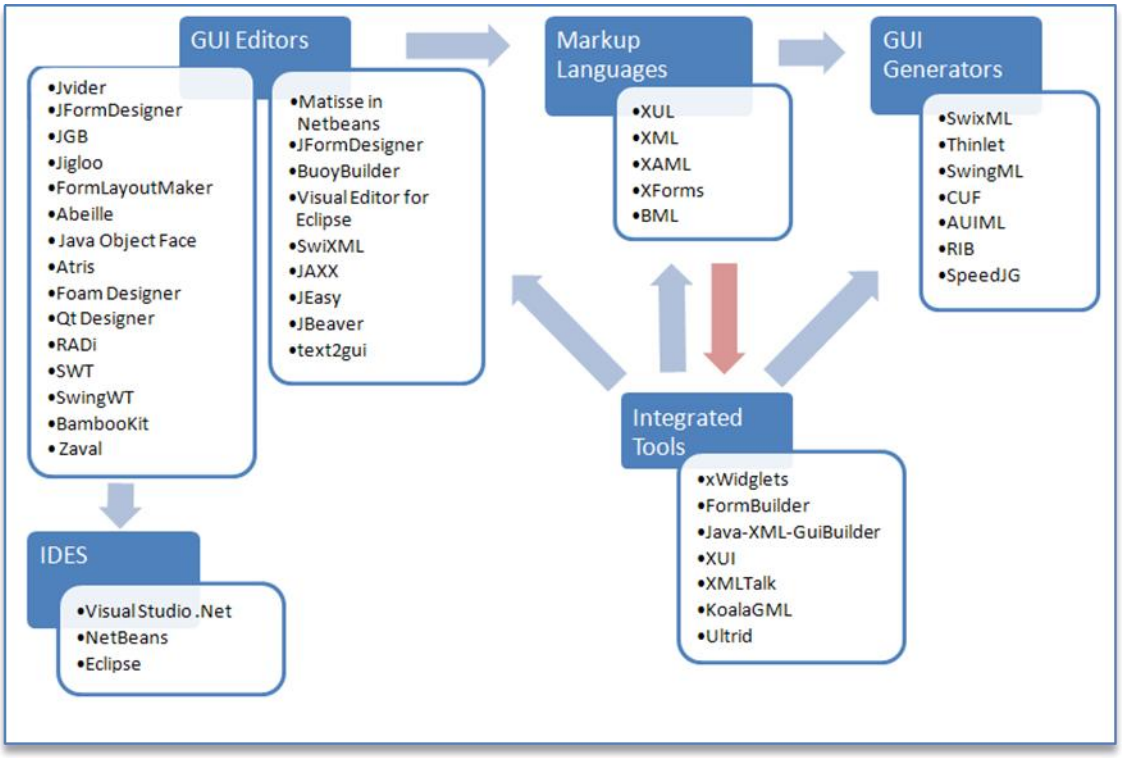


Figure 1. Existing tools for Graphical User Interface design and generation

2.1 GUI Editors

GUI Editors are used in order to create graphical user interfaces quickly and easily. AmI Designer is such a kind of editor.

Available GUI Editors offer graphical editing facilities for prototyping user interfaces. They mainly differ with respect to:

- the output of the GUI design process: this may be a GUI description in some markup language or GUI code which needs to be integrated in the overall application by programming
- the facilities made available to designers.

This subsection provides an overview of the main available GUI Editors.

2.1.1. GUI Designers

Jvider[1]

Jvider (see Figure 2) is a GUI builder tool for Java™ Swing applications, supporting the design of graphical user interfaces for Java™ applets and applications. Jvider uses standard Java™

[1] www.jvider.com

Swing components, allows testing the designed UI on the fly without compiling, supports viewing and exporting the produced Java™ source code as Frame or Applet, offers JAVABEANS™ support, and finally has the ability to load UI design from xml configuration at user application run time with Jvider Runtime. Jvider is claimed to have an intuitive user interface, and includes tutorials and samples.

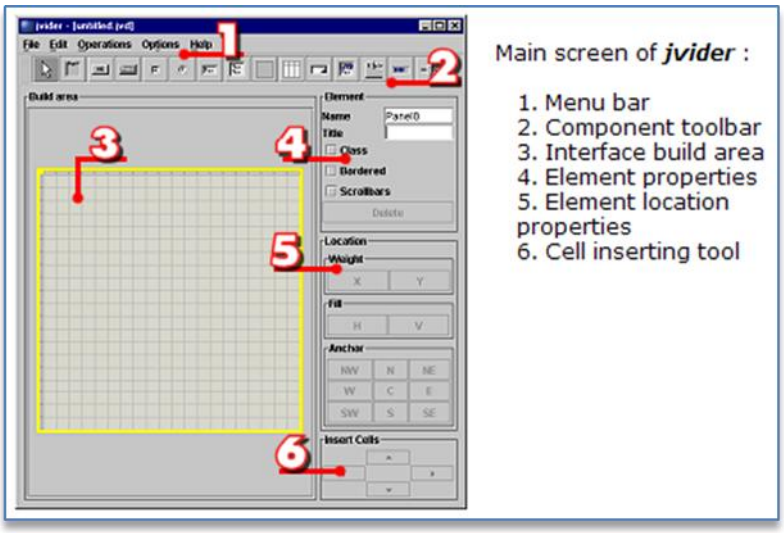


Figure 2. Jvider Designer

JFormDesigner[2]

JFormDesigner (see Figure 3) facilitates Swing GUI design, and is claimed to substantially reduce the time that is spent on hand coding forms.

It is claimed to be easy and intuitive to use, powerful and productive, and to have an easy-to-use but powerful user interface, suitable also for non-programmers for prototyping purposes. It can be used both as an IDE plug-in for Eclipse, IntelliJ IDEA and JBuilder and as stand-alone application.

It includes visual guidelines and tools for optimal spacing, alignment and resizing of components, as well as table layout support. Localisation support is also supported. JFormDesigner can either generate Java source code (the default) or XML files which can be loaded into an open-source (BSD license) runtime library at runtime.

[2] www.jformdesigner.com

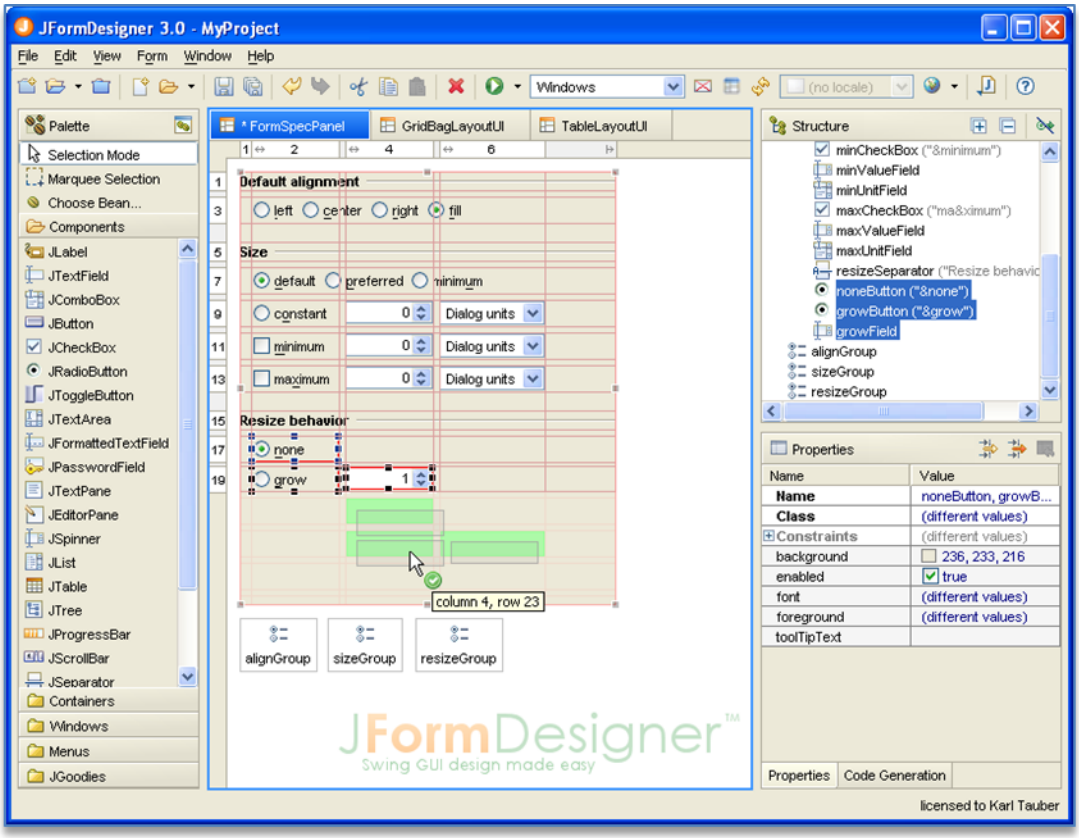


Figure 3. JFormDesigner Designer

JGB (Java GUI Builder) [3]

The Java Gui Builder program is designed to decouple the GUI building code from the rest of the application code, without hand-writing code. It allows describing the layout of windows and controls using an XML file, and supports on-the-fly validation. Using an XML file to describe GUI components allows users the flexibility to rewrite their windows so that they suit their needs, without opening up the innards of the program to the users.

Eclipse/VEP (Eclipse Visual Editor Project plugin) [4]

The Eclipse Visual Editor project is a vendor-neutral, open development platform supplying frameworks for creating GUI builders, and exemplary, extensible tool implementations for Swing/JFC and SWT/RCP (see Figure 4). These tools are exemplary in that they verify the utility of the Eclipse Visual Editor frameworks, illustrate the appropriate use of those frameworks, and support the development and maintenance of the Eclipse Visual Editor Platform itself.

The purpose of the Eclipse Visual Editor Project is to advance the creation, evolution, promotion of the Eclipse Visual Editor platform, and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services. In

[3] http://jgb.sourceforge.net/
[4] www.eclipse.org/vep/WebContent/main.php

particular, the Visual Editor Project intends to be useful for creating GUI builders for other languages such as C/C++ and alternate widget sets, including those that are not supported under Java.

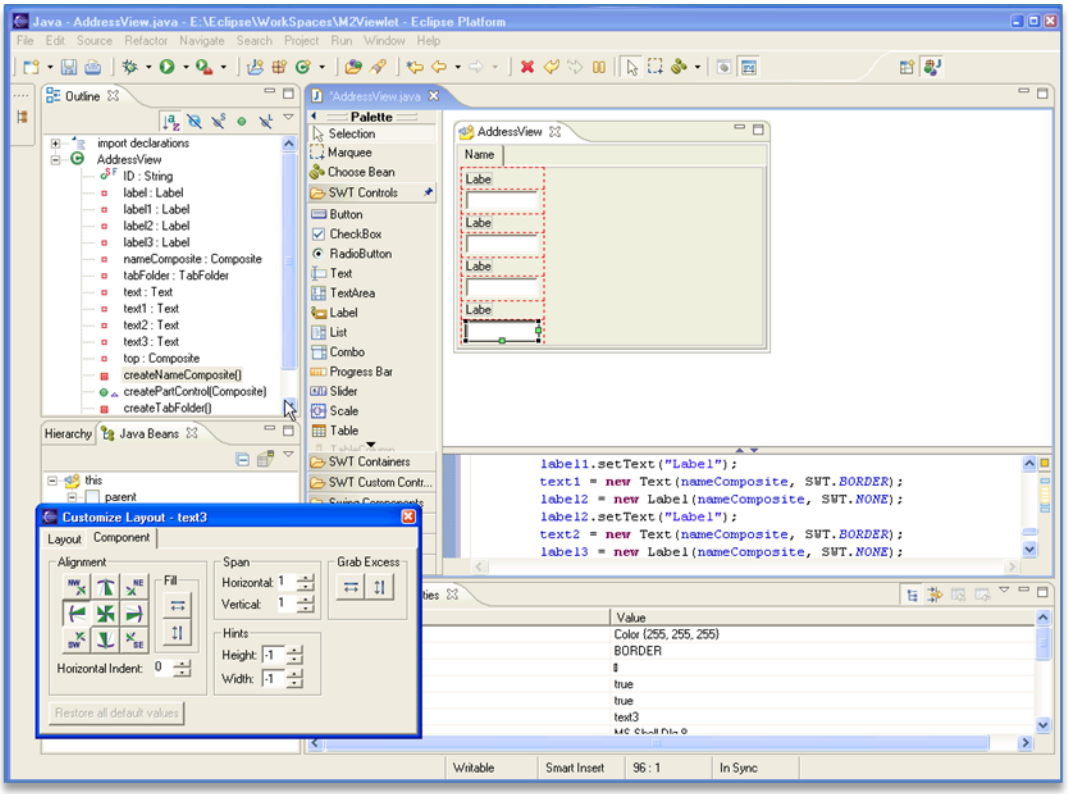


Figure 4. Eclipse Visual Editor Designer

Jigloo[5]

Jigloo is a SWT/Swing GUI Builder for Eclipse and WebSphere (see Figure 5), free for non-commercial use. CloudGarden's Jigloo GUI Builder is a plugin for the Eclipse Java IDE and WebSphere Studio[6], which allows building and managing both Swing and SWT GUI classes. Jigloo creates and manages code for all the parts of Swing or SWT GUIs, as well as code to handle events, and shows the GUIs as they are being built. Jigloo parses java class files to construct the form that is used when designing a GUI, and it can work on classes that were generated by other GUI builders or IDEs, or hand-coded classes. It can also convert from a Swing GUI to a SWT GUI and vice-versa.

Jigloo is claimed to be straightforward, fast, powerful, easy to use and fully integrated with Eclipse, and to lead to substantial time-savings for GUI development and maintainance tasks.

Jigloo itself is is highly-customizable, and the code it generates can also be customized, whereas existing code can be re-arranged to follow the preferred style. Custom classes can be added to forms, and JavaBeans with Customizers and custom properties are supported.

[5] www.cloudgarden.com/jigloo
[6] http://www.ibm.com/websphere/studio

In addition, Jigloo supports visual inheritance - it can design classes which extend other custom classes, which may be public, abstract or non-public. Navigation between code and form editors is very easy - with Jigloo highlighting the relevant section of code when the form editor has focus, or the relevant form element when the code editor has focus.

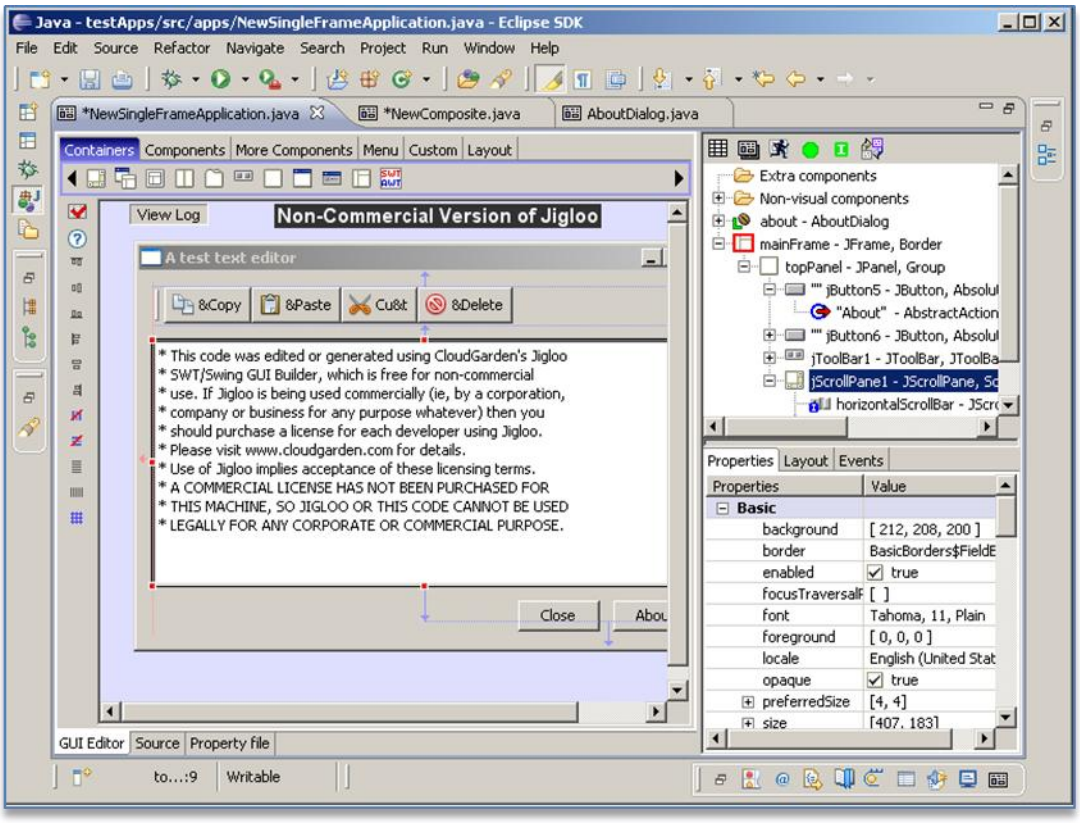


Figure 5. Jigloo Designer

FormLayoutMaker[7]

FormLayoutMaker is a WYSIWYG tool (see Figure 6) that enables easy creation of Java Swing layouts using the JGoodies FormLayout layout manager. FormLayoutMaker works in conjunction with any development environment, from NetBeans and Eclipse to VI and Emacs. FormLayoutMaker focuses on layout screens. Therefore, the designer cannot change colors, fonts, and all of the properties on a control.

[7] http://sourceforge.net/projects/formlayoutmaker/

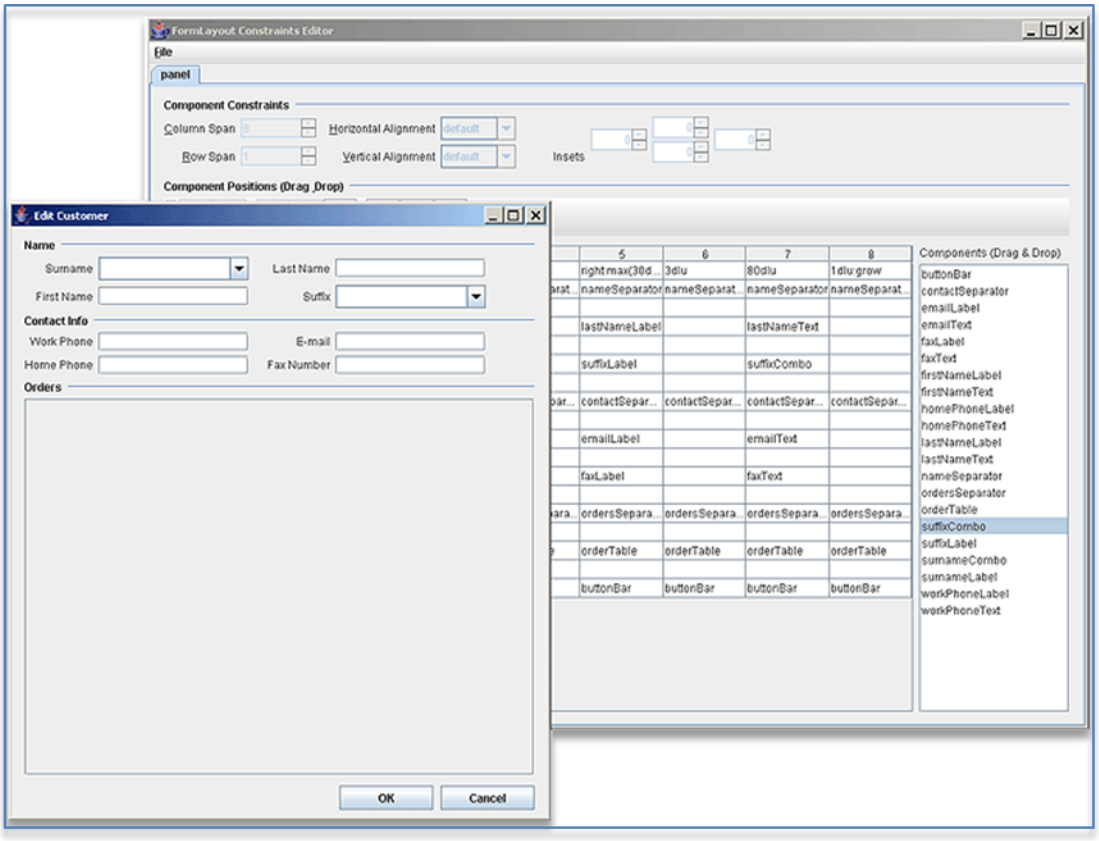


Figure 6. FormLayoutMaker Designer

Abeille[8]

Abeille Forms Designer is a GUI builder for Java applications. It includes a WYSIWYG editor (see Figure 7). Components can be easily customized by adding images or modifying their properties. Advanced fill effects are supported such as textures and gradients.

Abeille has intuitive layout rules and is based on the JGoodies FormLayout system (https://forms.dev.java.net). The FormLayout is a popular, open source layout manager for Java and is used by thousands of developers worldwide. Abeille comes with all the required software. Abeille stores forms in binary files which can be loaded by an application and added to any Swing container. While the designer is licensed under the LGPL, the forms runtime has a BSD license. This allows forms created by the designer to be used freely in commercial applications.

[8] https://abeille.dev.java.net/

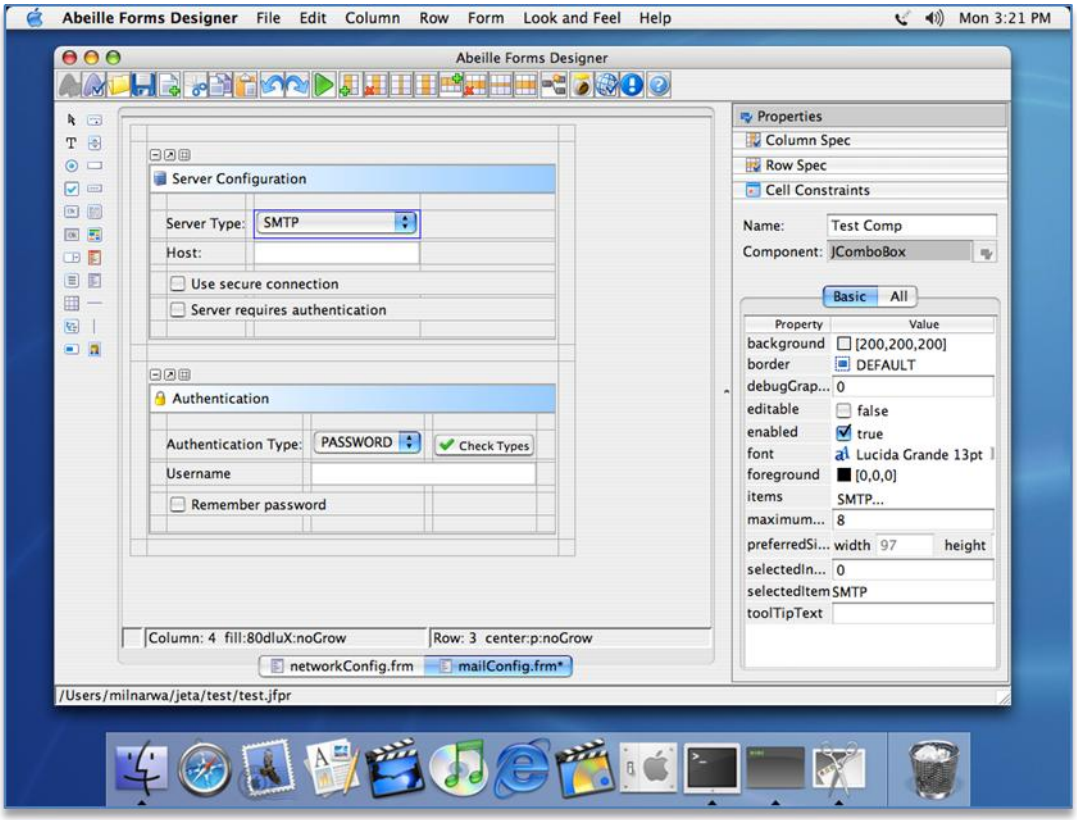


Figure 7. Abeille Designer

Java Object Face[9]

JOF provide a way for displaying and populating java objects. It automates the creation of GUI forms and it is able to produce AWT/Swing/JSP interfaces, just by passing java bean or XML to it. JOF is an Object-to-GUI solution.

Atris[10]

The Atris Framework (see Figure 8) makes use of Java and the Java/Swing API to simplify the development of GUI based applications. These applications can be run on any computer platform and operating system that supports the Java language (which includes all of the principal operating systems in use today). This means, for example, that a developer can create their program in a Microsoft Windows operating system environment, and have it run on any GUI based interface, including Linux, Solaris, Mac-OS, etc., without changing the code. This gives the developer access to many more platforms for their software and a write once, run anywhere ability.

Although Java/Swing is a rich GUI development language, it does not provide many of the usability 'features' that most GUI users have come to expect. The Framework itself is focused on a subset of applications that can be written with Java. This subset includes most GUI applications that require the persistent storage of data either locally or remotely.

[9] http://sourceforge.net/projects/jof/
[10] http://sourceforge.net/projects/atrisframework

The Atris GUI Framework is designed to handle many of the routine tasks and functions that a standard GUI interface provides to users. For example, tiling of windows, remembering the position and size of windows, automatic logging into databases, handling of application look and feel, the management of application properties, automatic search and sorting of data and a host of other house keeping tasks. By providing these features, the framework drastically simplifies the work that developers must do, while ensuring that every application developed with the framework maintains the identical look and feel, thus simplifying end user training.

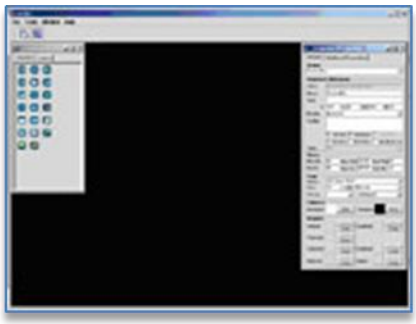


Figure 8. Atris Designer

Qt Designer[11]

Qt Designer (see Figure 9) is a powerful cross-platform GUI layout and forms builder. It allows rapid design and creation of widgets and dialogs using on-screen forms using the same widgets that will be used in the application. Forms created with Qt Designer are fully-functional, and they can be previewed so as to ensure that they will look and feel exactly as intended.

Qt Designer makes it easy to visually design advanced user interfaces - at any time, the code required to reproduce and preview the interface can be generated, changing and adjusting the design as often as needed. Qt Designer helps building cross-platform user interfaces with layout tools that move and scale controls (widgets) automatically at runtime, taking font sizes and language use into consideration. The resulting interfaces are both functional and native-looking, comfortably suiting your users' operating environments and preferences.

Qt Designer has been designed to work as a powerful stand-alone tool, but can also be utilized from within the an IDE It integrates seamlessly with Microsoft Visual Studio and Eclipse - and in all cases, the user experience is based on the IDE, enabling developers to leverage and extend their existing skills.

Qt Designer incorporates Qt's powerful auto-scaling, font-aware layout system. Qt layouts enable developers to create dialogs that maintain an attractive, professional look and feel regardless of the display, font size, language, style or underlying platform chosen by the end user.

[11] http://www.trolltech.com/products/qt/designer.html

Qt Designer supports previewing in different styles, ensuring that the layout and labels appear as intended when resizing.

Qt is extensible and customizable, allowing using custom controls as real widgets. Custom controls can be written from scratch, or can be defined based on one or more standard Qt widgets.

Qt Designer provides all the expected GUI tool functionality such as the ability to define dialogs and layouts, set tab order and edit control properties. In addition, Qt's powerful Signals and Slots inter-object connections can be edited from Qt Designer, as well.

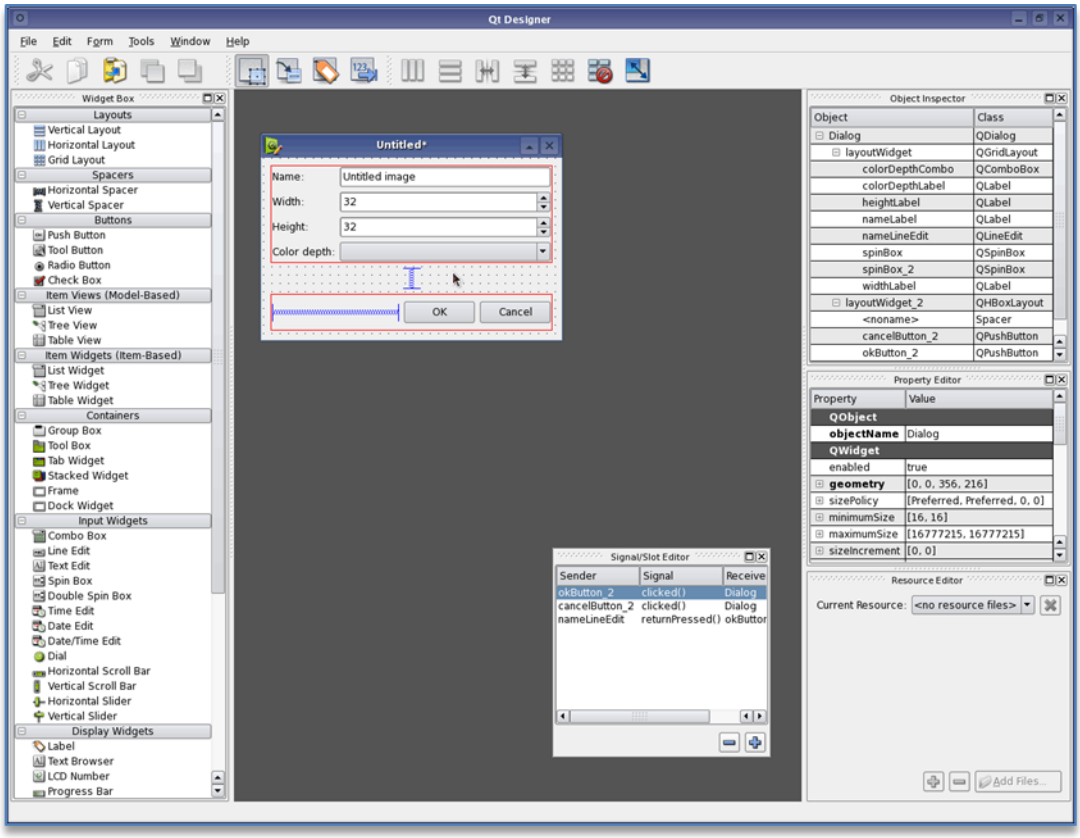


Figure 9. Qt Designer Designeri

RADi

RADi is an easy-to-use, though powerful, Java GUI builder and JavaBeansTM container. It is claimed to be intuitive to use, supports drag and drop and the designer does not have to care about layout managers. Though layouts of any complexity can be created, RADi provides tools to master this complexity. RADi comes with a royalty-free runtime library which processes the layout definition files, creates the GUI and executes listeners, event and property handlers.

SWT[12]

[12] http://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html

SWT is the software component that delivers native widget functionality for the Eclipse platform in an operating system independent manner. It is analogous to AWT/Swing in Java, but it uses a rich set of native widgets. Cross platform widget libraries are very difficult to write and maintain. This is due to the inherent complexity of widget systems and the many subtle differences between platforms.

This free GUI library from IBM provides an interface to the native system GUI (currently Windows and GNOME). The advantages are a more responsive user experience. The disadvantages are a different API, so programs can not simply be changed from Swing to SWT, and are not as portable, at least until versions are available for more systems. It's possible to compile completely native (no virtual machine) programs for Linux.

SwingWT[13]

SwingWT (see Figure 10) is a 100% pure Java library which aims to be a free implementation of Swing and AWT. Unlike Swing, it drives native peer widgets from SWT. This free GUI implementation provides a Swing compatible API, but uses SWT for the implementation. It can produce User Interface for mobile device applications.

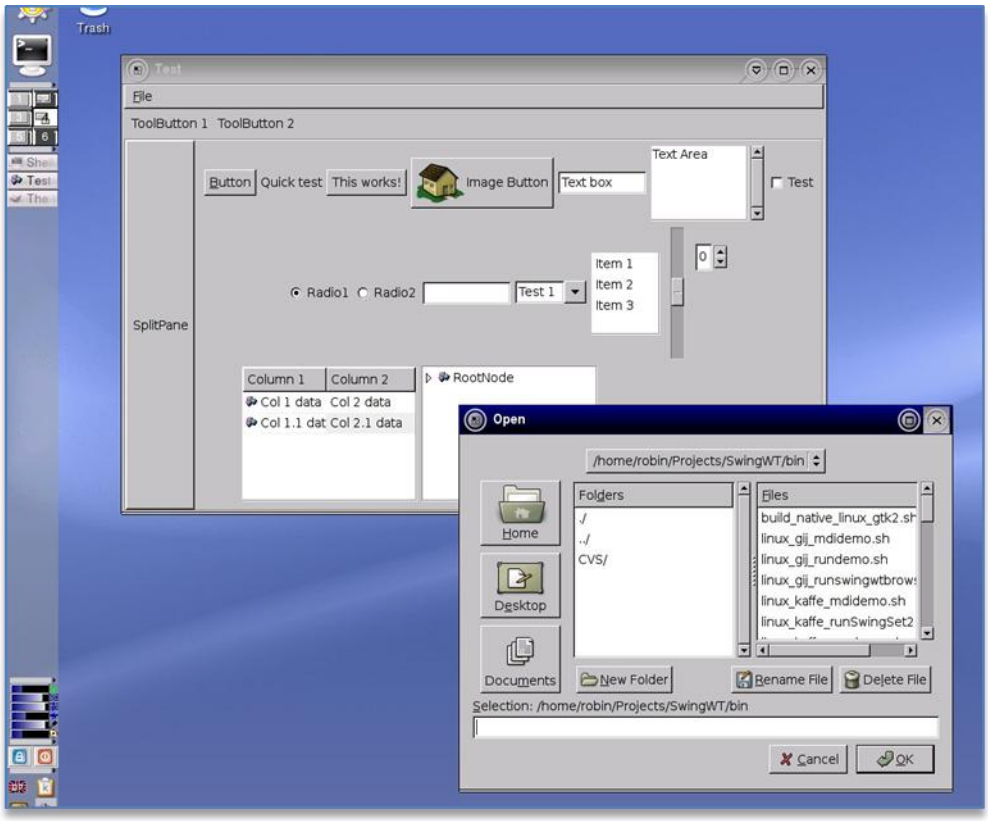


Figure 10. SwingWT Designer

BambooKit[14]

BambooKit (see Figure 11) is claimed to lower development costs dramatically. An XML-based environment enables web developers to quickly build Bambookit applications.

[13] http://swingwt.sourceforge.net/
[14] http://www.bambookit.com/

Bambookit overcomes the limitations of the "click-wait-refresh" model associated with HTML that requires a new page to travel from the server to the browser upon every user click. This refresh process can take precious seconds, and often minutes and significantly frustrates end user productivity. The Interactive User Experience provided by Bambookit-generated interfaces adheres to the principle of feedback through visual cues on any control that can manipulate the user interface environment. All controls that can be acted upon would highlight on a mouse rollover, mouse click and mouse release. It also reduces bandwidth by 90 percent as it eliminates unnecessary communications between client and server of the "click-wait-refresh" model associated with HTML. The "click-wait-refresh" model also weighs heavy on the network infrastructure, with each page occupying a footprint of kilobytes - most of which is redundant. From PCs, to handheld devices, to any Internet terminal Bambookit based thin client applications will run equally well on different environments: operating systems, browsers, webpads, mobile phones. There is no need to rewrite the application for every device. For end users, applications look, behave and perform the same, regardless of their access means and connection.

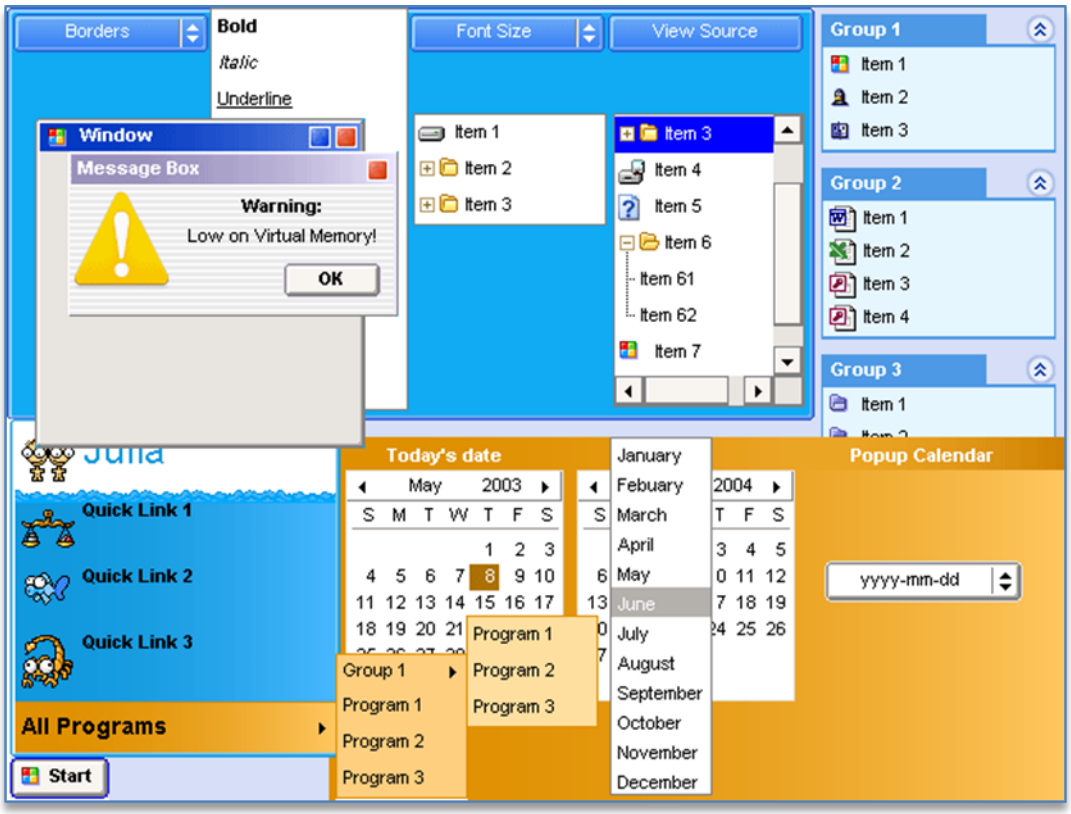


Figure 11. Bambookit Designer

2.1.2. GUI Editors which generate Markup Descriptions

Matisse in Netbeans[15] *- WYSIWYG GUI editor*

[15] http://www.netbeans.org/kb/articles/matisse.html

The Matisse editor is intended to reduce the learning curve and development time needed to produce professional quality Java GUIs. The editor exposes simple layout rules that are easy to understand and to use quickly, letting developers lay out components freely, and provides visual guidelines for optimal spacing between components and alignment of components. Matisse also infers the appropriate resizing behavior, freeing the developer from the complexities of Swing layout managers. The intuitive visual form builder can be used to produce a professional GUI easily, while, in the background, the IDE produces the correct implementation using a layout manager and other Swing constructs.

JFormDesigner[16] *- WYSIWYG GUI editor*

JFormDesigner facilitates Swing GUI design. It decreases the time spent on hand coding forms, and is claimed to quickly pays back its cost in improved GUI quality and increased developer productivity.

BuoyBuilder[17] *- WYSIWYG GUI editor based on Buoy toolkit*

BuoyBuilder™ is an editor building upon the Buoy toolkit, which simplifies Java™ Swing programming without sacrificing its power. BuoyBuilder encourages a clean model/view/controller design paradigm, eliminates virtually all GUI infrastructure code, and provides new layout mechanisms offering greater control over the developed UI.

BuoyBuilder supports working with live objects in a graphical environment.

Visual Editor for Eclipse[18]

The Eclipse Visual Editor is a vendor-neutral, open development platform supplying frameworks for creating GUI builders, and exemplary, extensible tool implementations for Swing/JFC and SWT/RCP. These tools are exemplary in that they verify the utility of the Eclipse Visual Editor frameworks, illustrate the appropriate use of those frameworks, and support the development and maintenance of the Eclipse Visual Editor Platform itself.

The purpose of the Eclipse Visual Editor is to advance the creation, evolution, promotion of the Eclipse Visual Editor platform, and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services. In particular, the Visual Editor Project intends to be useful for creating GUI builders for other languages such as C/C++ and alternate widget sets, including those that are not supported under Java.

SwiXML[19] *- XML representation of GUI*

SwiXml is a small Java library created to produce Swing GUIs from a small XML language. SwiXml does not introduce any new layout managers or component classes. Instead, it directly operates on the Swing classes using reflection. This means that the XML syntax is easy to learn for anyone used to the Swing API. It also has the side benefit of keeping the

[16] http://www.jformdesigner.com/
[17] http://www.buoybuilder.com/
[18] http://www.eclipse.org/vep/WebContent/main.php
[19] http://weblogs.java.net/pub/a/today/2006/02/21/building-guis-with-swixml.html

library very small (under 60k, plus the JDOM .jar), which makes application deployment a more pleasant experience.

JAXX[20] - XML based GUI representation

JAXX is an open-source XML User Interface framework for Java. JAXX enables to write simple XML files describing components and their interactions, and then compile those XML files into ordinary Java classes. User interface components can be developed much faster and more easily in JAXX than in ordinary Java code.

2.1.3. IDEs

Integrated Development Environments (IDEs) allow developers to create applications by drawing the graphical user interface and check the application's performance through the environment itself. In this sub chapter the three most popular IDEs, Microsoft Visual Studio, NetBeans and Eclipse are presented, focusing mainly on their design.

Microsoft Visual Studio .Net 2008

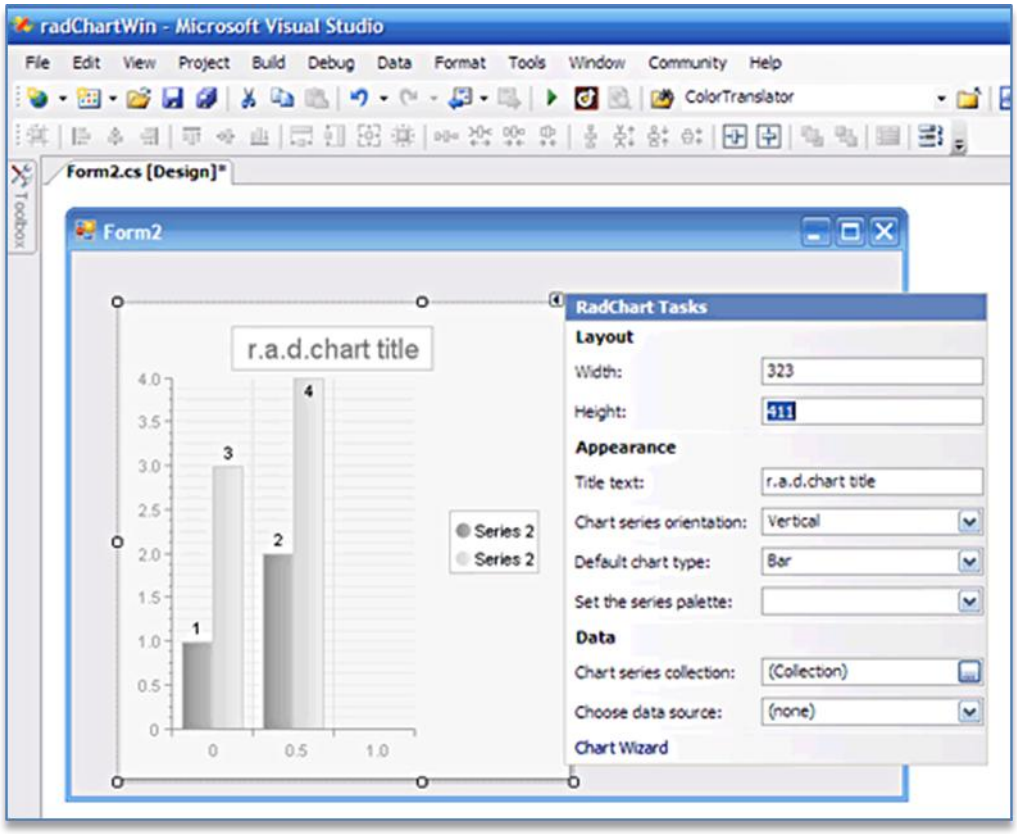


Figure 12. Microsoft Visual Studio IDE

Microsoft Visual Studio 2008 Professional Edition (see Figure 12) is a comprehensive set of tools that accelerates the process of turning the developer's vision into reality. It was engineered to support development projects that target the Web (including ASP.NET AJAX), Windows Vista, Windows Server 2008, the 2007 Microsoft Office system, SQL Server 2008, and Windows Mobile devices. The number of platforms that developers must target to meet

[20] http://www.jaxxframework.org/wiki/Main_Page

business needs is increasing rapidly. Visual Studio 2008 Professional Edition provides an integrated toolset for addressing all of these needs.

Today's developers face the challenge of targeting a broad range of platforms and crafting applications that quickly deliver value to the business. Integrated designers and language features in Visual Studio allow developers to build the connected applications demanded by today's businesses while taking advantage of the .NET Framework 3.5 to reduce development time.

The Microsoft Visual Studio development system is a suite of development tools designed to aid software developers—whether they are novices or seasoned professionals—face complex challenges and create innovative solutions. Every day, software developers break through tough problems to create software that makes a difference in the lives of others. Visual Studio's role is to improve the process of development to make the work of achieving those breakthroughs easier and more satisfying.

Visual Studio improves the process as follows:

- Productivity

Visual Studio tools include efficient code editors, IntelliSense, Wizards, and multiple coding languages in one integrated development environment (IDE) to high-end application life-cycle management (ALM) products in Microsoft® Visual Studio® Team System. New versions of Visual Studio keep bringing innovative tools to help developers focus on solving problems, not waste time on minutiae.

- Integration

With Visual Studio, software developers benefit from an integrated product experience that spans tools, servers, and services. Visual Studio products work well together—not just with one another, but also with other Microsoft software, such as Microsoft server products and the Microsoft Office system.

- Comprehensiveness

Visual Studio offers a choice of tools for all phases of software development—development, testing, deployment, integration, and management—and for every kind of developer—from the novice to the skilled professional. Visual Studio is also engineered to support development across all types of devices—PCs, servers, the Web, and mobile devices.

- Reliability

Visual Studio is engineered and tested to be consistently dependable, secure, interoperable, and compatible. Visual Studio offers an unmatched combination of security features, scalability, and interoperability. Although Visual Studio always incorporates forward-thinking features, it is designed to ensure backward-compatibility wherever possible.

- Visual Studio and the Microsoft Application Platform

The Microsoft Application Platform is a portfolio of technology capabilities, core products, and best practice guidance focused on helping IT and development departments partner with the business to maximize opportunity. As one of the core products of the Microsoft Application Platform, Visual Studio can help developers to deliver value-added services by providing a single, fully integrated development environment for all types of development, including Microsoft Windows, Microsoft Office, Web, and mobile applications. Visual Studio development solutions offer powerful ways to:

- Increase productivity and quality through integrated and familiar tools.
- Deploy, secure, and support your critical Web applications and infrastructure.
- Reduce costs through better visibility of your development process.
- Provide better predictability and planning through integrated process and methodology support.

- Advanced tools

Visual Studio 2008 provides advanced development tools, debugging features, database functionality, and innovative features for quickly creating tomorrow's cutting-edge applications across a variety of platforms.

Visual Studio 2008 includes enhancements such as visual designers for faster development with the .NET Framework 3.5, substantial improvements to Web development tools and language enhancements that speed development with all types of data. Visual Studio 2008 provides developers with all the tools and framework support required to create compelling, expressive, AJAX-enabled Web applications.

Developers will be able to take advantage of these rich client-side and server-side, frameworks to easily build client-centric Web applications that integrate with any back-end data provider, run within any modern browser, and have complete access to ASP.NET application services and the Microsoft platform.

- Rapid Application Development

To help developers rapidly create modern software, Visual Studio 2008 delivers improved language and data features, such as Language Integrated Query (LINQ), that make it easier for individual programmers to build solutions that analyze and act on information.

Visual Studio 2008 also provides developers with the ability to target multiple versions of the .NET Framework from within the same development environment. Developers will be able to build applications that target the .NET Framework 2.0, 3.0 or 3.5, meaning that they can support a wide variety of projects in the same environment.

- Break Through User Experience

Visual Studio 2008 offers developers new tools that speed creation of connected applications on the latest platforms including the Web, Windows Vista, Office 2007, SQL Server 2008, and Windows Server 2008. For the Web, ASP.NET AJAX and other new

technologies will enable developers to quickly create a new generation of more efficient, interactive, and personalized Web experiences.

- Effective Team Collaboration

Visual Studio 2008 delivers expanded and improved offerings that help improve collaboration in development teams, including tools that help integrate database professionals and graphic designers into the development process.

NetBeans IDE

NetBeans is a free, open-source Integrated Development Environment for software developers (see Figure 13). It provides all the tools needed to create professional desktop, enterprise, web, and mobile applications with the Java language, C/C++, and Ruby. NetBeans IDE is claimed to be easy to install and use straight out of the box and runs on many platforms including Windows, Linux, Mac OS X and Solaris.

The NetBeans IDE 6.1 release provides several new features and enhancements, such as rich JavaScript editing features, support for using the Spring web framework, and tighter MySQL integration. This release also provides improved performance, especially faster startup (up to 40%), lower memory consumption and improved responsiveness while working with large projects.

NetBeans creates professional-looking Java GUIs by placing and aligning components on a canvas. Web applications can be built visually using Ajax, CSS, and JavaScript. GUI applications that run on mobile phones, set-top boxes, and PDAs can be created, tested and debugged.

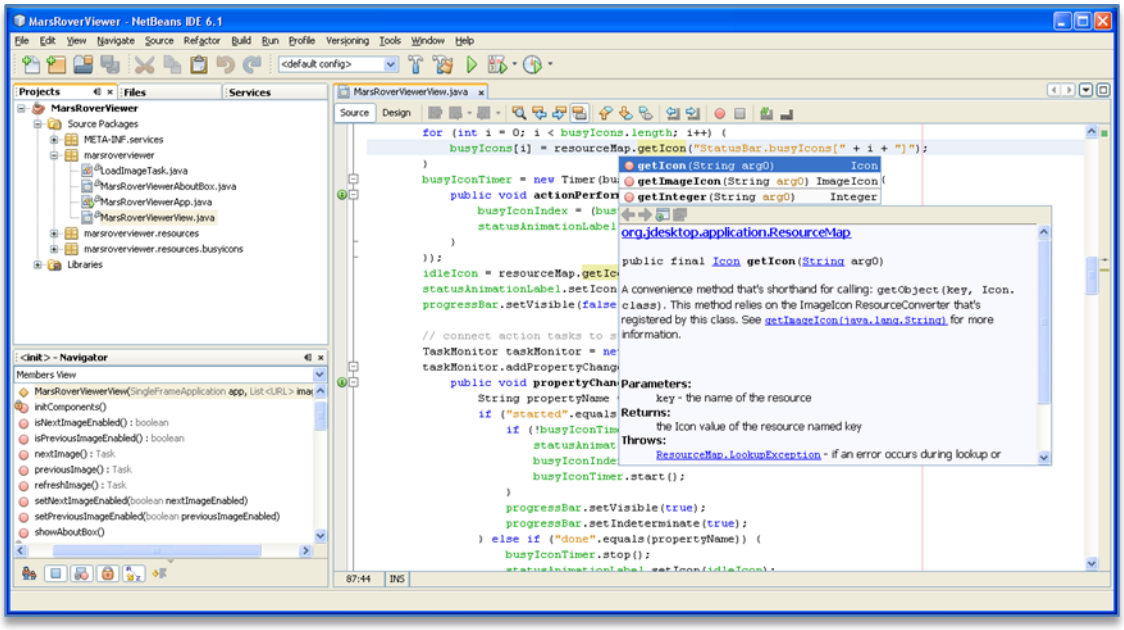


Figure 13. NetBeans Editor

The NetBeans editor supports several languages including Java, Ruby, C/C++, XML, HTML, RHTML, Javadoc, JavaScript, and JSP. It can be extended to support any other language.

The IDE parses the source code when being typed. This way the editor is able to immediately mark errors and highlight occurrences. Tooltips and editor glyphs unobtrusively display compiler hints, quick fixes, warnings, and language documentation.

The editor can generate code in Java or Ruby. A dialog is provided to give the developer complete control over which methods are created and which class attributes they access.

The User Interfaceof NetBeans is fully configurable.

Eclipse

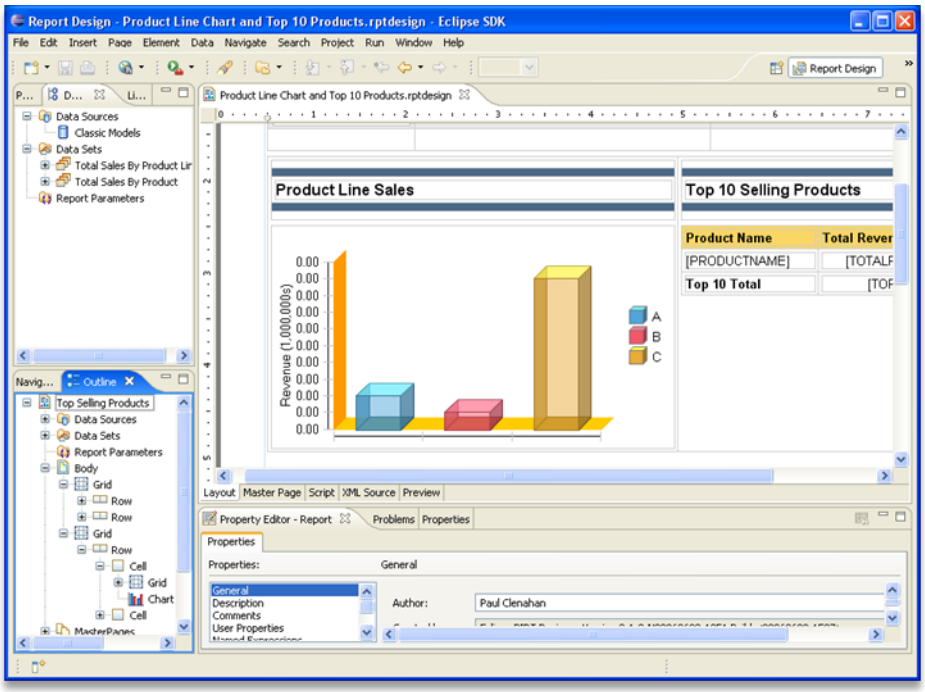


Figure 14. Eclipse IDE

The Eclipse IDE (Figure 14) employs plug-ins in order to provide all of its functionality on top of (and including) the runtime system, in contrast other applications where functionality is typically hard coded. The runtime system of Eclipse is based on Equinox, an OSGi standard compliant implementation.

This plug-in mechanism is a lightweight software componentry framework. In addition to allowing Eclipse to be extended using other programming languages such as C and Python, the plug-in framework allows Eclipse to work with typesetting languages like LaTeX, networking applications such as telnet, and database management systems. The plug-in architecture supports writing any desired extension to the environment, such as for configuration management. Java and CVS support is provided in the Eclipse SDK.

The key to the seamless integration (but not of seamless interoperability) of tools with Eclipse is the plugin. With the exception of a small run-time kernel, everything in Eclipse is a

plugin. This means that every plugin developed integrates with Eclipse in exactly the same way as other plugins. Eclipse provides plugins for a wide variety of features, some of which are through third parties using both free and commercial models. Examples of plugins include UML plugin for Sequence and other UML diagrams, plugin for Database explorer, etc.

The Eclipse SDK includes the Eclipse Java Development Tools, offering an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis. The IDE also makes use of a workspace, in this case a set of metadata over a flat filespace, allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards. The Visual Editor project allows interfaces to be created interactively.

Eclipse's widgets are implemented by a widget toolkit for Java called SWT, unlike most Java applications, which use the Java standard Abstract Window Toolkit (AWT) or Swing. Eclipse's user interface also leverages an intermediate GUI layer called JFace, which simplifies the construction of applications based on SWT.

2.2 Markup languages for GUI development

A widely accepted claim is that user interfaces should be separated from the logic of a program. A promising approach to achieve such a separation is representing the GUI in a markup language. Markup languages currently available for the representation of user interfaces are discussed in this section.

XAML

The Extensible Application Markup Language (XAML, pronounced zammel) is a declarative XML-based language created by Microsoft which is used to initialize structured values and objects. It is available under Microsoft's Open Specification Promise [50]. The acronym originally stood for Extensible Avalon Markup Language - Avalon being the code-name for Windows Presentation Foundation (WPF) [51]. XAML is used extensively in .NET Framework 3.0 technologies, particularly Windows Presentation Foundation (WPF) and Windows Workflow Foundation (WF). In WPF, XAML is used as a user interface markup language to define UI elements, data binding, eventing, and other features. In WF, workflows can be defined using XAML.

XAML elements map directly to Common Language Runtime object instances, while XAML attributes map to Common Language Runtime properties and events on those objects. XAML files can be created and edited with visual design tools such as Microsoft Visual Studio, and the hostable Windows Workflow Foundation visual designer. They can also be created and edited with a standard text editor, a code editor, or a graphical editor.

Anything that is created or implemented in XAML can be expressed using a more traditional .NET language, such as C# or Visual Basic.NET. However, a key aspect of the technology is the reduced complexity needed for tools to process XAML, because it is based on XML. As a result, a variety of products are emerging, particularly in the WPF space, which create XAML-based applications. As XAML is simply based on XML [52], developers and designers are able

to share and edit content freely amongst themselves without requiring compilation. As it is strongly linked to the .NET Framework 3.0 technologies, the only fully compliant implementation as of today is Microsoft's.

A XAML file can be compiled into a .baml (Binary XAML) file, which may be inserted as a resource into a .NET Framework assembly. At run-time, the framework engine extracts the .baml file from assembly resources, parses it, and creates a corresponding WPF visual tree or workflow.

When used in Windows Presentation Foundation, XAML is used to describe visual user interfaces. WPF allows for the definition of both 2D and 3D objects, rotations, animations, and a variety of other effects and features.

When used in Windows Workflow Foundation contexts, XAML is used to describe potentially long-running declarative logic, such as those created by process modeling tools and rules systems. The serialization format for workflows was previously called XOML, to differentiate it from UI markup use of XAML, but now they are no longer distinguished. However, the file extension for files containing the workflow markup is still "XOML" [53][54].

XAML uses a specific way to define Look and feel called *Templates*, different from the Cascading Style Sheets syntax, but closer to XBL [55]. This Windows Presentation Foundation example shows the text "Hello World!" in the top-level XAML container called Canvas.

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <TextBlock>Hello World!</TextBlock>
</Canvas>
```

The European Committee for Interoperable Systems (a coalition of Microsoft's largest competitors [56]) argues that Microsoft's use of XAML in its Silverlight product aims to introduce content on the web that can only be served from the Windows platform. XAML is viewable in non-Microsoft browsers on Windows and Mac, and Microsoft has provided direct support for the creation of a Silverlight viewer for GNU/Linux called Moonlight [57]. Microsoft's new XAML technology can be used to specify GUI (and much more) design in XML, and the tags directly map into C# classes and other MS languages.

XUL

The XML User Interface Language (XUL) is a markup language for creating user interfaces. It is a part of the Mozilla browser and related applications. It is designed to be portable and is available on all versions of Windows, Macintosh as well as Linux and other Unix flavours. XUL allows creating sophisticated applications without special tools.

It was designed for creating the user interface of the Mozilla application including the web browser, mail client and page editor. XUL may be used to create these types of applications. However, it may also be used for any web application. Like HTML, XUL can be used to create an interface using a markup language, using CSS style sheets to define appearance and

JavaScript for behavior. Programming interfaces are also available for reading and writing to remote content over the network and for calling web services. Unlike HTML however, XUL provides a powerful set of user interface widgets for creating menus, toolbars, tabbed panels, and hierarchical trees to give a few examples.

This means there is no need for third party code or large blocks of JavaScript in an application just to handle a popup menu. XUL has all of these elements built-in. In addition, the elements are designed to look and feel just like those on the user's native platform, even supporting OS level themes in Windows XP and MacOS X. Alternatively, standard CSS can be used. The XUL widgets also support localization and have support for accessibility using OS level accessibility interfaces.

XUL is an XML language and numerous existing standards including XSLT, XPath and DOM functions can be used to manipulate a user interface. In fact, XUL is powerful enough that the entire user interface in the Mozilla application is implemented in XUL.

In addition to many built-in user interface widgets available in XUL, additional custom widgets that use related language, called the Extensible Bindings Language (XBL), can be used. This language may be used to create custom tags and implement custom functionality.

XUL applications may be either opened directly from a remote Web site, or may be downloaded by the user and installed.

XUL may also be used to create standalone applications that may be used as part of the browser. A feature in XUL called an overlay allows a third party to create extensions to the browser itself, for example to add a custom toolbar, change menus, or add other features. This feature is popular in Mozilla Firefox -- there are almost 100 extensions available. Mozilla's upcoming mail client Thunderbird also has a number of extensions available. In fact, any XUL application can support extensions. XUL is a spinoff of the Mozilla browser user interface project that has inspired many similar projects.

XForms

XForms is an XML application that represents the next generation of forms for the Web. By splitting traditional XHTML forms into three parts—XForms model, instance data, and user interface—it separates presentation from content, allows reuse, gives strong typing—reducing the number of round-trips to the server, as well as offering device independence and a reduced need for scripting. XForms is not a free-standing document type, but is intended to be integrated into other markup languages, such as XHTML or SVG.

Forms are an important part of the Web, and they continue to be the primary means for enabling interactive Web applications. Web applications and electronic commerce solutions have sparked the demand for better Web forms with richer interactions. XForms 1.0 is the response to this demand, and provides a new platform-independent markup language for online interaction between a person (through an XForms Processor[21]) and another, usually

[21] http://www.w3.org/TR/xforms/#def-XForms-Processor

remote, agent. XForms are the successor to HTML forms, and benefit from the lessons learned from HTML forms.

BML

An XML-based component configuration or wiring language customized for the JavaBean component model. Bean Markup Language has retired.

JEasy[22]

Create complete Java applications quick and easy with JEasy. JEasy uses the new Swing components and is based on JAVA 2. JEasy makes a class library available to produce dynamically the most important swing components. All the GUI-components of an application are stored in a project inside the JEasyRepository. The repository helps specifying the properties of the Swing components and the relationship between them. JEasy presents the most important swing components with the most usual properties. A preview function shows the layout at the screen without having to write one line of code. Sample programs help with the first steps. It is easy to create or copy menus, frames, panels, data fields and other components and to give a first layout of a program. Coding just contains the handling of the events released by the user. A modification of the layout or the translation into another language does not have effects on the source. Special additional JEasy objects such as message were developed to serialize data out of forms. Look at RunExample.java and see how to transfer a complete address form from a store to a panel with two lines of code.

JBeaver[23]

JBeaver 1.4 allows to work as follows: (1) Widgets are dropped on the panel following the intended layout approximately (2) JBeaver calculates the appropriate FormLayout specifications and constraints (3) The user can verify the layout, fine-tune FormLayout settings and specify widget properties. JBeaver includes support for table layout.

text2gui

With Text2gui, an entire GUI hierarchy can be described with a simple syntax, which is much easier to write than Java. There is no need to recompile the application to make changes to the GUI, which saves much development time. Also, GUI descriptions are stored in resource bundle files, so this makes internationalizing the application almost trivial.

2.3 GUI Generators

Graphical User Interfaces (GUIs) are difficult to build. Even more difficult is to produce GUIs that are usable and easy to maintain. In particular, in the development of GUIs, it is very important to appropriately separate the GUI interface processing from the logic of the application.GUI Generators are engines that produce GUIs based on a markup language description, therefore greatly simplifying GUI development. They are also useful in maintaining GUI and logic separation.

[22] www.jeasy.de
[23] www.ratundtat.com

This section discusses a number of GUI Generators. The AmI Designer along with the AmI Player can be considered as a kind of GUI Generator, since, in order for the GUI to be created, AmI Designer generates the GUI's description and AmI Player needs to be used instead of an executable file.

SwixML

SwiX^{ml}, is a small GUI generating engine for Java applications and applets. Graphical User Interfaces are described in XML documents that are parsed at runtime and rendered into javax.swing objects.

Depending on the application, XML descriptors may be deployed with the remaining code or loaded from a remote server at runtime. This late binding of the GUI has many advantages. Enabling features in an application based on a license code or a user's role does not have to be hard coded anymore. Instead an XML document describing the application's GUI could be dynamically loaded. Generating the GUI based on descriptors also has some of the advantages that code generators provide, but without generating the non-maintainable code.

While **SwiX^{ml}** doesn't free the developer from knowing the *javax.swing* package, it reduces the amount of repetitive, sometimes error prone, and often complex GUI related code.

Thinlet

The original Thinlet GUI toolkit is a single Java class, thus it is extremely small (only 39KB) and the hierarchy and properties of the GUI is described by an XML file instead of the usual JavaBean components. The author's philosophy is to aim it at handheld devices. This is worth a look if interested in something small, and not based on Swing.

SwingML

SwingML is an open source framework that allows JFC/Swing[24] user interfaces to be defined using XML and J2EE programming style. Browser based applications are arguably the most popular way to deliver applications to the users desktop. Javascript coupled with AJAX technology is helping to provide users with a "rich" user interface experience. However, applications with heavy usage patterns are still cumbersome with a browser based UI, even supported with AJAX. There's still a place for user interfaces that can take advantage of repsonsivness and richness of client side user interface widgets. Looks good, but it isn't obvious how to connect the GUI to a Java application. This seems to be oriented toward Applet server communication.

CUF

CUF (Client Utilities & Framework) contains: utility libraries (this is the "U" part) and an application-level framework (the "F" part) for building GUI applications in either Java/Swing or .NET/WinForms environments. It is build to provide: practical help for details (the libraries part) as well as guidance and structure (the framework part) for developers of rich-client

[24] http://en.wikipedia.org/wiki/Swing_(Java)

applications. Using CUF, a developer can build a Java/Swing or a .NET/WinForms application more easily. This is especially true if the application is build in a business environment with a focus on forms based data manipulation ("SCRUD": search, create, read, update, delete). All libraries of CUF are independent of each other, and most of them are available both for .NET/WinForms as well as for Java/Swing. CUF (Client Utilities and Framework) provides an XML representation of the GUI that will work with either Swing or .NET. Interesting but there's no documentation.

AUIML

The AUIML Toolkit provides software development tools that allow developers to write an application once and run it in Java™ Swing or on the Web without any changes. The AUIML Toolkit includes the AUIML VisualBuilder, which is an Eclipse-based visual panel editor built on top of the Eclipse Visual Editor Project. The AUIML VisualBuilder allows developers to easily build user interfaces and generate Java™ data and event-handling code for them. Additional Java code can be written to AUIML's API in order to control application flow, data validation, and to listen for events. Once the application is implemented, it can be deployed as a Java Swing application or as an HTML servlet without changing the application's code.

The current release of the AUIML Toolkit includes both a Java Swing Renderer and a Web Renderer. The Swing Renderer provides rich-client functionality for installed applications by using Java Swing technology to display the panels. The Web Renderer runs as a servlet and sends an HTML representation of the panel to the client browser, allowing the same application to run on the Web without any source code changes. This toolkit runs on certain versions of Windows® and Linux® (see requirements).

AUIML is an XML dialect that is a platform- and a technology-neutral representation of panels, wizards, property sheets, etc. AUIML captures relative positioning information of user interface components and delegates their display to a platform-specific renderer. Depending on the platform or device being used, the renderer decides the best way to present the user interface to the user and receive user input.

The AUIML XML is created using the Eclipse-based AUIML VisualBuilder, which allows a developer to quickly build and preview user interfaces in the Java Swing and Web renderers. The AUIML VisualBuilder can also automatically create application launchers, data beans, event handlers, and help system skeletons for the user interface. Because it plugs into Eclipse, building the user interface and application code is an integrated process.

Includes an interface builder, and same interface can be used either as Swing or HMTL servlet. It works with the Eclipse Visual Editor.

RIB

The IBM Reflexive User Interface Builder (RIB) is an application that constructs and renders graphical user interfaces (GUIs) for Java™ Swing and Eclipse Standard Widget Toolkit (SWT) based upon a descriptive XML document. (Java Swing is a rich GUI toolkit included with Java that provides operating system-independent GUI components. Eclipse SWT is an add-on GUI toolkit that takes advantage of host operating system GUI components for maximum host

integration.) RIB is both a specification for a mark-up language in which to describe GUIs and an engine for creating (and, if desired, rendering) them. This application can be used as a stand-alone application for testing and evaluating basic GUI layout and functionality, or it can be used as a library within the context of a Java application for creating and rendering GUIs for that application.

Version 1.0.1 of RIB marks a split of functionality. The validation function has been moved to RAVEN, a new alphaWorks technology. RIB is now specifically for creating and rendering Java rich-client GUIs based on a description specified in an XML document. This technology runs on Java 1.4.2 and above and Eclipse 3.0.1 and above. RIB currently supports only Windows NT and Windows XP.

Although other XML script-driven, Java-based GUI engines have been developed, the power of RIB's GUI creation approach stems from its reliance upon the Java Reflection API, which allows classes to be introspected in order to reveal their fields, constructors, and methods. Proceeding in this way, RIB can create and render GUIs based solely upon the information in an XML document. This document need not conform to any pre-defined DTD or XML schema.

Further information is available in this article: Introducing the Reflexive User Interface Builder. Please note that this article is based upon an older version of RIB.

www.alphaworks.ibm.com/tech/rib - Reflexive User Interface Builder is another IBM approach using XML to represent a Swing / SWT user interface.

Xmlgui

The Beryl XML GUI library was written to ease the development of graphical user interfaces using Swing on Java. It lets you store user interfaces as XML markup. This will help you avoid unnecessary clutter in your source - Swing code mixed with application logic can become a troublesome and hard to read mess as the application size increases. The library comes with a visual component builder, which makes development a breeze.

The mission of this project is to create a framework which makes the use of the bare Swing API unnecessary. Any steps to further simplify the construction of user interfaces will be taken. However, there will under no circumstances be any simplifications which limit the underlying API. This project will always stay under the liberal LGPL license so that it can be used in commercial projects.

xmlgui.tigris.org – It is interesting but with a little documentation. Moreover it provides an editor, defines additional useful widgets and has been used in real project. However it is not appropriate for simplifying student programs.

SpeedJG

SpeedJG is an XML-based GUI builder tool to create state-of-the-art Java Swing GUI applications, user interfaces, and frontends. The core part of this Java GUI builder is a parser that reads the meta-data described in XML to create Java GUI components on the fly. An

IDE, itself generated by and using this parser, enables the Java developer to design and rapid prototype a GUI, generate the meta-data, check the layout, and create the corresponding source code.

This GUI editor is not bound to a specific Java software development environment. It can be used together with any currently established Java IDE or simply on its own. The meta data to describe the GUI layout is stored as XML because the structure of XML ideally fits to the hierarchical structure of Swing (JFrame, JPanel, JComponent. etc.). In addition, this format is readable on any platform. Therefore you are able to share your Swing GUI e.g. with your friends and / or colleagues regardless which platform or Java IDE they are using. By default this Swing GUI builder generates pure Java source code that is also executable without any .jar file to be licensed. Thus you don't have to study any new APIs when developing a Java GUI with SpeedJG. This GUI builder enables you to create complex GUIs because the structure of the components used corresponds to the structure of XML. Thus you can simply design multiple nested panels with different layouts (Swing is not VB). To see an example of a complex Swing GUI look at SpeedJG - the GUI of this application is entirely generated at runtime by this Java GUI builder too. With SpeedJG the developer of a Swing GUI is focused on the main properties when customizing a component. He is not to overstressed with all possible properties from the inheritance hierarchy in alphabetical order. Instead, only those properties which are relevant in respect of the component currently to be customized are presented and ordered by importance. At any time, the layout and appearance of any (not only the top-level JFrame or JPanel) component can be checked without having to compile it before. This is done at the push of a button by interpreting the meta data stored as XML. When the GUI is finished its source code can be exported into a Java source file and compiled. This Java GUI builder enables editing in parallel several Swing components. Each component is edited within a tab of its own and can be checked for itself. After saving, the user directly changes to another tab, and if this is, for example, an editor of a parent component, he can check the effects in a superordinate context. When generating Swing source code this Java GUI builder strictly follows the MVC approach by separating the GUI (view) code from the controller code, and the model code, that's up to the developer.

To give an example of a multilingual Java GUI, this separation allows simplifying delivering the complete GUI object to a translator class that can access all the components by their name without any knowledge of the internal structure, and set the texts of the labels, buttons, frame titles etc., depending on the preferred user language. When exporting the generated Swing source code into a file, this Java GUI builder by default only overwrites the previously generated code lines. Thanks to the clear-cut MVC separation you don't have to modify within the generated code lines but only within the stubs offered outside the generated code. Thus if you modify the layout of your GUI with this Swing GUI builder and re-generate the code, your individually added Java code lines handling the GUI access remain untouched and valid. Java Swing source code can be generated not only for top-level JFames or JDioalogs but basically for all components. When developing a very complex GUI this feature helps - in conjunction with the MVC separation - to delegate self-contained GUI functionality to separate classes that handle parts of the whole application within their own responsibility.

2.4 Intergrated Tools

xWidglets

xWidglets, is a declaratory language of description of graphical interfaces based on XML created by the company Dpi SoftWare. xWidglets includes and understands a whole of beacons making it possible to define all the elements useful for the use of an advanced graphic application such as buttons, lists, menus, or of the zones of edition. This possibility is facilitated by the use of development tools of the type RAD what facilitates the work of the developer and makes it possible to write the interface of an application as easily as a Web page.

The development of a xWidglets application is identical to the development of an application traditional Client-serveur. With each graphic object (windows, buttons, seizable tables, template, etc.) are associated events (click of a mouse, double-click, etc) for which the developer can choose to write code (Java) which will be carried out by the machine customer. The application can thus communicate (via requests HTTP) with a Web server but also with any type of waiter of application J2EE or .Net.

Description in format XML of the pictorial display of the screens offers a visibility and a reutilisability of the presentation part of the applications and this, whatever the platform used. There exist also similar tools such as XUL.

GUI editor produces XML UI representation. Free download includes editor, gui engine, documentation. The web site is in French, but at least some of the screenshots show English menus. It's not clear how this is used as an interface to a Java application (my interest), but an example is given using JavaScript.

FormBuilder

FormBuilder is designed to handle all parts of a CGI form request - generation, submission, and validation. As such, FormBuilder applications will usually loop back on themselves. That is, you will have a single script that, when invoked, will take care of your entire application just based on the CGI parameters. FormBuilder uses an Object-Oriented calling-style. Usage is exceedingly easy, as there are only a couple functions the developer needs to know about. The SwingEmpire FormBuilder GUI editor builds on the excellent JGoodies layouts, but it doesn't seem mature yet.

Java-XML-GuiBuilder

The Java-XML-GuiBuilder creates on the basis of XML documents a runable Java swing surface. These tools are meant for the production by horizontal prototypes, the specification/documentation of the user's interface, the runtime to. It uses GUI builder from XML and is open source.

XUI

XUI is a Java and XML framework for building rich-client applications. XUI incorporates many features designed to make the task of building applications quick and easy. XUI is a Java and XML framework with which you can build: Desktop applications, Handheld applications,

Mobile applications, Web and Enterprise applications and Standalone applications. XUI supports Swing and AWT and other widget sets. XUI has a range of standard components. You can easily extend the components, add new components or register third party components. It seems to be actively worked on and it doesn't appear to make small programs simpler. This is not the same as the XUI project listed below.

XMLTalk

XMLTalk is a Java/Swing user interface framework that allows developers to specify the entire user interface in an XML file instead of Java code. More importantly, XMLTalk specifications also specify how the user interface is connected to the application. It targets to handheld devices.

KoalaGML

KoalaGML, the koala GUI Markup Language, allows developers to rapidly generate graphical user interfaces by writing XML documents that define the layout and content of the interfaces. It is both a markup language and a toolset for generating the Java GUI code from the markup. Inspired by Java Server Pages (JSP), koalaGML consists of a rich and growing set of features, including: session beans, custom imports, exception handling, custom error forms, relative layout of elements (with tables ala HTML) and a huge set of widgets.

Ultrid

Ultrid is an ongoing development supporting a variety of layouts, including table layout, splash screens, wizards, localization, and much more. There is strong support for various scripting languages.

2.5 Discussion

As demonstrated by the wide variety of tools reported in the previous sections of this Chapter, the need of providing user interface development support tool is widely felt, and many options are available for speeding up and simplifying the design and implementation of GUIs in the desktop and mobile environments.

Design is usually supported through WYSIWYG editors which allow designers to perform rapid prototyping visually. Such editors may be standalone or embedded in integrated environments, and can produce user interface code or a description in a UI markup language. Various markup languages are also available. Finally, based on the markup description, UI generators can produce a running user interface automatically, thus saving much programming efforts.

Although automatic code generators are often criticized for the quality of the code they produce, the combination of a WYSIWYG GUI Editor with a markup description and a an automatic GUI generator appears currently to be the more promising approach towards rapid and efficient development of user interfaces.

However, no tool currently available offers the possibility of creating GUIs for AmI services provided through various devices in a smart home environment, as no tool is appropriately equipped for communicating with such an environment. Additionally, another very important limitation of available tools is that they do not produce accessible user interfaces, and therefore they are not suitable for addressing the mainstream accessibility requirement posed by AmI environments.

Finally, many of the described tools are bound to specific development platforms or interaction object toolkits. In AmI, however, it is particularly important to ensure that new interaction metaphors, appropriate to the new types of applications that the AmI environment integrates, can be elaborated, evaluated and used. Therefore, the extensibility of the basic toolkit is particularly important when designing for AmI.

The AmI Designer and AmI Player developed in the context of this thesis have been designed and developed in order to address the above gaps in the current state of the art. Their design will be presented in Chapter 4, and their implementation in Chapter 5. They provide all the necessary features to allow accessible user interfaces to be integrated in the ambient environment.

3. Design

The purpose of this chapter is to describe the design of the AmI Designer and AmI Player, focusing on how they support the easy development of graphical user interfaces for ambient intelligence environments. The Chapter starts with an overall description of developed tools, and than by a detailed description of both the AmI Designer and AmI Player, reporting their functionality, user interfaces and outcomes.

3.1 User Interfaces generation for Smart Home Applications

The AmI Designer and the AmI Player have been specifically developed to achieve the following objectives:

- support the easy and quick development of user interfaces, along the lines of the GUI generators discussed in Chapter 2.
- automatically integrate the developed interfaces in smart home environments without the need of further programming
- ensure that the user interfaces developed by means of the AmI Designer and run by the AmI Player are fully accessible and usable for all final users, including disabled and elderly people, thus providing a convenient way for final users to manipulate all the diverse devices in the environment through consistent interaction.
- ensure the extensibility of the tools themselves to additional interaction styles and additional services, as well as potentially additional interaction platforms.

Towards achieving the above, the AmI Designer allows easily composing user interfaces, and automatically discovers existing services in the ambient environment, supporting the developer to bind those services with the user interface.

The interfaces that are generated by the AmI Player can accept / send messages from / to the ambient environment and can be used by motor and visually impaired users through different interactive devices. These interfaces can also be remotely used. Besides the visual representation there is also audio feedback. The applications that are generated do not use standard operating system widgets such as windows, but the tool users can develop their own widgets if they wish. This possibility is provided by the dynamic architecture of the system. A big advantage is that the user can create fancy widgets and user interfaces. In the current implementation, the AmI Designer application is limited to the MS Windows platform due to the usage of the .NET platform. Nevertheless, the AmI Player should can potentially run on various operating systems with minor changes. Future extensions of the tools will address multiplatform development.

The following is a summary of the main features of the developed tools.

The AmI Designer is an editor which creates descriptions of a designed room in XML. The designer has different widgets available to design a user interface. These widgets are

detected dynamically from the editor. Developers can program their own widgets as far as they follow a specific architecture which is described in Chapter 4. AmI Designer communicates with the Middleware, is continuously informed about the available services of a room and can find the Middleware signals. Therefore, the designer can define how the application manipulates the devices in the room and how the external environment interferes with the GUI. Additionally, the designer can set the widgets' scanning order in the accessible version of the user interface. By using AmI Designer, it is quick and easy to design user friendly GUIs. AmI Designer is targeted to computer expert users.

The AmI Designer automatically discovers existing services in the ambient environment and allows the user to bind those services with the user interface. The interfaces that are generated by the AmI Player can accept / send messages from / to the ambient environment and can be used by motor and visually impaired users through different interactive devices. These interfaces can also be remotely used. Besides the visual representation there is also audio feedback. The applications that are generated do not use standard operating system widgets such as windows, but if users want to they can develop their own widgets. This possibility is provided by the dynamic architecture of the system. A big advantage is that the user can create fancy widgets and user interfaces.

The AmI Player uses the export XML description of the AmI Designer to generate GUIs. AmI Player sends and receives messages and signals from the middleware, calls services and uses devices. The exported GUI is manipulated by several input devices. The design of the generated application depends on the designer's design. AmI Player's generated GUI is accessible from all, such as hand-motor and visual impaired users, children, elder, inexperienced and simple users.

The problem that the tools address is the lack of a system that generates accessible GUIs that interact with Smart Environments and let the user to manipulate the devices that exist. Alternatively, the user had to create different applications for each kind of device. For the end-user, the main advantage is the ability for all, including disabled users, to use the application and manipulate the Smart Environment.

3.1 AmI Designer

The design of the AmI Designer followed a user-centered methodology. The target user group was considered to be experienced but also beginner designers and developers of Smart Home applications.

Two brainstorming sessions with 3 expert user interface designers / developers were conducted in order to capture user requirements for the tool. In the first session, an initial list of ideas was elaborated. In the second session, ideas were ranked and a final list of requirements was produced. These included:

- Organization of the design based on the home environment rooms, in order to find the services of each room so the applications can control the specific rooms.

- AmI Designer's interface was designed in way that can produce any action with the less steps required.

- The way that the editor is divided in areas it is influenced from the Visual Studio .Net IDE since it has the best editor. So in the left area is the tool box, in right area is current item's properties and finally at the center we can see the stage.

Subsequently, a task analysis was conducted, which led to the identification of the tools' functionality. Following the elaboration of a first partially functional prototype, expert formative evaluation was conducted with 4 user interface designers / developers. The evaluation followed a think aloud protocol, and led to the final design of the tool.

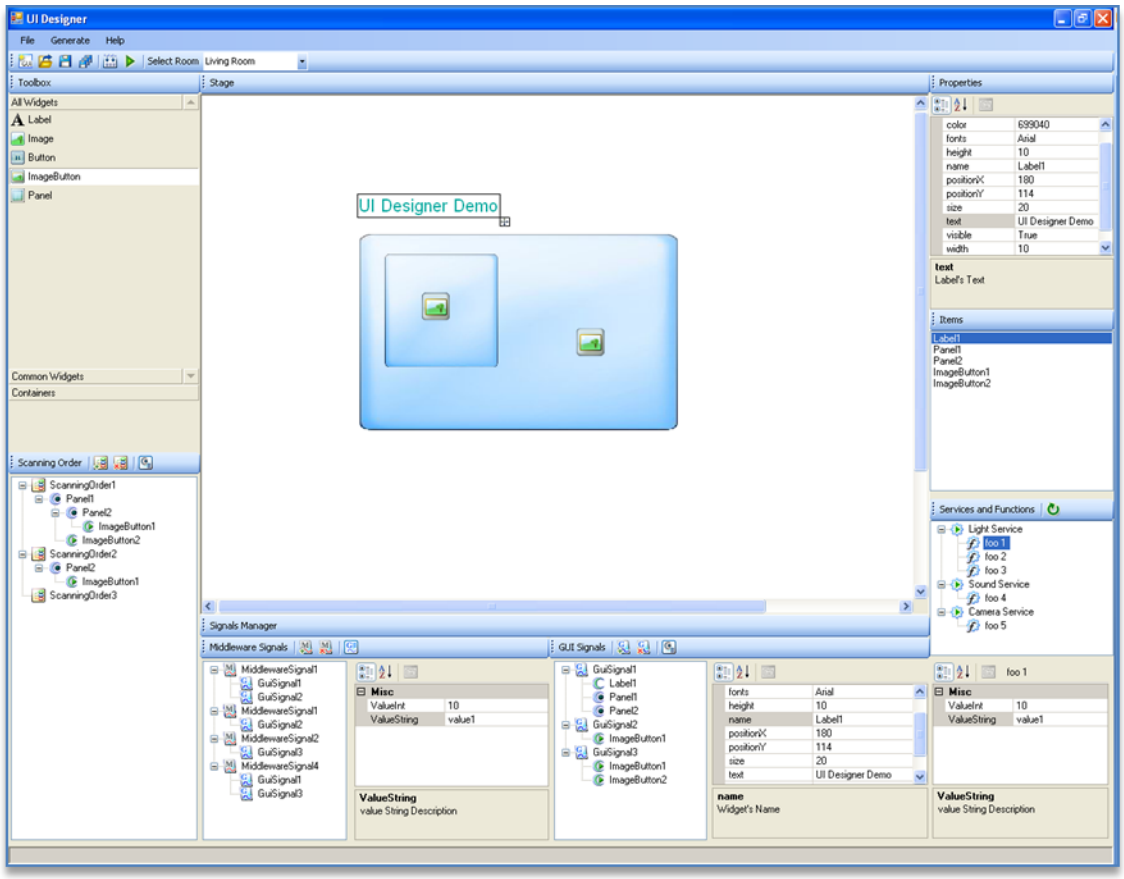
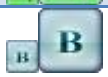
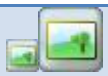


Figure 15. AmI Designer's Graphical User Interface

Figure 15 above depicts the graphical user interface of the AmI Designer. The interface is designed so as to allows AmI Designers to access all needed information and functionality through a single screen, ensuring that any user action takes no more than two steps. Each object used within the application has a specific icon associated with it so that it is easier for users to learn how to use the application since for same objects the same icons are used.

Table 1 below presents all icons used within the interface of the AmI Designer.

Table 1. GUI's Symbols

Symbol	Name	Description
	Create GUI	Creates a new user interface, requires from the user to specify the desired dimensions.
	Open	Opens an existing user interface.
	Save	Saves the current user interface.
	Save as	Saves the current user interface using a custom name and path.
	Generate XML	Generates an interface description. This description is an XML file that describes the accessible interface and its interaction with the environment.
	Play Interface	Starts the AmI Player and generates the interface.
	Help	Opens the help documentation of the application.
	Label	Allows users to insert text in the interface.
	Image	Allows users to insert images in the interface.
	Button	Allows users to insert buttons in the interface. These buttons can be used to call services using specific arguments or to alter elements in the user interface.
	Image Button	Allows users to insert image buttons in the interface. These buttons can be used to call services using specific arguments or to alter elements in the user interface.
	Panel	Allows users to insert panels in the interface. These panels can be used to group various items.
	Empty Widget	Items that users cannot interact with, used for decorative or informative purposes only. Such widgets are labels and images.
	Action Widget	Items that users can interact with to call services using specific arguments or alter elements in the user interface. Such widgets are buttons and image buttons.
	Container Widget	Items that are used to group other items. Items can be attached or detached from a container at any time.
	GUI Signal	When produced, GUI signals modify properties of items within the current stage.
	Add GUI Signal	Inserts a new GUI Signal with a specific name.
	Remove GUI Signal	Deletes a GUI Signal.
	Attach Item	Attaches a selected item to a container widget.
	Detach Item	Detaches a selected item from a container widget.
	Middleware Signal	Signals that are transmitted from the middleware with specific arguments to produce GUI Signals. Middleware is the means of communication between the ambient environment and the

		applications.
	Add Middleware Signal	Adds a middleware signal from a specific list that is created by a query to the middleware.
	Remove Middleware Signal	Removes a middleware signal.
	Change GUI Signals	Changes GUI Signals that are related to specific middleware signals.
	Scanning Order	All stage items can be selected to create an accessible user interface. Users can access items serially according to the scanning order already defined.
	Add Scanning Order	Creates a new scanning order using a specific name.
	Remove Scanning Order	Deletes a selected scanning order.
	Change GUI Items	Modifies items within a scanning order or a GUI Signal.
	Refresh Services	Refreshes the list of services and functions of the selected room.
	Service	A service that is available in the selected room.
	Function	A function of a specific service.
	Bound Function	Shows the function of a service that is called by an action widget.
	Unbind	Unbinds an action widget from the modification of an item or the calling of a service.

3.1.1 AmI Designer's main areas

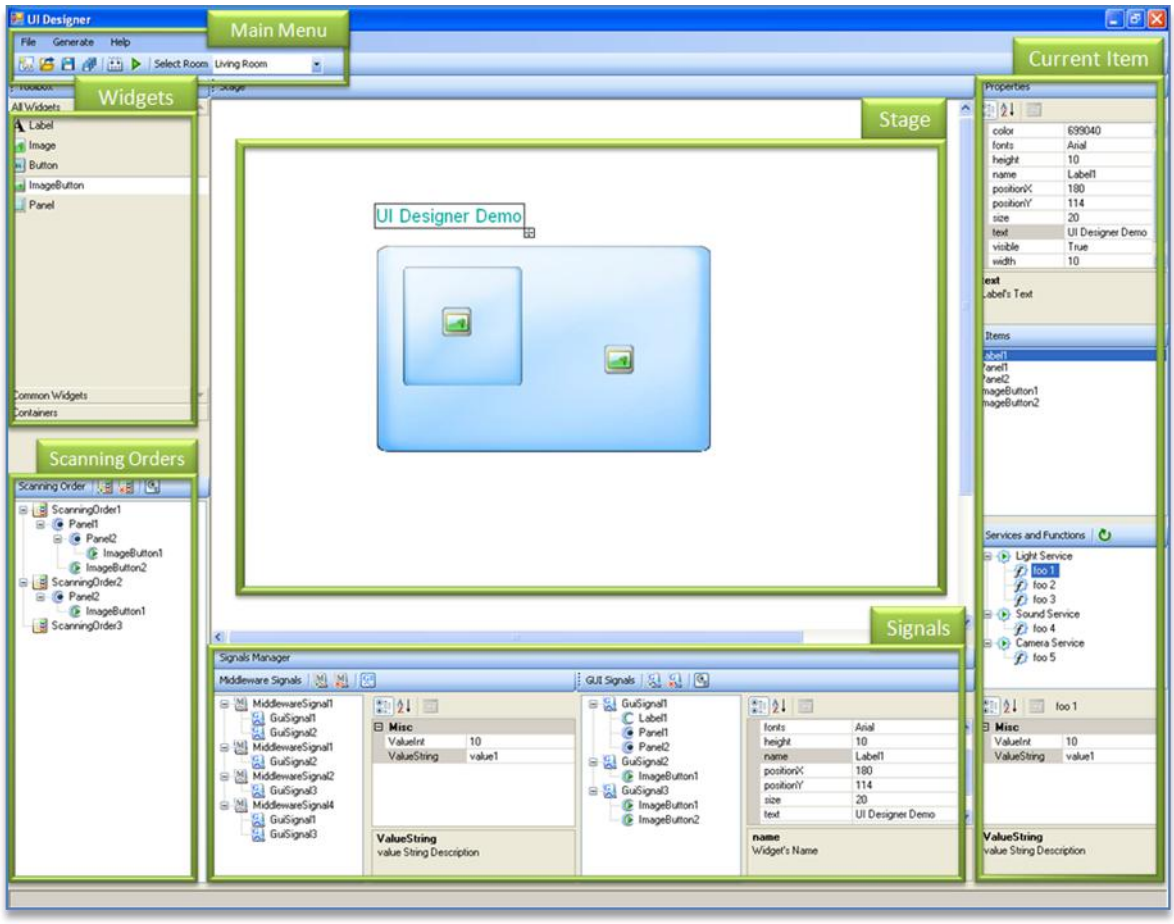


Figure 16. AmI Designer's Areas

Figure 16 presents the main areas of AmI Designer user interface (within green frames). These areas, which will be described in further details below, are: main menu, widgets toolbox, stage, current item editor, signals editor and scanning order editor.

3.1.2. Main Menu

Figure 17. Main Menu Overview

The main menu (see Figure 17) includes three items: File, Generate and Help. The File menu (see Figure 18a) is used to create a new UI, to open an existing one and save the currently open one. Through the Generate menu (see Figure 18b) users can generate an XML file that describes the UI or call the AmI Player. Finally, through the Help menu (see Figure 18c) they can open the help documentation of the tool.

Figure 18. Tool Strip Menus: (a) File, (b) Generate and (c) Help

Figure 19. (a)Generate XML and (b) Play Interface

Alternatively to the menu, the buttons on the toolbar placed right under the menu bar can be used to accomplish the aforementioned actions quickly. Figure 19 shows the buttons used for XML generation and calling of the AmI Player.

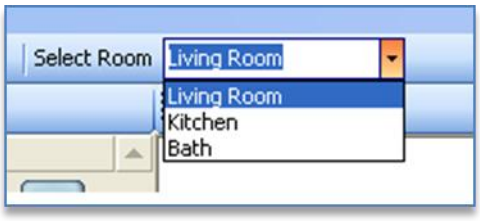


Figure 20. Select Room

In addition, on the menu bar there is a drop-down list through which users can select the room for which they wish to design an interface (see Figure 20). This functionality is provided to the user because each room is likely to contain different services since devices differ from room to room in the smart house. Thus, selecting a specific room results in refreshing the list of services and functions specific to that room.

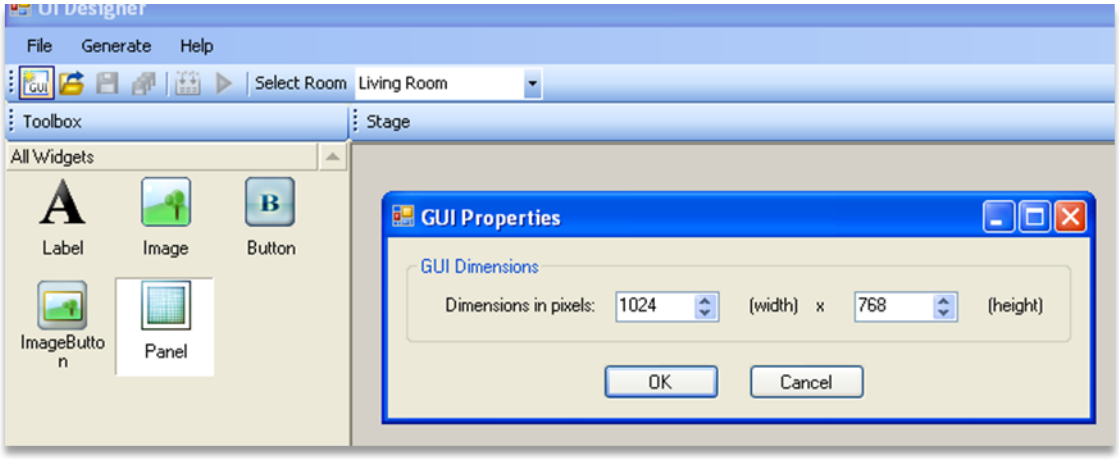


Figure 21. Create New GUI

When a user clicks on the "Create GUI" button a new window appears (see Figure 21) asking details of the interface dimensions (in pixels).

3.1.3. Widgets

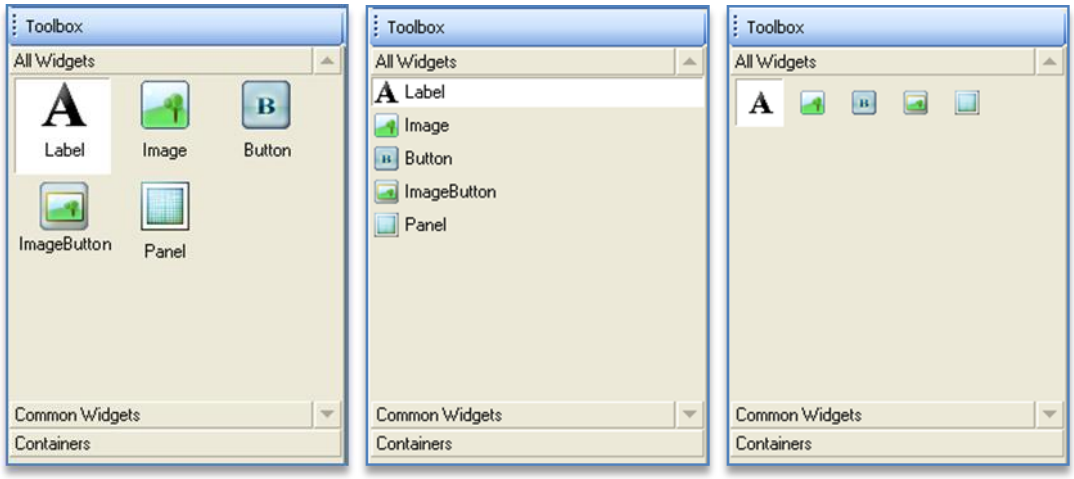


Figure 22. Toolbox Different Views of Widgets: (a) Large Icons, (b) List and (c) Small Icons

Figure 22 shows the toolbox containing the widgets that can be used to design interfaces. There are three different ways of presenting the available widgets within the toolbox: (a) large icons, (b) list and (c) small icons.

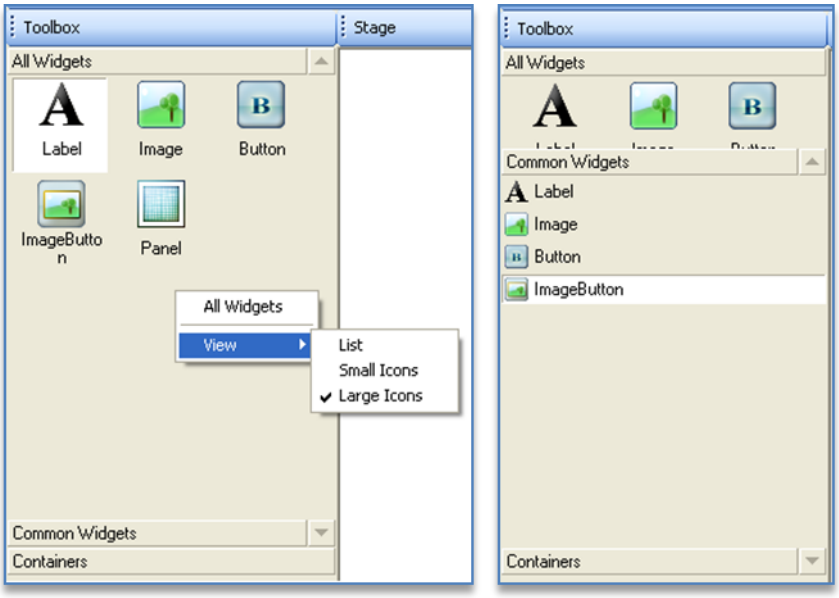


Figure 23. Changing Toolbox (a) View and (b) Panel

The user can change the widgets view at any time by right-clicking and selecting from the "View" menu the desired style (see Figure 23). Also, the toolbox consists of different panels containing different widget types. Three panels are used in the toolbox: the first one contains all the widgets, the second one contains the common widgets and the last one shows the container widgets. Each panel can be viewed by clicking on the respective panel title. As previously mentioned, each user can create new widgets. The dynamic discovery of the available widgets will be discussed in Chapter 4 which reports the tool's implementation.

To insert a widget in the GUI being created, the corresponding item must be dragged from the toolbox and dropped on the stage. The only actions that can be taken within the stage are creation of new items, as well as moving and resizing of existing items.

3.1.4. Current Item

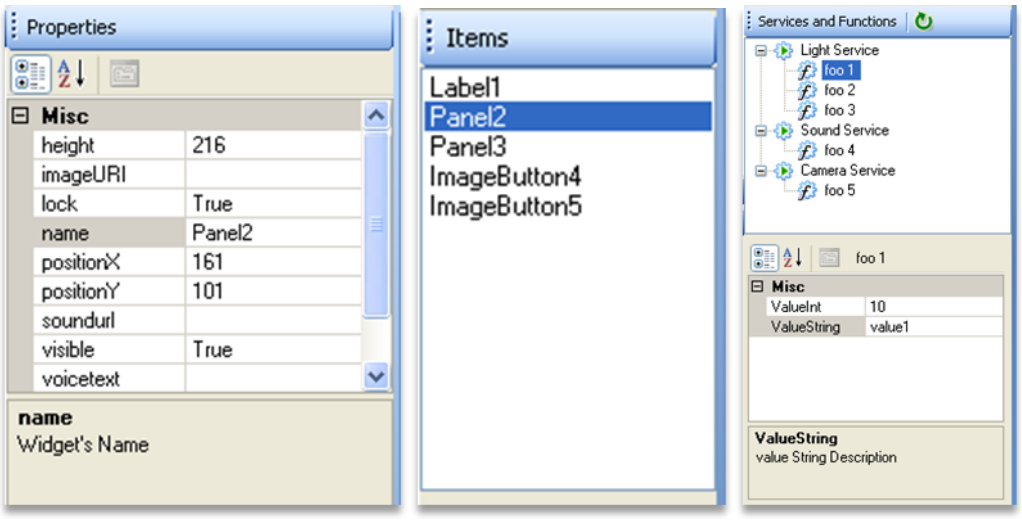


Figure 24. Current Item: (a) Properties, (b) Selection and (c) Services & Functions

In the current item area on the right side of the interface the following information is displayed: (a) the properties of the currently selected item on the stage, (b) a list of all the items within the stage with the currently selected item marked, and (c) a list of all services and functions, as well as the arguments of the currently selected function (see Figure 24).

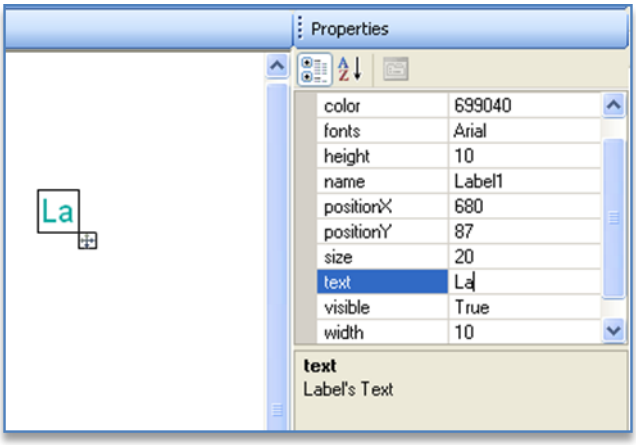


Figure 25. Change Item's Properties

Properties vary according to each item category, and include name, size, dimension and other properties that invent dynamically. An important property is the voice text associated to an item, which is used at run time to "speak aloud" the interface so as to provide non visual interaction (for example, for blind users, or for users busy with other tasks in the environment). The user can modify the properties of the selected item by clicking on the respective field of the properties grid. The properties grid presents all items with their values, as well as the value of the selected property (see Figure 25). In case the user enters an invalid value, an error message appears and the respective value is reset to its previous state. Changes in values are immediately applied to the items in the stage. When an item is selected in the stage, the property grid is automatically updated. This also happens when a selected item is resized or moved.

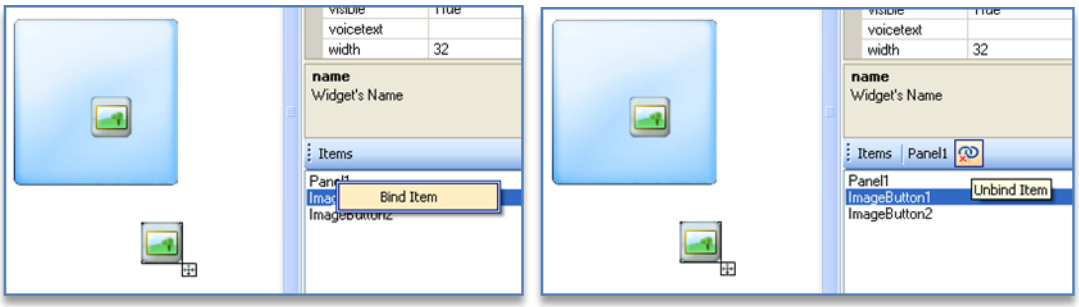


Figure 26. (a) Bind and (b) Unbind an item with / from an action widget

As previously mentioned, action widgets can be used to modify interface elements. More specifically, they are used to alter the visibility state of container widgets. If the selected item is a widget, then right-clicking on any item in the list is allowed except for the selected one. Immediately a menu appears ("bind item") that allows the selected action widget to bind another item, implying that the action widget will change the items visibility status. In case the selected item is an action widget and is already bound to another item, then the

other item's name appears above the list, as well as a button that allows the user to unbind it (see Figure 26).

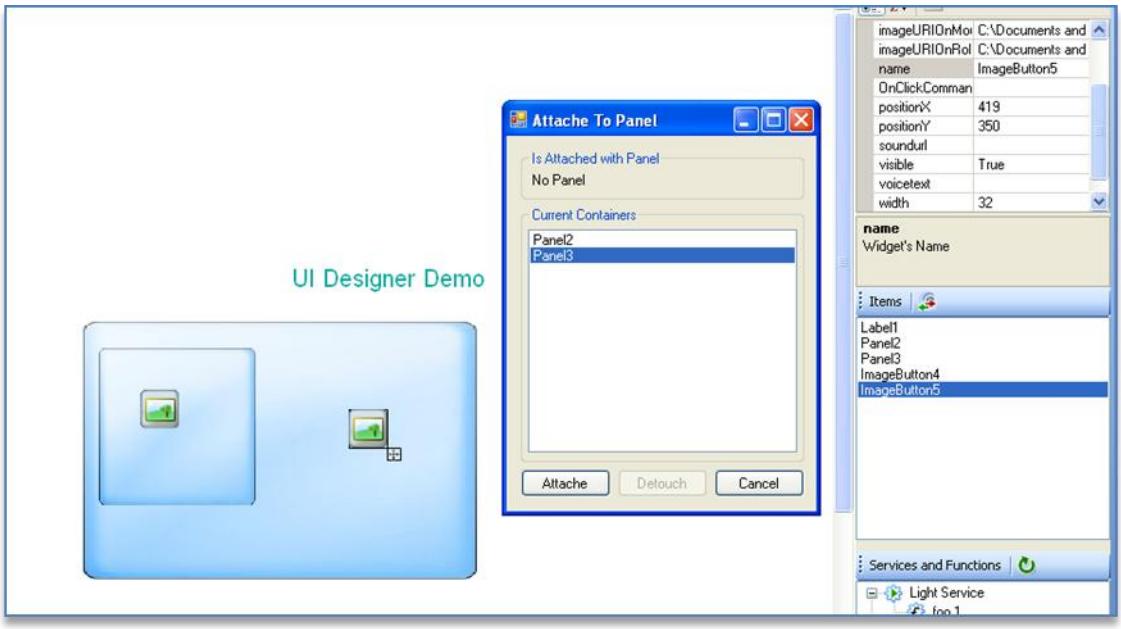


Figure 27. Attach item to panel

In case a selected widget placed within a container, a button appears over the list of current items. By clicking on this button, a new window appears that presents a list of all containers on the back of the selected item. By selecting the desired container and by clicking on the "attach" button, the selected item is grouped with the container (see Figure 27). When an item is attached to a container, by moving the container on the stage the attached item is also moved. In case the selected item is already attached to some container, another button is activated in the new window, the "detach" button, which can be used to ungroup the item and the container (see Figure 28).

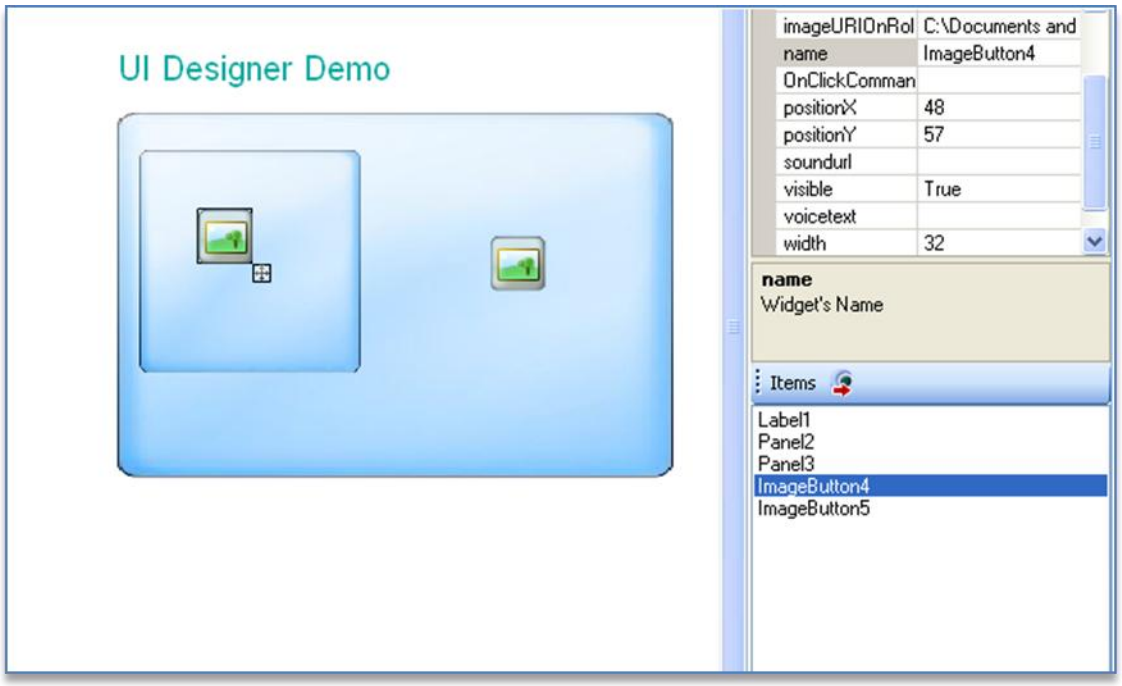


Figure 28. Detach item from panel

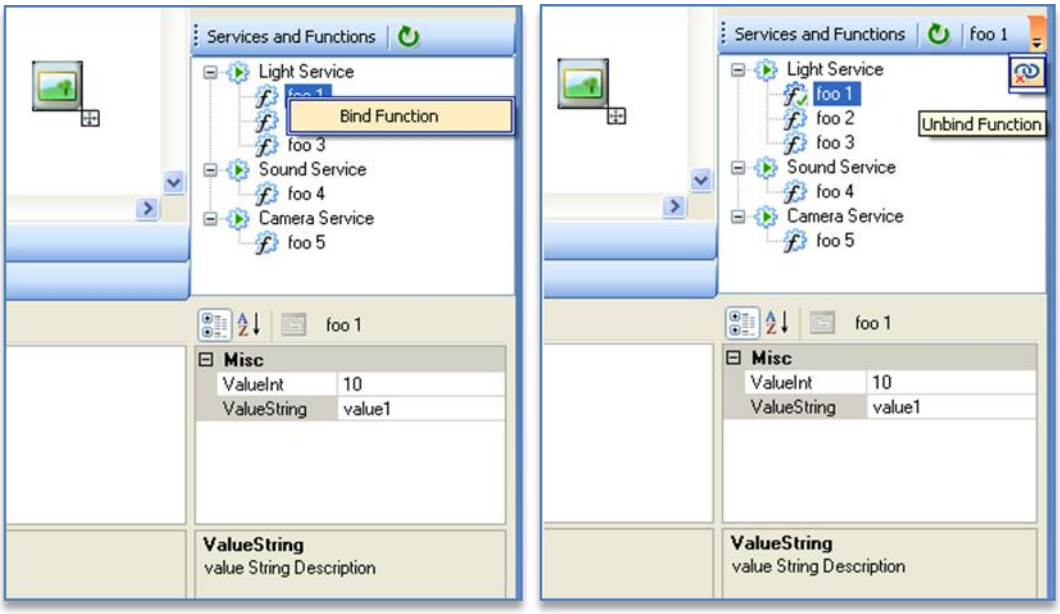


Figure 29. (a) Bind and (b) unbind a service's function with / from an action widget

An action widget can also be bound to a function of a service. On the bottom-right part of the interface, a list containing the services and functions of the selected room is displayed. Below the list, the arguments of the selected function appear. A user can bind a function to an action widget in a way similar to binding interface items. By right-clicking on a function, a bind menu appears. When this menu is selected, the currently selected function appears ticked (see Figure 29a). When an action widget is bound and the respective item is selected, the name of the function to which it is bound appears, along with a button that allows the user to unbind the function (see Figure 29b).

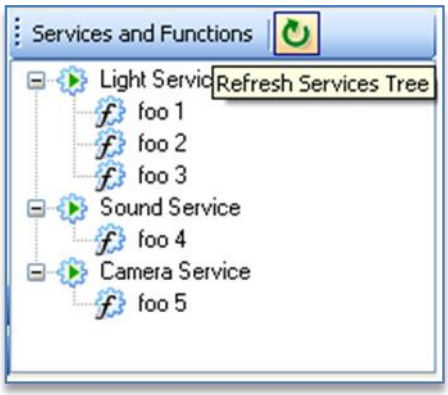


Figure 30. Refresh current room's services and their functions

At any time the services within a room may change. For this reason, a refresh button is available above the services list that activates service discovery and updates the list accordingly.

3.1.5. Signals

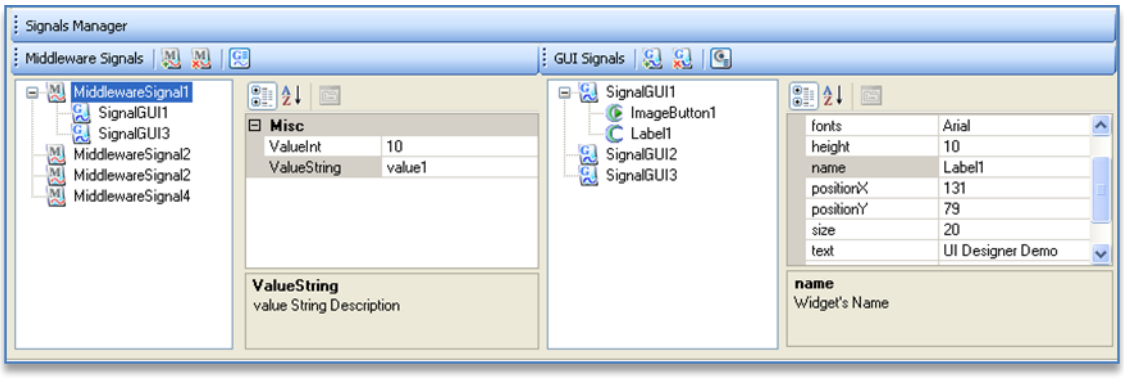


Figure 31. Signals Overview

The smart environment should be able to interact with the generated interface to inform it about changes occurring in the environment. For this reason, two types of signals are implemented: Middleware Signals and GUI Signals. Middleware signals are transmitted from the middleware with specific arguments to produce GUI Signals. Middleware is the means of communication between the ambient environment and the applications. When GUI signals are produced, they modify properties of items within the current stage. The items that are affected by each signal, as well as the new properties that are obtained from them, are presented at the bottom part of the interface (see Figure 31). A tree view shows the hierarchy of signals and items, and a property grid is available where there user can modify the property values of items (see Figure 32).

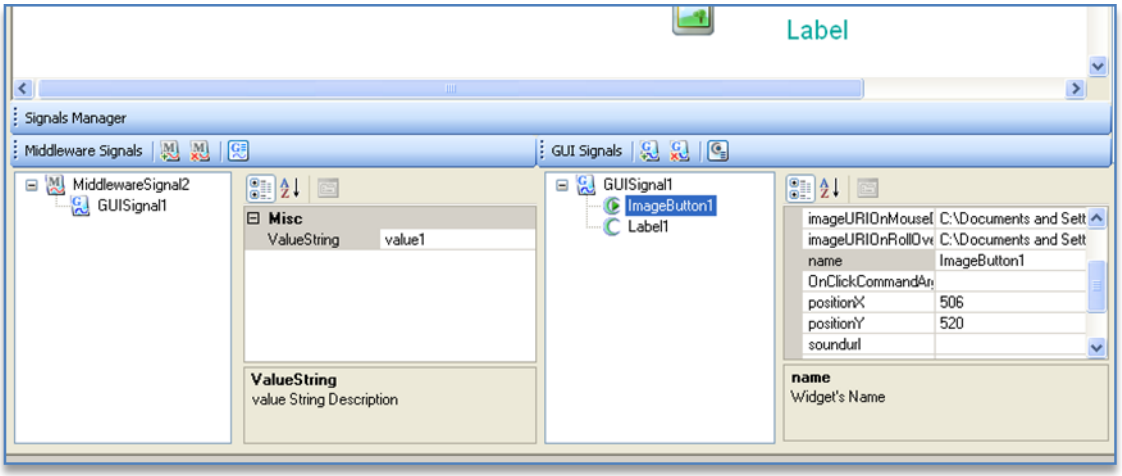


Figure 32. Signals' Example

Middleware Signals

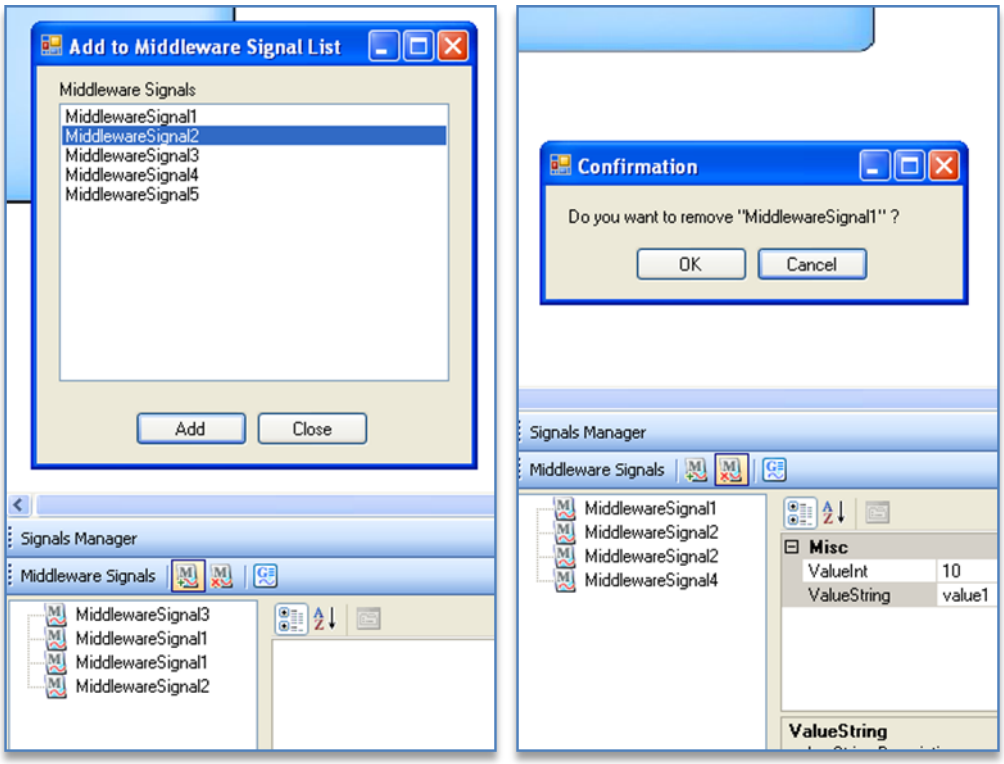


Figure 33. (a) Add and (b) remove Middleware Signals

Above the middleware signals list there are three buttons. The first one opens a new window that presents all available middleware signals that exist in the environment. The way middleware signals are discovered in the environment will be explained in Chapter 4. The user can select the desired signal and has the ability to add it multiple times to the list using the “add” button (see Figure 33a). This is justified by the fact that a user may need to distinct signals according to their arguments. The user can also remove any signal from the list after confirming the selection (see Figure 33b).

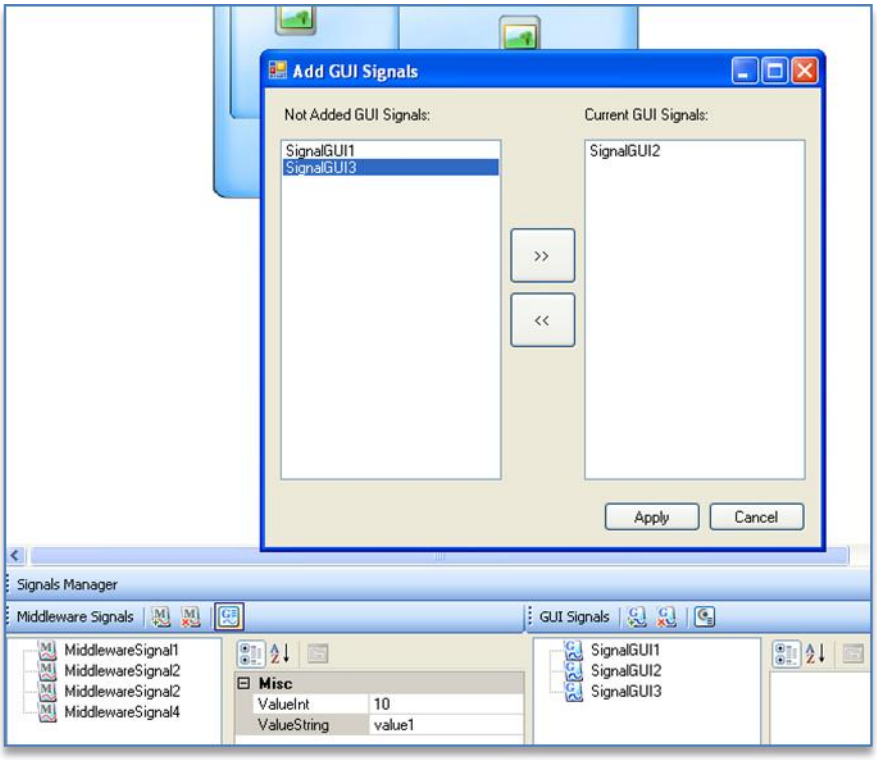


Figure 34. Change Middleware signal's GUI Signals

The third button is used to associate GUI signals to the selected middleware signal, so that when a middleware signal is produced with the specific arguments the respective GUI signals are produced in the interface (see Figure 34).

GUI Signals

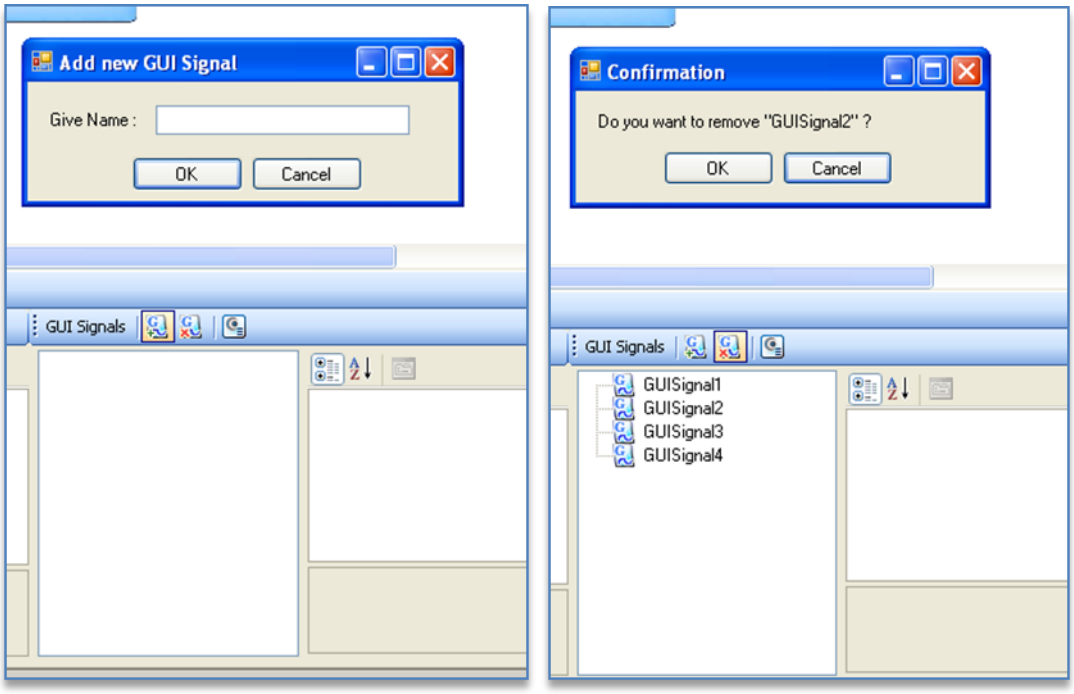


Figure 35. (a) Add and (b) remove GUI Signals

At the bottom of the interface there is an area used for GUI signals. There are three buttons available, used for insertion and deletion of GUI signals (see Figure 35), as well as the modification of items that correspond to each GUI signal (see Figure 36).

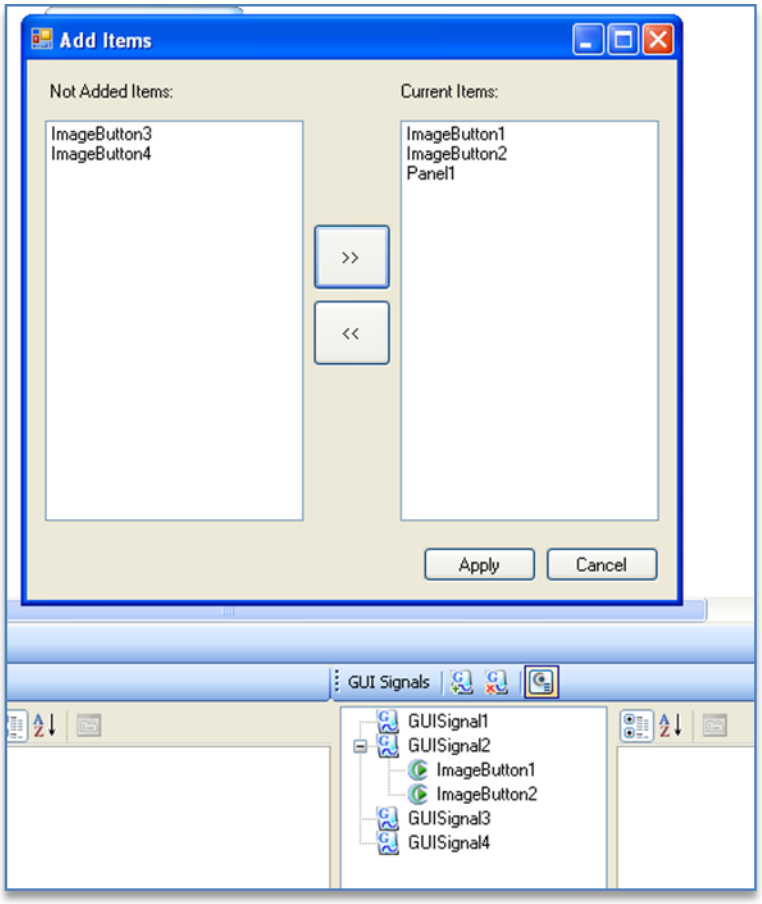


Figure 36. Change GUI signal's items

3.1.6. Scanning Orders

Scanning is an interaction technique that enables motor impaired users to have access to interactive applications and services [16]. The basic idea is that a special "marker" (e.g., a colored frame) indicates the interaction item that has the input focus. The user can shift the focus marker to the next / previous interaction object using any kind of switches (e.g., keyboard keys, special switch hardware, remote control). When the focus is over an object the user wants to interact with, another switch is used to perform "selection". Additionally, in cases where the user can use just a single switch, focus movement can be automatically generated by the system at constant time intervals. This variation of the technique is referred to as "automatic scanning", while any other case is generically called "manual scanning" (even if the hands are not actually used at all). Besides these features, on each item focus or selection a sound is produced or the item's name is read-out loud. This way the user interfaces can be used by visually impaired users in addition to motor impaired ones. A more extensive description of scanning will be presented in Chapter 5, reporting a case study of user interface development using the AmI Designer.

Figure 37. Scanning Order Overview

The order according to which elements in the interface will be scanned at run time is defined by the scanning order list that can be found at the left bottom part of the interface (see Figure 37). The only items that are allowed to have children in the scanning hierarchy are the containers.

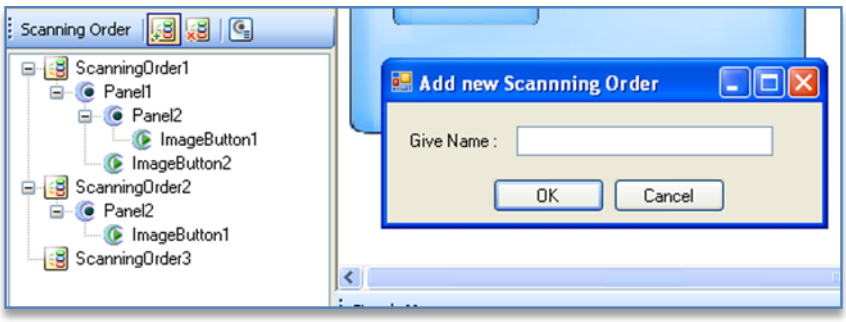


Figure 38. Create new scanning order

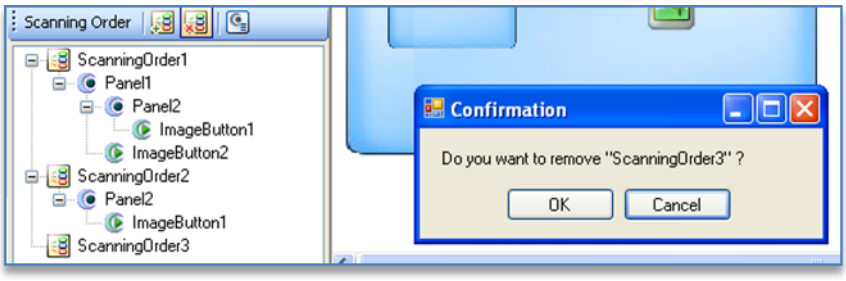


Figure 39. Delete a specific scanning order

Three buttons are available. The first one is used to create a new scanning order (see Figure 38), the second one is used to delete the selected scanning order (see Figure 39), while the third is used to insert items into a scanning order (see Figure 40).

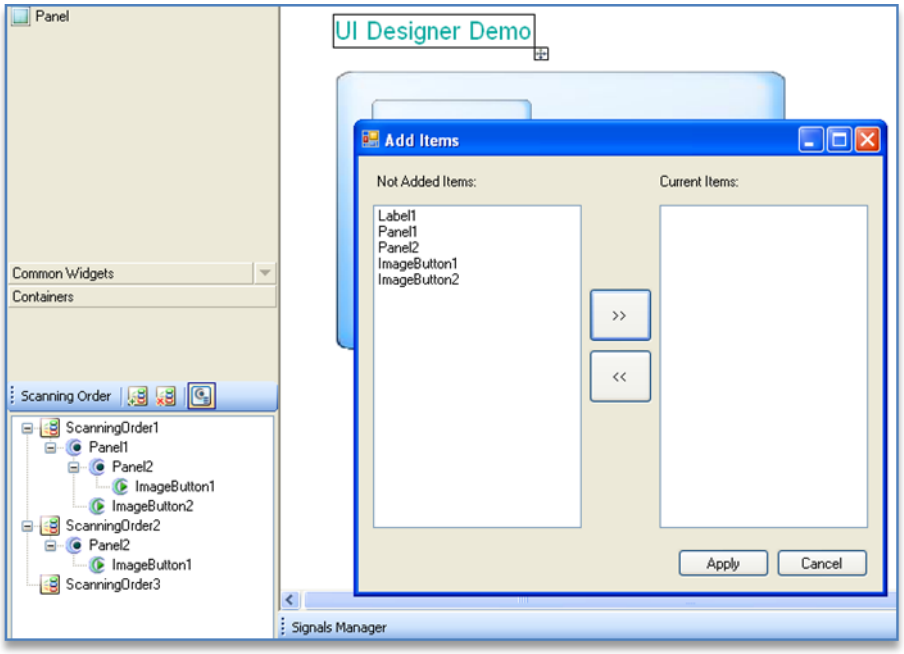


Figure 40. Change Scanning order's items

3.2 AmI Player

The AmI Player has no specific interface. It is simply an application that parses the XML file that is generated by the AmI Designer. In this XML, file the AmI Designer describes in detail the interface that will be produced along with the signals and scanning orders. The XML file that will be loaded by the AmI Player depends on the configuration file of the AmI Player.

```
<?xml version="1.0" encoding="utf-8" ?>
<ServicesUI>
  <MakeWidgets>
    ...
  </MakeWidgets>
  <Signals>
    ...
  </Signals>
  <MiddleWareSignals>
   ...
  </MiddleWareSignals>
  <Scanning>
   ...
  </Scanning>
</ServicesUI>
```

Figure 41. XML Main Parts

The basic parts of the XML file (see Figure 41) are: MakeWidgets, Signals, MiddleWareSignals and Scanning.

```
<MakeWidgets>
  <Widget isContainer="true">
    <properties>
      <type>...</type>
      <name>...</name>
      ...
    </properties>
    <containingWidgets>
      <Widget isContainer="false">
        <properties>
          <type>...</type>
          <name>...</name>
          <commandOnClick type="backend">CallService</commandOnClick>
          <commandArguments>Service's Arguments</commandArguments>
          <!--
          <commandOnClick type="gui">ShowPanel</commandOnClick>
          <commandArguments>Name of Widget to change its visibility</commandArguments>
          -->
        </properties>
      </Widget>
      <Widget isContainer="false">
        <properties>
          <type>...</type>
          <name>...</name>
          ...
        </properties>
      </Widget>
    </containingWidgets>
  </Widget>
</MakeWidgets>
```

Figure 42. Make Widgets

The MakeWidgets part (see Figure 42) is responsible for describing the items of the user interface. Any widget used in the interface is enclosed in Widget tags, whose sole attribute is isContainer that accepts true or false values when the widget is a container or not respectively. Within the Widget tags there are the properties tags that describe the properties of the respective widget. The obligatory tags within properties are type, name, height, width, positionX and positionY. In the case of an empty widget only the properties are needed. In the case of action widgets, there are two important tags, commandOnClick and commandArguments. When the “type” attribute of commandOnClick is “gui”, the widget is responsible for modifying some interface element, whose name is given in commandArguments. In case the “type” attribute of commandOnClick is “backend”, the widget is responsible for calling an external service, more specifically the function with the arguments as they are described in commandArguments. In case a container widget is described, there is a containingWidgets tag within which all widgets that belong to the specific container are described. Nesting of containers within other containers is also allowed.

In the example of Figure 43, a container widget of panel type is created that contains an image button and a label. Note that the X and Y coordinates of the child items of a container use the upper left corner of the container as a (0, 0) starting point.

```
<?xml version="1.0" encoding="utf-8" ?>
<ServicesUI>
   <MakeWidgets>
     <Widget isContainer="true">
       <properties>
         <type>widgetPanel</type>
         <name>Panel11</name>
         <imageURI>\\UIPlayerFlash\\images\\ColorPanel.png</imageURI>
         <positionX>950</positionX>
         <positionY>0</positionY>
         <width>74</width>
         <height>495</height>
         <visible>true</visible>
         <lock>true</lock>
         <soundurl></soundurl>
         <voicetext></voicetext>
       </properties>
       <containingWidgets>
         <Widget isContainer="false">
           <properties>
             <type>widgetImageButton</type>
             <name>ImageButton7</name>
             <imageURI>sel spot colour_white.png</imageURI>
             <imageURIOnMouseDown>sel_spot_colour_white.png</imageURIOnMouseDown>
             <imageURIOnRollOver>sel_spot_colour_white.png</imageURIOnRollOver>
             <positionX>12</positionX>
             <positionY>16</positionY>
             <width>57</width>
             <height>57</height>
             <commandOnClick type="backend">CallService</commandOnClick>
             <commandArguments>All White</commandArguments>
           </properties>
         </Widget>
         <Widget isContainer="false">
           <properties>
             <type>widgetLabel</type>
             <name>Label1</name>
             <text>Generate Lights</text>
             <color>0x0AAAA0</color>
             <fonts>Arial</fonts>
             <size>20</size>
             <positionX>20</positionX>
             <positionY>20</positionY>
             <width>200</width>
             <height>20</height>
           </properties>
         </Widget>
       </containingWidgets>
     </Widget>
```

Figure 43. Make Widget Example: (Panel, ImageButton and Label)

```
<Signals>
  <Signal>
    <SignalName>GUISignalName</SignalName>
    <ChangeWidgets>
      <Widget isContainer="...">
        <properties>
          <type>ExistingItemType</type>
          <name>ExistingItemName</name>
          ...
        </properties>
      </Widget>
    </ChangeWidgets>
  </Signal>
  <Signal>
    ...
  </Signal>
  ...
</Signals>
```

Figure 44. Make GUI Signals

Signals tags are used to describe GUI signals (see Figure 44). Each GUI signal is described by a Signal tag that contains a SignalName tag (the name of the signal) and a ChangeWidgets tag (containing existing widgets with their new properties). These widgets are described as presented above.

```
<MiddleWareSignals>
  <MiddleWareSignal>
    <GuiSignalName>ExistingGUISignalName</GuiSignalName>
    <MiddlewareSignalName>MiddlewareSignalName</MiddlewareSignalName>
    <Arguments>
      <Argument type="range">
        <int>3</int>
        <int>5</int>
      </Argument>
      <Argument type="simple">
        <int>3</int>
      </Argument>
      <Argument type="simple">
        <string>stringValue</string>
      </Argument>
      ...
    </Arguments>
  </MiddleWareSignal>
</MiddleWareSignals>
```

Figure 45. Make Middleware Signals

MiddleWareSignals tags are used to describe middleware signals (see Figure 45). Each one consists of the tags GuiSignalName (the GUI signal the middleware communicates with), MiddlewareSignalName (the name of the middleware signal) and Arguments (the arguments of the middleware signal with specific values). The argument values may be either simple values or ranges of values. In case an argument is simple, it is described by an attribute "type" equal to "simple". Otherwise, the "type" attribute has a "range" value. When a

middleware signal is received using the specified arguments, a GUI signal will be produced with the specified name.

```
<Scanning>
  <initialScanningItem>ExistingItemName</initialScanningItem>
  <ScanningType>ScanningType</ScanningType>
  <ScanningOrder>
    <Item isContainer="false">
      <name>ExistingItemName</name>
    </Item>
    <Item isContainer="true">
      <name>ExistingItemName</name>
      <children>
        <Item isContainer="false">
          <name>ExistingItemName</name>
        </Item>
      </children>
      ...
    </Item>
    ...
  </ScanningOrder>
</Scanning>
```

Figure 46. Scanning Order Schema

Finally, the scanning part is described using Scanning tags (see Figure 46). The first thing that should be described is the item that will commence the scanning, using the initialScanningItem tags. The scanning type is described using ScanningType tags (manual or automatic). The scanning order of the items is described next, in ScanningOrder tags. If an item is a container, the attribute “isContainer” is equal to “true”, otherwise “false”. If an item is a container, besides the name tag, it also contains children tags that describe the scanning order of the container’s children.

4. Implementation

This chapter describes the implementation of AmI Designer and AmI Player, following a short overview of the ICS-FORTH AmI architecture in order to illustrate how the two developed tools interact with the overall environment. The AmI Environment includes many devices, such as PDAs, projectors, monitors, computers, cameras and sensors which should communicate with each other. The Middleware is the AmI environment's layer that is responsible for this communication. The devices that will be activated depend on external measurements which are indicated from the sensors: these measurements are processed by the Decision Engine which is considered as the "Brain" of an AmI Environment and defines the state of the Environment. In order for a specific device to be used, a separate application needs to be built. As already discussed the objective of the developed tools is to provide a way to design and create applications which will be able to manage more than one device. The architecture described above is shown in Figure 47.

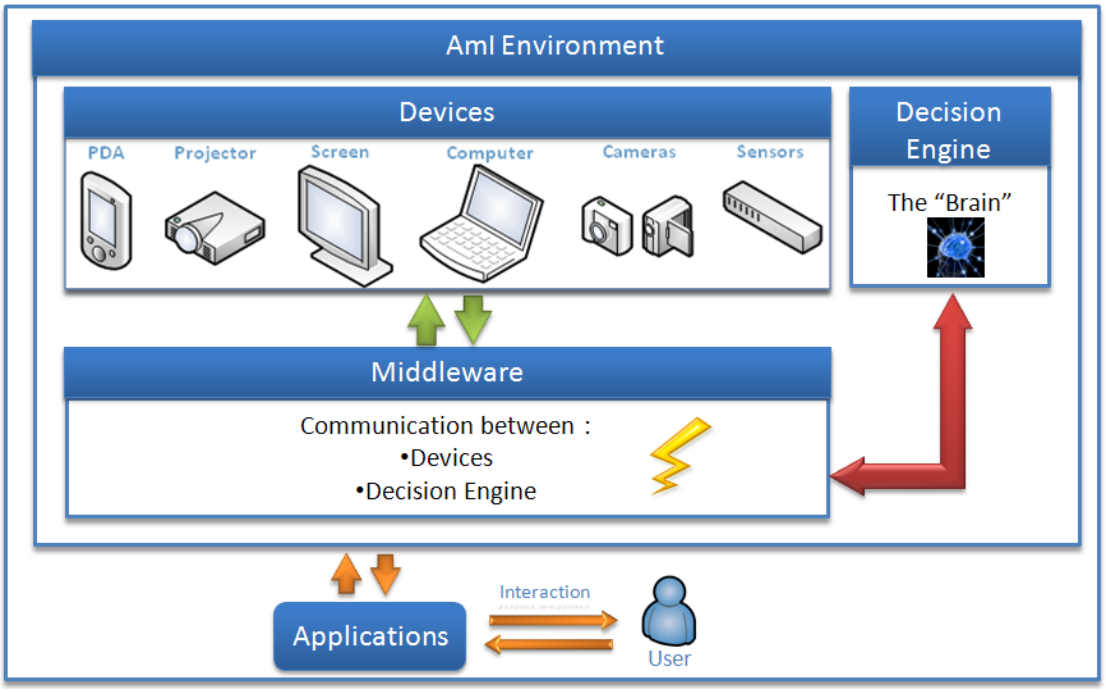


Figure 47. ICS-FORTH AmI Architecture

For such an application to be built, the following process needs to be carried out. First of all, the design of the GUI has to be performed using the AmI Designer, which exports the GUI description to an XML file. Afterwards, the AmI Player uses these XML files in order to generate GUIs that control the AmI Environment (see Figure 48), and the user can interact with them.

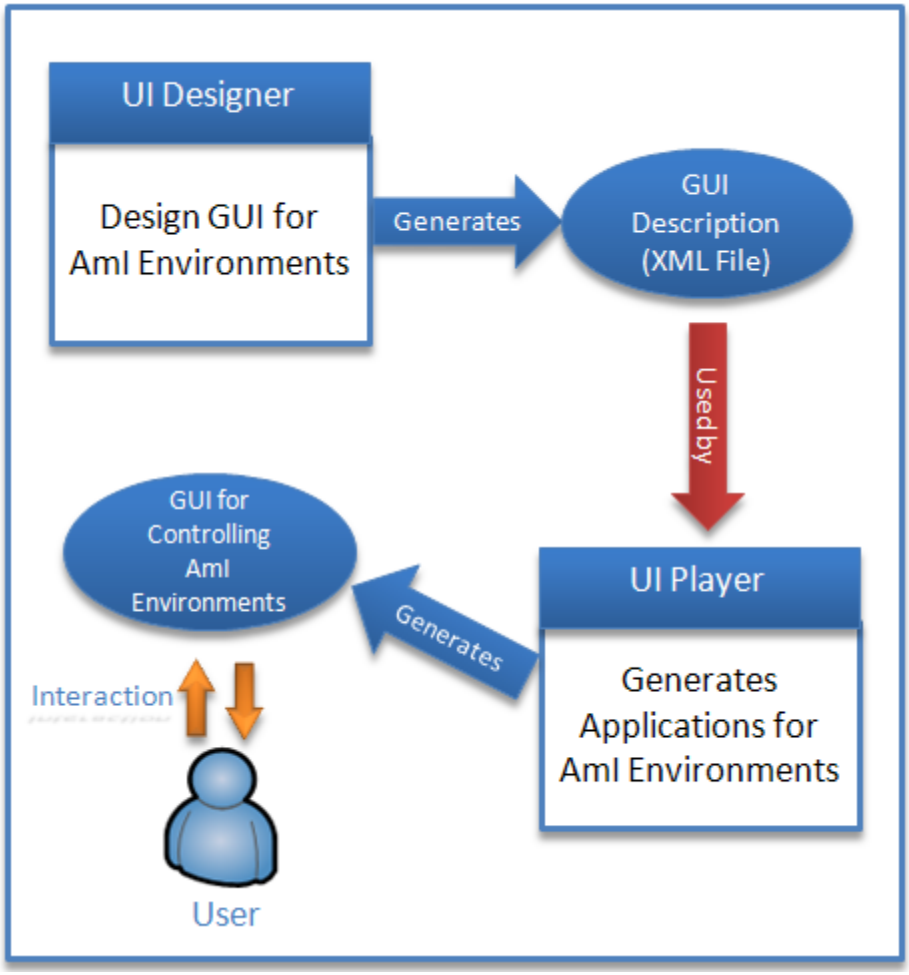


Figure 48. System's Flow

4.1 AmI Designer

The AmI Designer is an application that communicates with the Middleware so as to be informed about what happens in the environment. The AmI Designer has been implemented as consisting of two separate parts: (1) A C# application, which communicates directly to the Middleware and (2) a Flash stage, which is responsible for the GUI design. For the creation of the system, communication between the C# application and flash stage is essential (see Figure 49). The way this communication takes place is described in further details later on in this section.

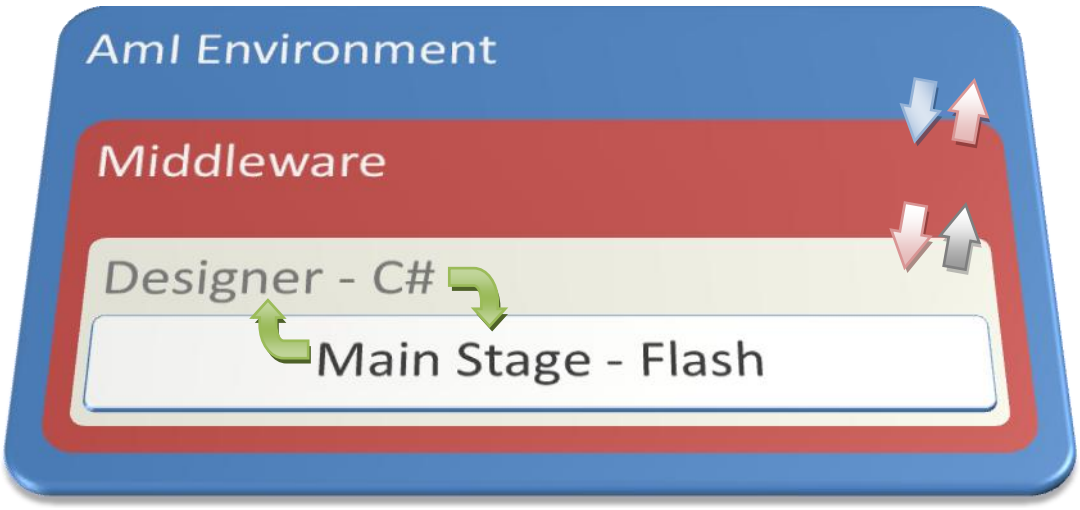


Figure 49. Communication Overview: Environment ↔ Middleware ↔ Designer (GUI ↔ Stage)

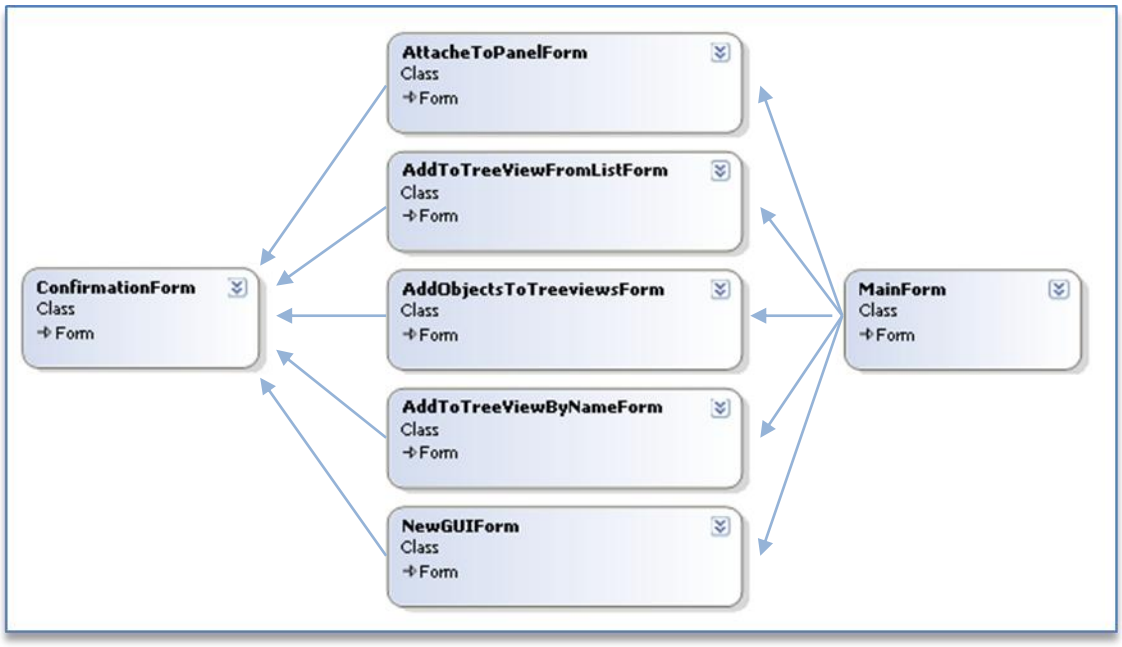


Figure 50. Designer's Forms

In the AmI Designer's GUI implementation the forms in Figure 50 are used. The "MainForm" is the core form of the application which calls the other sub-forms (e.g., AttacheToPanelForm) if needed, as shown above. In order to return to the "MainForm" from a sub-form, the "ConfirmationForm" is used.

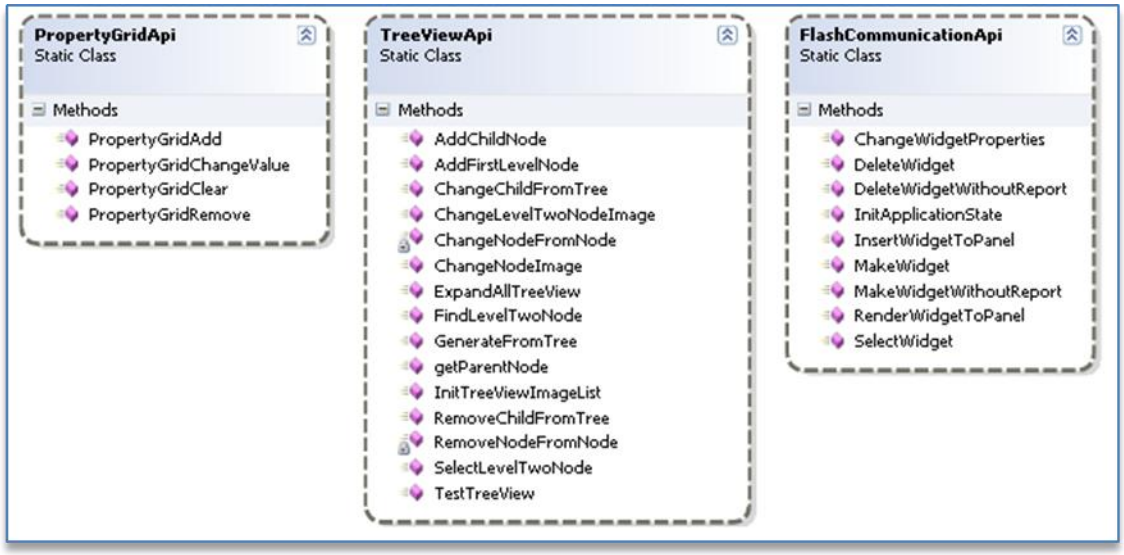


Figure 51. Utility Classes

The project uses three utility classes (see Figure 51): (1). “PropertyGridApi” is used for managing the dynamic properties of the widgets. These properties are described by the flash stage as mentioned previously; (2) “TreeViewApi” is used in order to carry out tree view operations, such as add, remove and find; (3) “FlashCommunicationApi” is used in order for the C# Application to communicate with the flash stage.

Four hierarchies are used in order to retain the produced GUI’s description information.

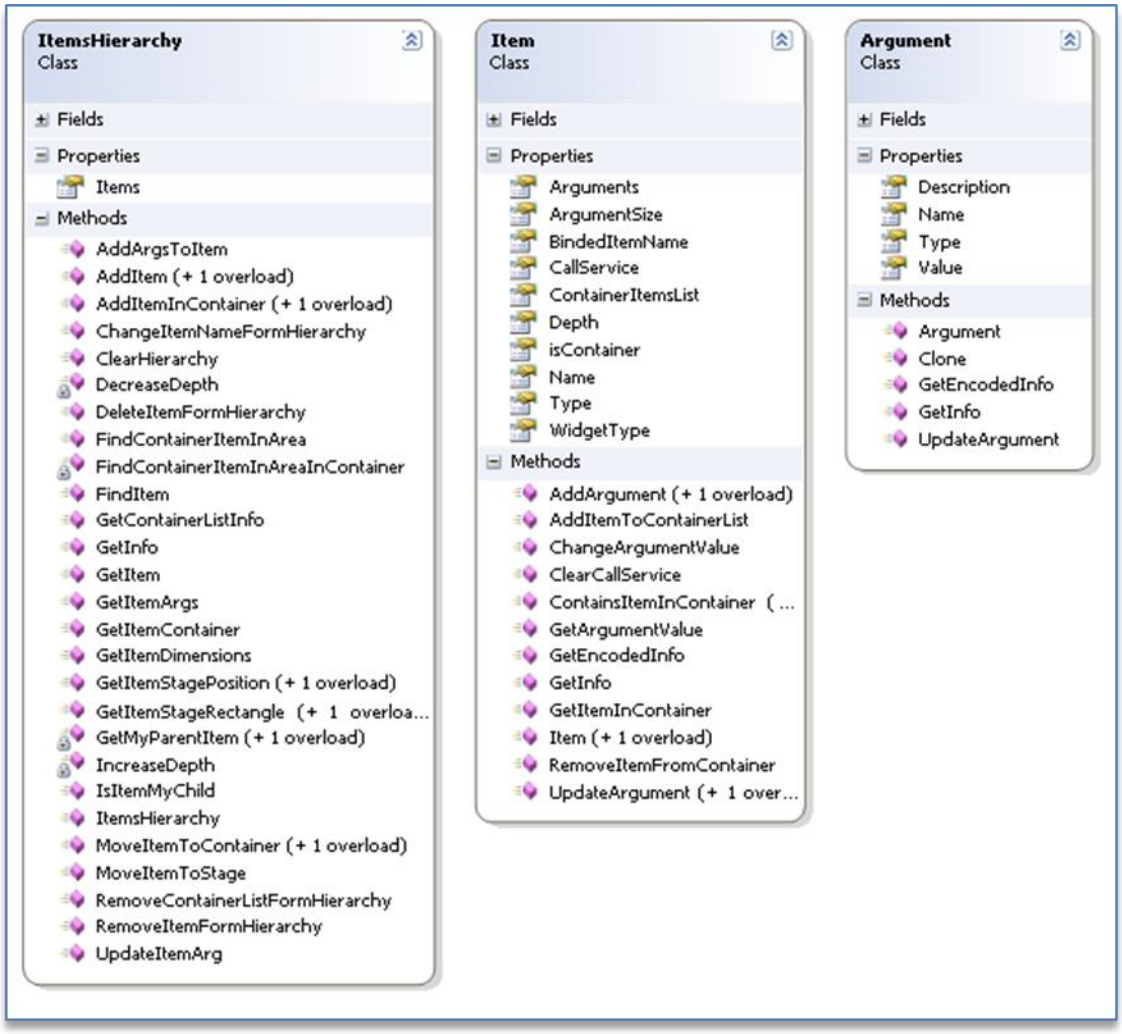


Figure 52. Item Hierarchy

The first hierarchy ("ItemsHierarchy") describes the current items of the flash stage (see Figure 52). Each item consists of specific arguments, which are dynamically updated by the flash stage. In case an item is a "WidgetContainer", then it contains a list of contained items. Thus, all the finding procedures of the hierarchy are carried out recursively.

Figure 53. Middleware Signal Hierarchy

The second basic hierarchy is the "MiddlewareSignalsHierarchy" which is updated by the Middleware (see Figure 53). It contains information and the analytic description (Middleware Signal's arguments) of Middleware Signals of the AmI Environment.

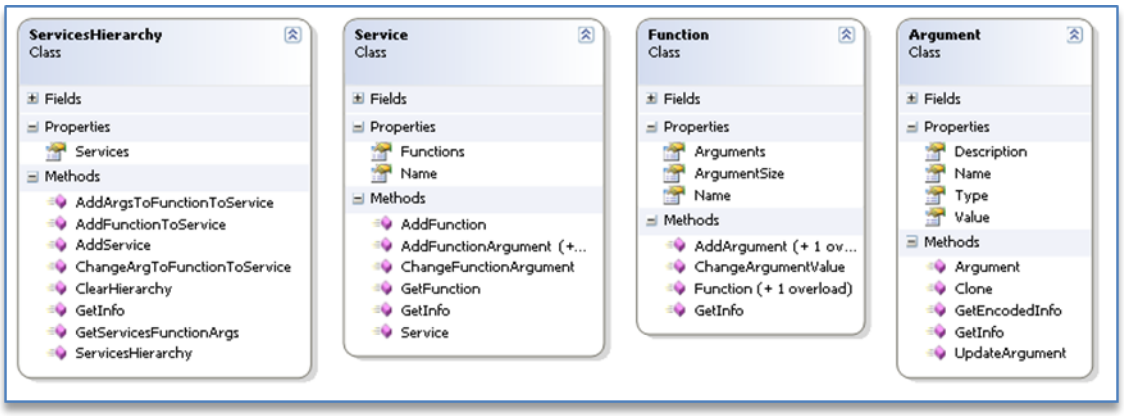


Figure 54. Services and Functions Hierarchy

The "ServicesHierarchy" is used for describing in detail the Services that exist in a specific room of the AmI Environment (see Figure 54). Each service consists of different functions which can be called from the GUI according to specific arguments.

Figure 55. GUI Signals Hierarchy

The last hierarchy, "GUISignalsHierarchy", is used for retaining the information concerning the GUI signals that exist in a specific GUI (see Figure 55). Each GUI Signal connects with a Middleware Signal as mentioned above.

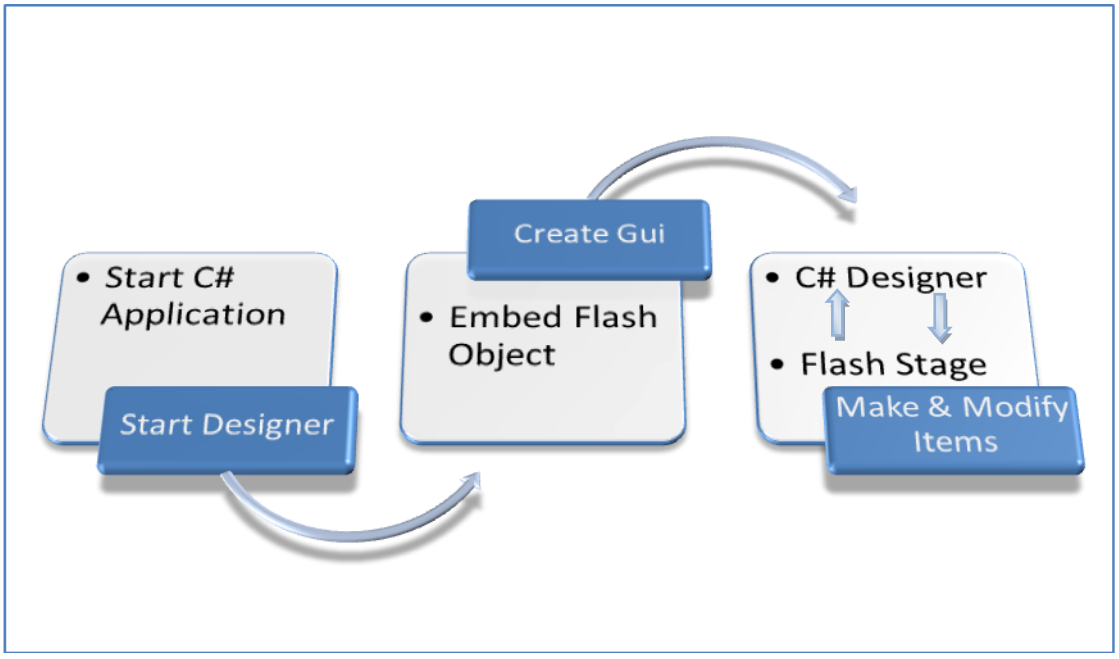


Figure 56. AmI Designer Flow

The steps that are carried out in order for a GUI to be created are as follows (see Figure 56). Firstly, the AmI Designer application is started. The next step is to create a flash stage. This is a simple flash object which is embedded in the C# application. Lastly, the creation or modification of an item causes message exchange between the C# application and the flash stage that was previously described. These messages are depicted in Figure 57.

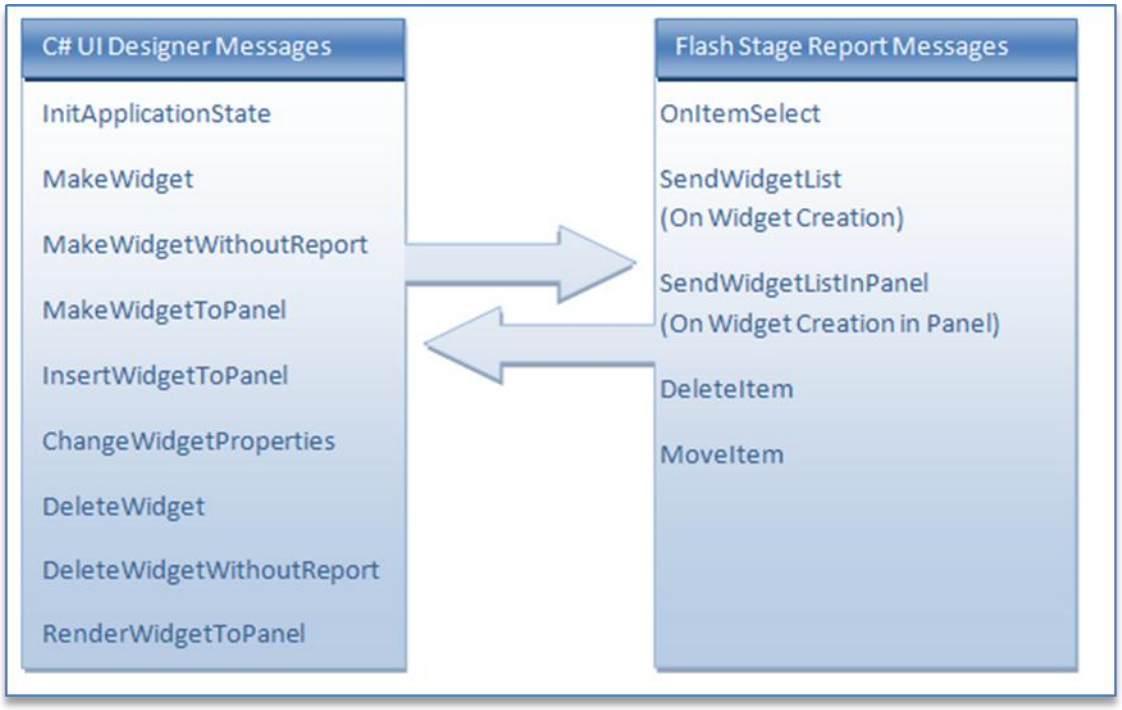


Figure 57. C# Designer and Flash Communication

The messages of the C# application (C# AmI Designer messages) create requests to the flash stage, while the messages of the flash stage to the C# application (Flash Stage Report Messages) acknowledge the received request.

4.2 AmI Player

AmI Player is an application that communicates with the Middleware so as to activate devices and receive signal from them. The AmI Player consists of two separate parts: (1) a C# application, which communicates directly to the Middleware, and (2) a Flash stage, which is responsible for the GUI generation. Communication between the C# application and the flash stage is essential (see Figure 58). This communication takes place through calls of Middleware functions with specific arguments.

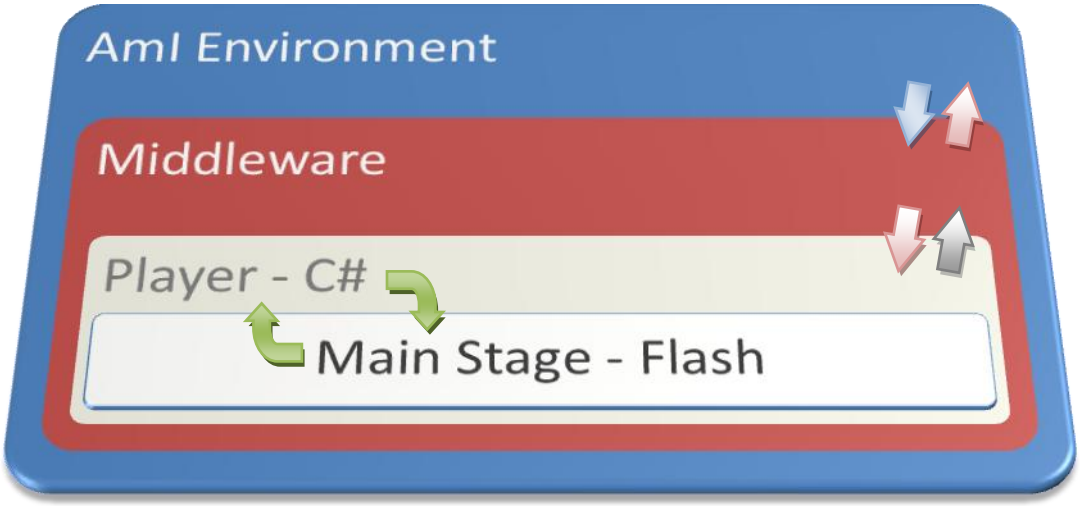


Figure 58. Communication Overview: Environment ↔ Middleware ↔ Player (GUI ↔ Stage)

The steps that are carried out in order for a GUI to be generated and for the user to be able to use the devices are the following (see Figure 59). Firstly, the configuration file of the AmI Player is modified to define the XML file that the AmI Player will use. This XML file is the output of the AmI Designer. The next step is to start the AmI Player application. This is a simple flash object which is embedded in the C# application. Lastly, the GUI is generated and the user can interact with it, controlling by this way the smart room.

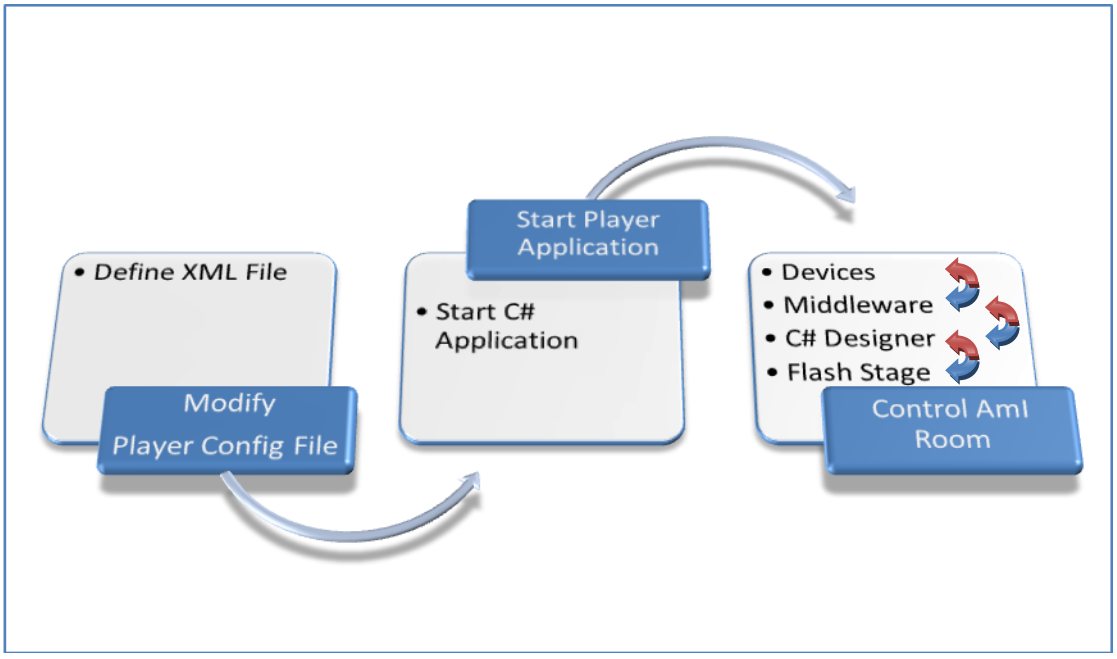


Figure 59. AmI Player Flow

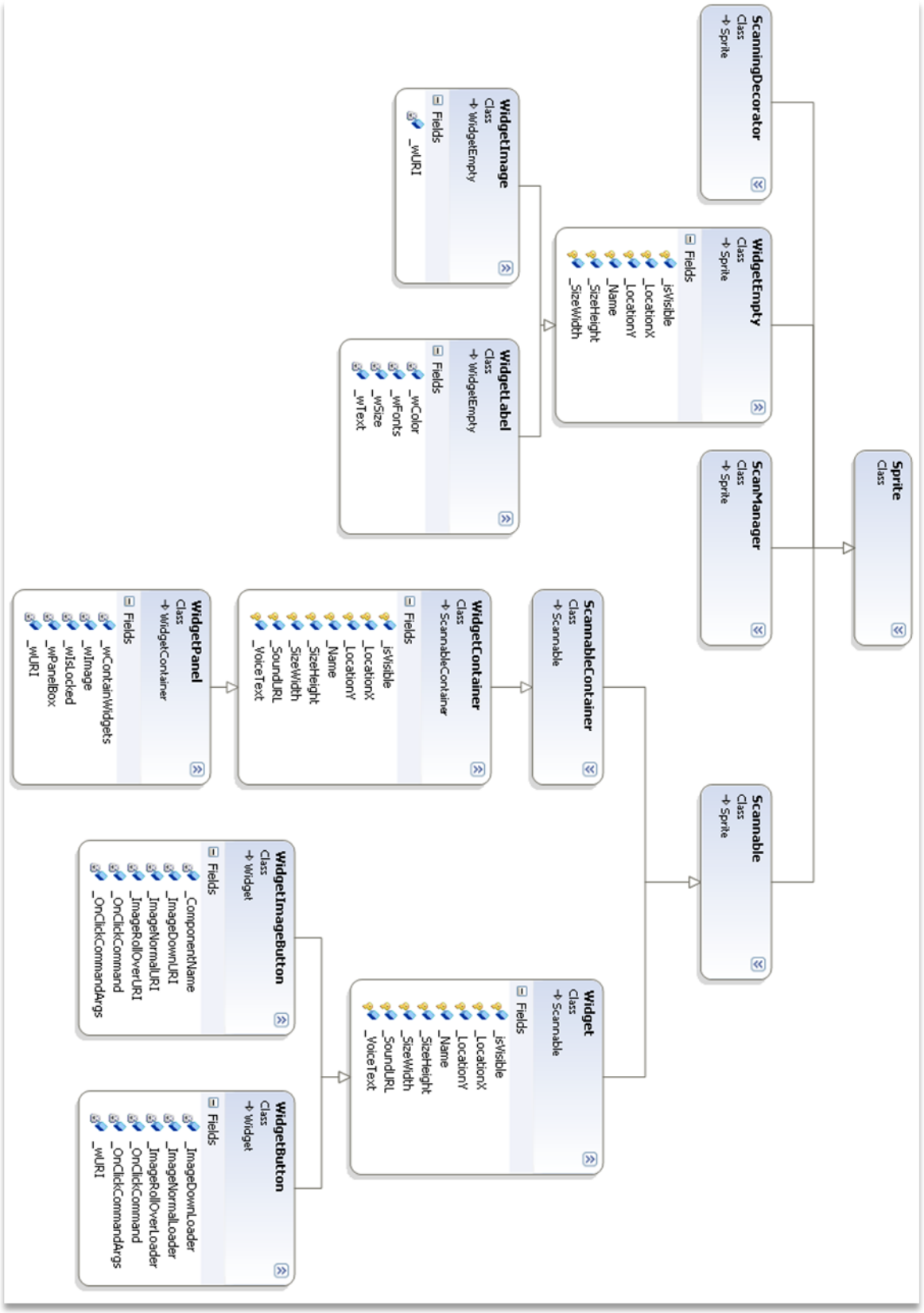


Figure 60.Flash Stage Architecture

Figure 60 depicts the basic architecture of the Widgets and the Scanning Mechanism. There are three different kinds of Widgets: (1) “WidgetEmpty” are items that users cannot interact

with, but appear in the user interface for decorative or informative purposes only (such widgets are labels and images); (2) “Widget” are items that the users can interact with to call services using specific arguments or alter elements in the user interface (buttons and image buttons); (3) “WidgetContainer” are items are used to group other items. Items can be attached or detached from a container at any time. If someone wishes to implement a widget, the corresponding class needs to be extended (“WidgetEmpty”, “Widget” and “WidgetContainer”). The scanning mechanism will be described in details in Chapter 5.

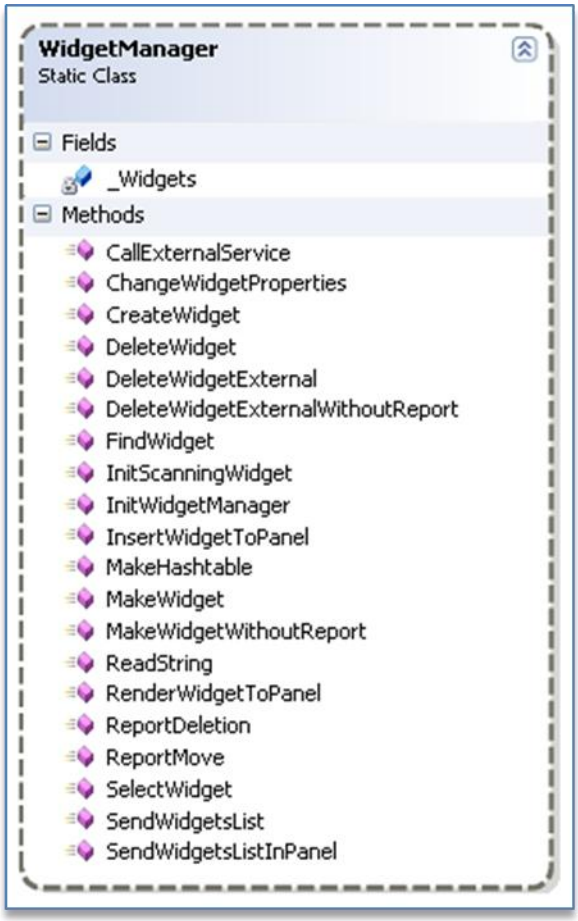


Figure 61. Flash Widget Manager Functionality

The basic class of the flash stage is the “WidgetManager”, which is responsible for creating, deleting and modifying the stage’s items (see Figure 61). Apart from the stage’s operations, it implements the communication of the flash stage and the C# Application. This class is common both to the flash stage of the AmI Designer and the AmI Player. The “WidgetManager” receives messages from the C# Application and either responds to it or modifies the GUI. The only information that is retained inside the “WidgetManager” consists of the current items of the flash stage and their properties. Lastly, the “WidgetManager” is responsible for notifying the AmI Designer about the available widgets to be displayed in the Widget Toolbox.

5. Developing user interfaces in Smart Homes: a case study

Currently, ICS-FORTH has set up an AmI laboratory, called “AmI Sandbox”, which is based on the above described architecture and integrate a number of AmI devices. Services to use such devices are currently being elaborated. In this context, a service for light management name CAMILLE has been developed using the AmI Designer and the AmI Player and has been evaluated through a user-study. Future applications of the developed tools are envisaged to move from a laboratory environment to full scale Smart Homes. In such a scenario, a specialized AmI architect/designer will be able to create accessible interface for controlling each home according to the needs, requirements and preferences of each inhabitant.

In the case study, the AmI Designer was used for the design of the application, and the AmI Player was used to generate the GUI to provide interaction with the lights. The steps that are carried out were the following. First, of all the basic parts of the GUI were created. Afterwards, the scanning order of the stage’s widgets was setup. Then the widgets that call the services that control lights were determined. Lastly, it was determined which widgets will change based on specific Middleware signals. At the end of this process, the CAMILE application was created. This Chapter describes the design decisions taken and their rationale, and presents the user interface that was developed. Finally, it discusses the role of the AmI Designer and the AmI Player in the creation for the user interface, summarizing the accumulated experience.

5.1 CAMILE (Controlling AmI Lights Easily)[59]

Ligh control is a key task of everyday home life. Rooms are usually equipped with several light sources that, depending on the current context (e.g., night / day, working / reading / watching TV, single person / group) require to be set at different configurations. Furthermore, each light source may have different control parameters / capabilities than the others, e.g., the possibility of dimming, or changing its color. This task usually involves physical movement in order to access the control of each source, and presents the challenge of locating and mapping the related switches / controls. It also requires knowing the current state of the lights, which may not be readily available to a blind person, or even to someone residing in a different room. A well-known, typical, related usability problem is that of natural mapping [15]; i.e., the case where one has to identify the distinct function of a row of identical switches through a frustrating trial and error process.

CAMILE is an interactive application for intuitively controlling multiple sources of light in smart home environments, built so that it can be used by anyone, the young, the elderly, people with visual disabilities, and people with hand-motor disabilities alike. In order to accommodate such a wide range of users, three different modes of interaction were developed: (a) touch-screen-based for sighted users with no motor impairments; (b) remote controlled operation in combination with speech for visually impaired users or tele-operation by sighted users; and (c) scanning (both manual and automated) for motor-impaired users.

CAMILE has been installed in an AmI laboratory space (called the "AmI Sandbox") comprising three different rooms in the basement of the main building of ICS-FORTH.

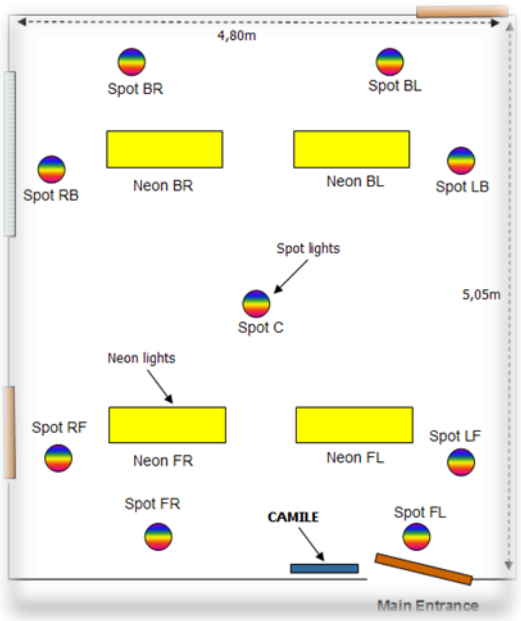


Figure 62. Layout of the computer-controlled lights on the ceiling in the AmI Sandbox

Among the technologies installed in this space there are several computer controlled lights. More specifically, on the main room's ceiling there are four neon lights that can be dimmed and nine LED spotlights that can produce colored light of any RGB value. A sketch of the lights' layout in the AmI Sandbox is illustrated in Figure 62. As can be seen in this Figure, each light has a distinct name, e.g., Spot FR, where F stands for front, B for Back, L for left and R for right. By convention, the direction that the user is facing at when using CAMILE; i.e., towards the room's main entrance is considered as "front".

Figure 63. The lighting control console originally used for in the AmI Sandbox (Scan 812 DMX by Work)

All the lights can be controlled using the DMX communications protocol, which is most commonly used to control stage lighting and effects. Using this protocol, lights are typically controlled through a dedicated lighting control console (also called lighting board, or lighting desk). In Figure 63. the console initially used for the purposes of light control is illustrated (Scan 812 DMX by Work). As can be seen, its main interface comprises a number of identical sliders and push down buttons. While using this device for controlling the lights of the AmI Sandbox, several usability problems have been spotted:

- There is no intuitive or even learnable mapping among the controls and the lights.

- Lights are basically controlled through the sliders, but not all lights are controlled in the same way, since:
  - Each neon light is controlled through just a single slider. The slider's position denotes the light's intensity.
  - Each LED spot light is controlled through three sliders. Each slider's position denotes an RGB value. In this case, it is very difficult to set the light at a specific color (except of course red, green and blue).
- Since the number of sliders is less than the number of the required light controls, extra buttons are used to set different modes. In each mode the sliders control a different set of lights. This fact undermines any attempt to meaningfully label the sliders.

Overall, the experience acquired with the lights console has shown that this interface is totally inappropriate for use by "anyone" in a casual everyday environment, e.g., in a home. In fact, it can even compromise the safety of its users, since in emergency situations it does not allow for quick, faultless actions. And of course, it is totally inaccessible by people with hand-motor or visual disabilities.

In this context, a "soft", computer-based interface was to designed through the AmI Designer that would allow users of different age and computer expertise, as well as people with motor and visual disabilities, to easily, effectively and faultlessly control a large number of lights with different functional capabilities, in an AmI environment.

Related research work in smart home environments and the interaction possibilities for their users (e.g., [7],[11]), has experimented with alternative input and output modalities for home control in an effort to come up with interfaces that are both useful and usable in an everyday real life scenario. However, a lot of the experimentation is targeting very specific groups of users. For example, Lucero et al [12] implemented an advanced multi-source bathroom lighting system that allowed the users to change the color and the intensity of the room from a small touch screen interface. Their target audience in the usability tests was young adults with no disabilities. Although, the tests showed that their light system was easy to learn and use, it would almost impossible to be used by someone with kinetic or visual disabilities. Bonino and Garbo [4] proposed an eye/head-driven application that allows controlling a domotic home through a house gateway. The target audience for their application was people suffering from Motor Neuron Disease. Its functionality, although effective for the people that suffer from this debilitating disease, would be limiting for someone with no such disability.

In contrast to previous efforts, CAMILE was intended to be an application for everyday use that would be equally functional and usable by any user, including the elderly, young children, the visually impaired, and the motor impaired alike, thus providing a positive experience to all users independent of their capabilities.

5.2 CAMILE User Interface Design

CAMILE has been designed as an easy to use interface for controlling the multiple light sources available within a single space. The system's user interface is illustrated in Figure 64.

. The various light controls are arranged on the 2D plane in the same pattern as the respective light sources are positioned in the actual room. The background is deliberately flat and "mundane", so that interactive controls stand-out. On it a faded image of the room is drawn that only includes the most prominent static landmarks (e.g., the window and the doors) in order to aid sighted users to mentally align the interface's orientation to the real space.

Different types of light sources are depicted using different symbols. The current state of each light is reflected on its control's image. When the user touches a light control, a circular menu pops-up (e.g., see the point where the hand touches in Figure 64.), offering related control options, depending on the capabilities of the specific lights, e.g., alternative colors for the LED spot lights and alternative light intensity levels for the neon lights (see Figure 65.). When a selection is made, a sound is played and the symbol's color changes accordingly.

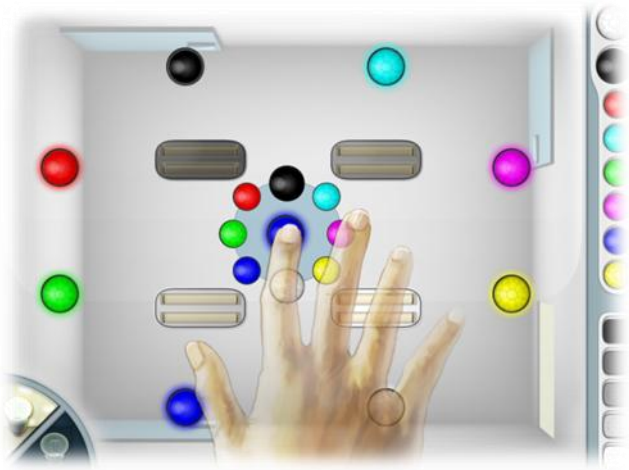


Figure 64. CAMILE's user interface
(*Also depicting the user's hand selecting the central spot light*)

A number of "accelerator" buttons reside on the edges of the screen. At the bottom-left corner there are two controls for quickly turning all the lights on and off, respectively. At the top right edge a color palette can be used to affect the state of all LED spot lights concurrently, while another group of controls at the bottom right corner affects the state of all neon light sources.

Figure 65. Example of a circular menu for setting the intensity of a neon light

5.2.1. Design Rationale

This section discusses the design rationale behind the CAMILE's user interface, as well as the various design decisions taken by weighing alternative design trade-offs.

Overall interface layout

Since the spatial layout of the lights is particularly meaningful for the users, Norman's [15] notion of natural mapping was adopted, and the light controls were arranged on a 2D plane in the same pattern as the respective lights in the room.

Symbol design

Due to the nature and broad audience of the application, the interface solely relies on symbolic (i.e., icons), rather than textual (i.e., labels) information. Different types of lights have a different visual manifestation, so that users can easily infer their capabilities.

Use of landmarks

In order to take advantage of the natural mapping attribute of the interface, the relation among interactive elements in the interface and real world objects had to be made obvious to the users. In line with the adopted symbolic design approach, this had to be done visually. Thus, landmarks, i.e, any prominent objects in a particular environment, were used to provide spatial reference points in the interface. According to Ark et al. [1] landmarks provide a basis for the alignment and formation of mental models, and are used to decide where one is located within a spatial environment. It was decided to use only exit points (e.g., doors and windows) as landmarks for several reasons: (a) every room has at least one; (b) they are very prominent and easy to identify; (c) their position is always fixed; (d) they are in the periphery of the room, thus allowing us to keep the main part of interface clean and uncluttered; and (e) they can easily be represented by an abstract symbol without any identification problems. Furthermore, the representations of the various lights can also act as landmarks.

Interaction devices

A touch screen was selected as the preferred interaction device (see Figure 66), since it fitted the task and it is generally deemed as a very intuitive device for novice computer users [18]. One of its basic advantages is that the input device is also the output device [8]. A known fact about touch screens is the difficulty to point at targets that are smaller than the finger width [2]. Although some research solutions have been devised to overcome this problem [2]; [17], still it is better to proactively design for this fact, when an application does not require high resolution selection. Thus, in the case of CAMILE, the following related design decisions were taken:

a. Use touch controls that are big enough to accommodate an adult's finger. According to ISO 9241-9 [9] "the size of the touch-sensitive area should be at least equal to the size of the ninety-fifth percentile male distal joint breadth". Regarding elderly users, Jin et al.

[10] suggest that the preferable size would be 19.05 mm^2, but when the available space is limited, a button size of 16.51 square mm is also acceptable.

b. Leave enough space among neighboring buttons. ISO (2007) recommends providing space of at least 5 mm around each touch target. Jin et al. [10] found that for rows of adjacent buttons on a single screen, older adults prefer a spacing of 6.35 mm, and, according to Sun et al. [18], spacing between buttons has much less impact on performance than button size.

c. Make the touch areas and screen buttons easily distinguishable from other graphics [14].

d. Use "touch guards". Due to the aforementioned low pointing resolution of the human finger, a common problem of touch screens emerges when a user tries to interact with a pop-up object (e.g., a menu or a dialogue) and accidentally touches a nearby overlapped object. A "touch guard" is a "protective" layer (which may or may not be visible to the user) that blocks interaction to a specified area underneath and around the current object, thus preventing accidental touches.

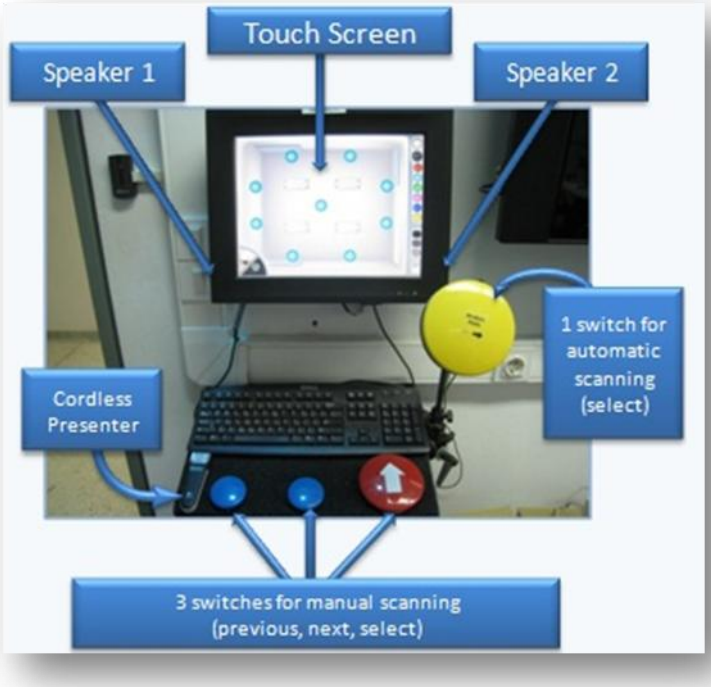


Figure 66. Hardware set-up used for the user-based usability evaluation of CAMILE

In order to make the interface accessible to hand-motor impaired users, three binary switches (2 small and one big Buddy Buttons by Tash Inc.) have been attached on a small self underneath the touch screen and another big Buddy Button was mounted on an adjustable arm (see Figure 66). The 3 switches are used in combination with the manual scanning technique while the single switch is used with automatic scanning (see section 3.1.6. Scanning Orders). For the blind users, the Logitech's 2.4 GHz Cordless Presenter was

selected, a device that is typically used for changing slides while delivering MS PowerPoint® presentations. In the case of CAMILE, it is used as a very simple remote control, since blind users do not have to reside near the screen while using the system.

Auditory feedback

In order, on the one hand, to compensate for the lack of tactile feedback, and, on the other hand, to provide multimodal cues whenever a button is pressed on the touch screen, visual feedback is always accompanied by a related sound effect. In fact, using auditory feedback between 50 and 400 ms significantly reduces error rates when small targets are used [3].

Analog vs. discrete control

A key design decision concerned the level of control that users should be provided. The lights console presented above, due to its analog nature, provides a very fine-grained level of control. Although this may be a desirable attribute for an expert user such as a professional lights engineer, it can be highly inefficient and ineffective for a casual user. Also, in the case of lights control, the actual usefulness of analog controls is questionable, since setting the color LED lights by manipulating 3 sliders that control RGB values is a very challenging task and also the analog spectrum of light intensities supported by a dimmable light can easily be replaced by a selection of 4-6 discrete values without really losing any needed functionality. As a result, users can only select from a palette of predefined values representing distinct colors or light intensities depending on type of the light. This fact may apparently seem restrictive, especially in the case of the colored LED lights, but qualitative feedback from the evaluation sessions (see Chapter 6) showed that most users are willing to sacrifice some freedom of choice over ease of use. Still, for more demanding “power” users, in a future version of the system a single color slider covering the entire color spectrum will also be presented along with the current color palette.

Circular menus

Since discrete control was selected for choosing the desired light color and intensity values, some kind of menu was required for presenting the available choices. Since the number of the available choices is limited and static, instead of the typical windows drop-down menus or grid-like color palettes, it was decided to employ circular menus (see Figure 64. and Figure 65.). In such menus, all items are at an equal distance from the point where the cursor (in the specific case, the user’s finger) is located, and, according to Fitts’ law [13], the time for selecting a target on a screen is proportional to the logarithm of the distance to the target divided by target width. Also, circular menus have been found to be the most efficient when they contain only few items [15].

Accelerator buttons

Except programming each light separately, another typical task for lights control is group control. Some typical examples include turning on or off all the lights when entering or

leaving the room, creating a specific ambience by turning all the LED lights at a specific color, or dimming all neon lights at the same level. In order to support these tasks, a number of "accelerator" buttons were added to the periphery of the user interface. The reason for this placement is two-fold: (a) these buttons do not clutter the main part of the interface; and (b) based on Fitt's law, the four corners, as well as the sides of the screen, are the most quickly (and unmistakably) accessed parts [19].

Scanning for motor-impaired users

In hierarchical scanning, there are two types of objects the user can interact with:

a. *Container objects*, used to group related objects and increase scanning efficiency. Some examples are "all the spotlights", "the neon lights' intensities palette". When such an object is selected, scanning is "locked" inside its contents until the user selects to exit.

b. *Simple objects*, which cannot contain any other object, e.g. a single light control. When such an object is selected, a corresponding action is performed; e.g., the spot light turn into the selected color.

CAMILE's scanning hierarchy and sequence has been designed so that frequently performed actions (e.g., turn on/off all lights) reside at the top level and similar items are semantically grouped (e.g., all neon lights, all LED lights, dimming accelerator buttons).

Depending on the object type, the focus frame can have two different states, which are usually visually indicated by a differently colored and shaped frame:

- *Select / enter state* (denoted by a green-colored frame, see Figure 67 - left): Upon selection, if the object is simple, then a related action is activated; if it is a container, then the scanning focus shifts to its contents.

- *Exit state* (denoted by a red-colored frame, see Figure 67 - right): Only container objects can be in this state. Upon user selection, the state changes to "enter", so that the user can either re-enter the object's contents (using "select"), or move on (using "next" / "previous"). In the case of CAMILE, we an arrow symbol was added to the frame, so that color-blind players can differentiate the two states.

Figure 67. Example of different states of the focus frame: *enter* (left) and *exit* (right)

The focus frame becomes visible only when a user starts interacting with the system using the binary switches residing underneath the touch screen. If the user presses one of the 3 switches, then manual scanning is activated. If the user presses the single switch, then automatic scanning is activated. If the user touches the screen, then scanning is deactivated and the focus frame is hidden again.

Speech-enabled scanning for visually-impaired users

A slightly modified version of hierarchical scanning was utilized for making CAMILE accessible to visually impaired users. The basic difference is that, in this case, there is only one state. When a "container" object is selected, the focus does not "lock" in its contents; i.e., when the last contained item is scanned, the focus moves back to the container. This technique is more suitable and easy to use for blind users than hierarchical scanning, and works similarly to the familiar way they typically use the "Tab" key for accessing Windows applications and Web pages. When focus changes, the current interaction object and its state is spoken using speech synthesis. Each light has a unique identification name according to its type and position in the room. In CAMILE, lights of the same type were differentiated using their relative position to the user when facing the entrance door (see Figure 62.). There is also the option of using user-defined names through an initialization file, so that they best match the user's mental model (e.g., neon light over the couch, spot light over the tv). Blind users interact with the interface using the buttons of the wireless presenter device, similarly to the way that motor-impaired users use the three switches.

Alternative color encoding for the color-blind

Two different design features can provide for the case of color-blind users, who might not be able to differentiate among the various colors, and also for the case in which the interface is rendered on a black and white screen. Speech can be enabled so that the light colors are read by the system, but also an option can be triggered through the application's configuration file where the first letter of each color is added on the respective color selection button. The black and white options are not augmented with a letter and so the possible ambiguity of the letter 'B' (between 'Black' and 'Blue') is avoided.

5.3 Discussion

For the creation of CAMILE application, the steps described below were followed. First of all, the AmI Designer was used to design a user interface which consists of different, simple widgets such as, Label, Panel and ImageButtons. Next, the light services were bind with the specific action widgets, followed by the definition of the widgets' scanning order. Lastly, GUI Signals were created to influence the GUI depending on the incoming Middleware Signals. After design accomplishment and the production of XML description, AmI Player was ran at touch screen PC. This way, the interface that controls the lights in the AmI laboratory was created. In conclusion, the results that came out of this experience are the AmI Designer is a very easy and useful tool which allows fast creation of accessible GUI.

CAMILE was designed through AmI Designer by an expert user without the need to write "a line of code". Thus, the use of AmI Designer greatly simplified the development, and allowed several fast prototyping cycles for effective usability evaluation (see section 6.2). Without AmI Designer, the same development would have required hours of programming, as well as specialized programming knowledge and experience. On the other hand, AmI Designer made the development easy because the produced results are visible at design time, and changes in the UI need only a few "clicks". The conclusion that comes out from this process is that the AmI Designer can be used for the easy and quick design of accessible applications to

manage a Smart Home. For these reasons the developed tools, AmI Designer and AmI Player, have significant potential to bring positive impact on the development and adoption of AmI technologies.

6. Evaluation

6.1 AmI Designer

AmI Designer targets a particularly expert user population, namely user interface designers. The evaluation process of the tool has therefore followed a hybrid approach involving a mixture of user satisfaction and expert evaluation methods.

The evaluation method was selected based on the consideration that, besides general heuristics, there are no available heuristics or guidelines for evaluating a design support system from the perspective of the designer. Additionally, AmI Designer is a unique tool, i.e., there are no known existing systems providing equivalent functionality that can be used for comparative assessment. These two facts have determined the selection of empirical evaluation methods. On the other hand, the existence of a high fidelity prototype made feasible the use of a user-based method to assess the users' subjective opinion on the perceived usability of the system. The IBM Usability Satisfaction Questionnaires (Lewis 1995[58]) were adopted for subjective usability measurement in a scenario setting. These questionnaires are available for public use, have been satisfactorily in use for several years now and are considered as extremely reliable. They include an After-Scenario Questionnaire (ASQ), which is filled in by each participant at the end of each scenario, and a Computer System Usability Questionnaire (CSUQ), which is filled in at the end of the evaluation. The two questionnaires adopt a scale from 1 (strongly agree) to 7 (strongly disagree).

6.1.1 Usability Evaluation Set-up

All the evaluation sessions took place in the main room of the AmI Sandbox. Along with each participant there was one evaluator in the room. The evaluator first introduced the participant to the goals and functions of the system and then read aloud a number of tasks that the participant should try to accomplish. The evaluator would answer any raised questions and would help the participants if they got confused or stuck with a task.

Participants

Seven volunteers participated in the evaluation, 2 females and 5 males. All the participants work in the Human Computer Interaction Laboratory of ICS-FORTH as accessible user interfaces designers.

Process

The process started with introducing the participants to the functions and goals of the room. After that, the objectives of the experiment were explained to them, and the process of the test was described. Each participant was then given a consent form that described everything that was going to take place during and after the test.

After that, the AmI Designer's interface was demonstrated to each user. All participants were asked to perform the following series of design tasks (

Appendix I – AmI Designer Evaluation Tasks):

Task 1: Design a Simple GUI

Task 2: Turn Change Widgets' Properties

Task 3: Bind Items and Services and attach Label to Pane

Task 4: Signals' Manipulation

Task 5: Make a Scanning Order

When the users were finished with the tasks, they exported the XML Description and "played" with their own generated accessible user interface. During the experiment, the participants were requested to freely express their opinion about the system, mention any problems they faced and ask questions. After the experiment, the participants were requested to fill out the IBM Computer Usability Satisfaction questionnaire (Appendix II – AmI Designer Evaluation Questionnaire) which included statements about the overall usability of the system.

6.1.2 Results

The results of the user satisfaction questionnaire are reported in

Table 2 (ASQ), Table 3 (CSUQ) and Table 4 (Total). As previously mentioned, the two questionnaires adopt a scale from 1 (strongly agree) to 7 (strongly disagree). So, the total results of the first table are from 21(strongly agree) to 147 (strongly disagree) and the results of the second one are from 7(strongly agree) to 49 (strongly disagree). Table 4 presents the total score of all questioners. The first column (red color) shows the negative score and the second column (green color) presents the total user satisfaction. The total user satisfaction results are also presented in the diagram of Figure 68.

Table 2. ASQ Results

	Tasks' Negative Marks (ASQ)														
	Task 1			Task 2			Task 3			Task 4			Task 5		
	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)
User 1	2	1	1	2	2	1	3	3	3	1	1	1	1	1	1
User 2	1	1	1	2	2	1	2	2	1	2	2	3	1	1	1
User 3	1	1	1	2	1	1	3	1	2	1	1	1	1	1	1
User 4	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1
User 5	1	2	1	1	1	1	3	3	1	3	2	1	1	1	1
User 6	1	2	2	1	1	2	3	3	2	2	2	2	1	1	2
User 7	2	2	2	4	4	4	4	4	5	2	2	2	4	4	4
Total	**29 (21 - 147)**			**36**			**51**			**34**			**31**		

Table 3. CSUQ Results

	General Questions' Negative Marks (CSUQ)																	
	General Questions																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
User 1	1	2	2	1	1	1	1	1	1	1	1	1	1	1	4	1	1	2	1
User 2	3	2	1	1	2	1	1	2	1	1	1	1	1	1	1	1	1	2	1
User 3	1	2	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	2	1
User 4	1	1	1	1	3	2	2	2	1	2	2	1	1	2	1	1	1	2	1
User 5	2	2	1	2	2	1	2	1	3	3	1	2	1	1	3	1	1	1	2
User 6	2	2	2	2	2	2	1	2	3	3	2	2	2	2	2	1	2	2	2
User 7	3	3	3	3	3	3	4	3	4	5	4	4	5	4	3	2	2	3	3
Total	**13**	**14**	**11**	**11**	**14**	**11**	**12**	**12**	**14**	**16**	**13**	**12**	**12**	**12**	**15**	**8**	**9**	**14**	**11**

Table 4. Total Results

	Results	
	Negative (238-34)	Positive (100%)
User 1	49	92,65
User 2	48	93,14
User 3	41	96,57
User 4	44	95,10
User 5	55	89,71
User 6	65	84,80
User 7	113	61,27
Total	59,29	87,61

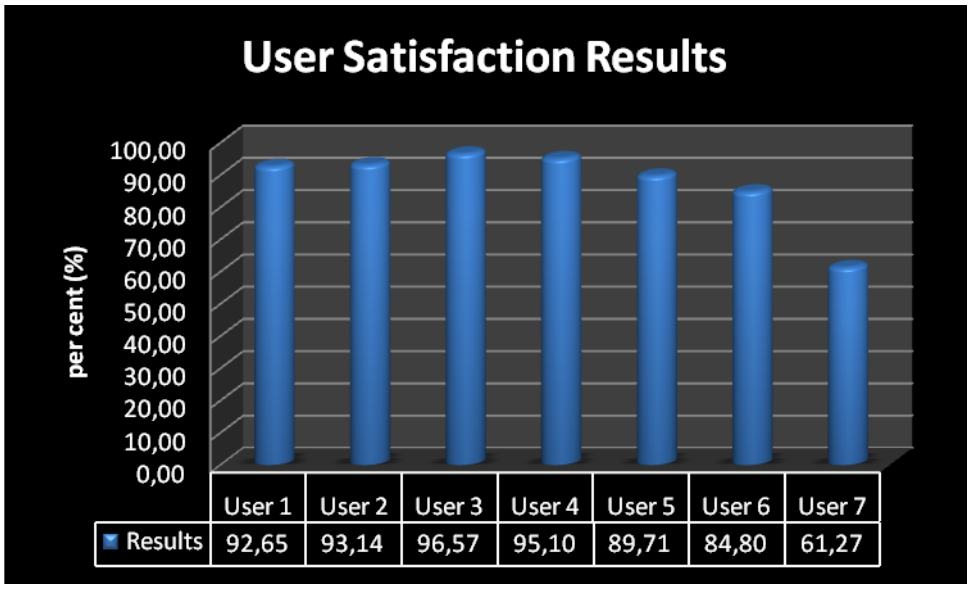


Figure 68. User Satisfaction Results Diagram

The total user satisfaction as perceived by all the participants was 87.61 which is considered as quite high. The individual calculated total user satisfaction as perceived scores per participant are presented in Table 4 and in the diagram of Figure 68.

6.1.3 Discussion of the evaluation outcomes

In general, users were satisfied with the AmI Designer Application. Most of them reported that the application provided a lot of functionality, but it had a few problems in the user interface. The main identified problems were:

- The binding process of services and items is difficult for the users. They would like to see a more graphical approach.
- The design stage has few functionality (resize, move, create and delete). Some of the functionality can be used with right click in each stage's item.
- The property grids must be more functional. When user selects a property, such as a URI or color, a new form should be presented. Then, the user could = select a URI from a dialogue box and a color from a panel, instead of writing it by hand.
- Users could not attach an item to a panel, if the item has been created before the panel. This occurred because there is no functionality about the change of the Z dimension of an item.
- The way that the widgets are created in the stage is the drag and drop technique. Some users would like to insert widgets by clicking their image from the toolbox.
- There is no way to undo a process. This would be useful to make the error recovery process more easy and quick.

The above are the most significant problems identified in the application. The diagram in Figure 69 provides a graphical representation of the answers to the questionnaire. Higher values on the y axis correspond to more negative values of the answers. The main problem was in Task 3, which is the binding process. This problem causes the negative results for questions 2 (system simplicity), 5 (efficiently complete) and 15 (organization of information). Finally the second main problem is error recovery (question 10), which can be solved through the undo functionality.

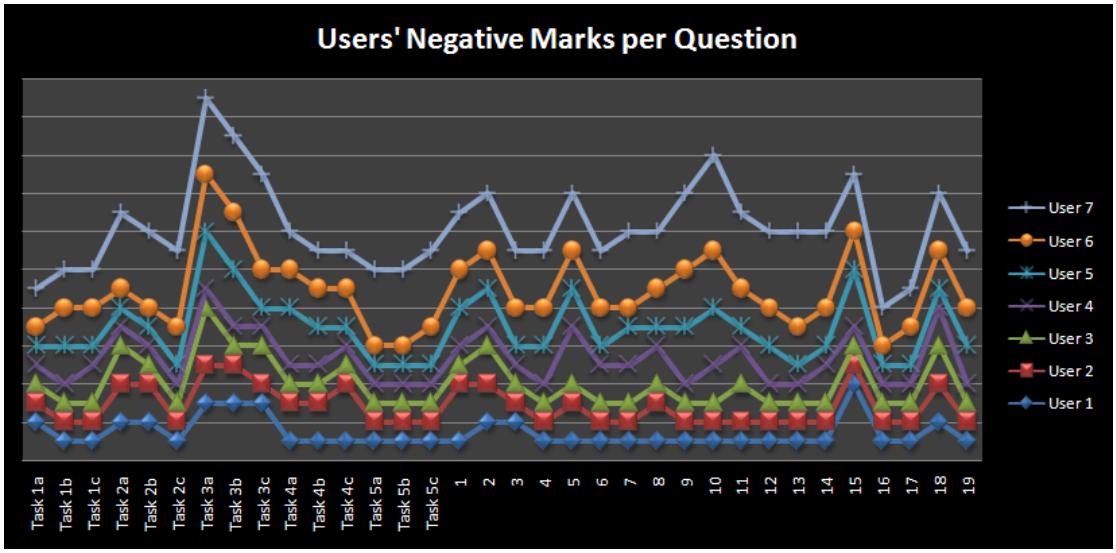


Figure 69. Users' Marks per Question

6.2 CAMILE UIs

6.2.1. Usability Evaluation Set-Up

The goal of the evaluation of CAMILE was to assess it against the four usability principles as outlined by Booth [5]:

(a) *Usefulness* – does the light controller meet the users' needs? Does it help the user achieve his or her goals?
(b) *Effectiveness (ease of use)* – what is the degree of effort to use the controller? Can the users with disabilities use the application successfully?
(c) *Learnability* – Is the application ease to learn? Do the users use it more effectively after some amount of time?
(d) *Likeability* or *Satisfaction* – What do the users think about the application? How do they feel about it?

For that purpose, the evaluation included both a quantitative and a qualitative method of measurement. Specifically, for the quantitative analysis, Brook's System Usability Scale (SUS) [6] was used, a simple, ten-item attitude Likert scale, because it covers a variety of aspects of system usability, such as the need for support, training, and complexity. For the qualitative analysis, the think-aloud method was used asking the users during and after each task to comment on their experience.

All the evaluation sessions took place in the main room of the AmI Sandbox. Two cameras were used to record the experiments, one for capturing the touch screen area, including system visual and speech output, as well as the user's hands (when appropriate) and one for capturing the face, body and speech of the participant. Along with each participant there were two evaluators in the room. One of the evaluators was first introducing the participant to the goals and functions of the system and then reading aloud a number of tasks that the participant should try to accomplish, while the other was mainly operating the two cameras.

Both evaluators would answer any raised questions and would help the participants if they got confused or stuck with a task.

Participants

Ten volunteers participated in the evaluation, 6 females and 4 males. For each one of them a short personal information questionnaire containing demographic and computer experience information was filled in. A brief description of the participants is provided below:

- Three participants were blind. One had high computer expertise, using a computer on a daily basis through a Braille display and a screen reader, one had intermediate expertise, using a computer on a daily basis through a screen reader, and one had no computer expertise at all, but was using a cell phone with screen reading software. The age of the participants ranged from 18 to 40 years.
- One participant had a severe hand-motor disability and was using a computer on a daily basis through an adapted wireless mouse device. His age was between 41 and 55 years.
- Two participants were minors (aged 11 and 9) with no disabilities. The older one had extensive computer skills and expertise, while the younger one was quite familiar with computers but with limited expertise.
- The other four participants had no disabilities. Three of them were in the 18-40 age-group. Two of them were experienced computer users and one had almost no computer expertise. Another participant was over 55 years old and had no computer expertise, and also mentioned that he almost never uses any type of remote control.

Process

The process started by introducing the participants to the functions and goals of the room. After that, the main objectives of the experiment were explained to them, and the process of the test was described. Each participant was then given a consent form that described everything that was going to take place during and after the test, requesting their signature for their consent to the video taping, and stated that their personal information would remain anonymous and would only be used for the purposes of this experiment. In the case of the two children, their parents signed the consent form and also remained inside the room during the experiment.

Then, the participant's profile (e.g., age, computer expertise, use of similar technologies, disabilities) was captured by a means of a short questionnaire that was filled in through a brief interview session before the experiment started.

After that, the arrangement of the physical light sources in the room was explained, and the CAMILE's interface was demonstrated to each user, according to the way it was expected to be used by the particular user category. Sighted users with no hand-motor impairments tested the system using both: (a) the touch screen; (b) the remote controller. They also used the system "blindly" (i.e., while not looking at the screen). Blind users used the remote controller, and the hand-motor-impaired user used the binary three switches. All participants were asked to perform the following series of light control tasks:

T1: Turn all the lights on.

T2: Turn all the spotlights to their preferred color.

T3: Dim all the neon lights to the darkest level without turning them off completely.

T4: Make each wall have a different color.

T5: Dim each neon light to a different dimming level.

T6: Turn all the lights off.

T7: Create a lighting environment to their liking.

When the system was used blindly (either by the blind or the sighted participants), the fourth task provided great insight about the naming conventions that had been selected for describing the light sources when using speech output. During the experiment, the participants were requested to freely express their opinion about the system, mention any problems they faced and ask questions. After the experiment, the participants were requested to fill out the System Usability Scale questionnaire which included statements about the overall usability of the system

6.2.2 Results

Quantitative Results

The mean System Usability (S.U.) as perceived by all the participants was 78.1, which is considered as quite high. The individual calculated Total System Usability scores per participant are presented in Table 5. It is expected that these scores will improve further after design improvements will be made to overcome the few usability problems that were observed during the tests. These problems are discussed in the following section.

Table 5. Calculated System Usability per Participant* and Mean System Usability

Participant	Characteristics	S.U.
P1	Sighted, Adv. Comp., 18-40	97.5
P2	Blind, Adv. Comp., 18-40	70
P3	Sighted, Interm. Comp., 18-40	85
P4	Blind, Beg. Comp., 18-40	85
P5	Sighted, No Comp., 18-40	80
P6	Blind, No Comp., 18-40	80
P7	Motor-Imp., Interm. Comp., 41-50	80
P8	Sighted, No Comp. Skills, Above 55	47.5
Mean S.U.		***78.1***

* The two children that participated in the experiment did not fill out the SUS questionnaire, so they are not included in this measurement.

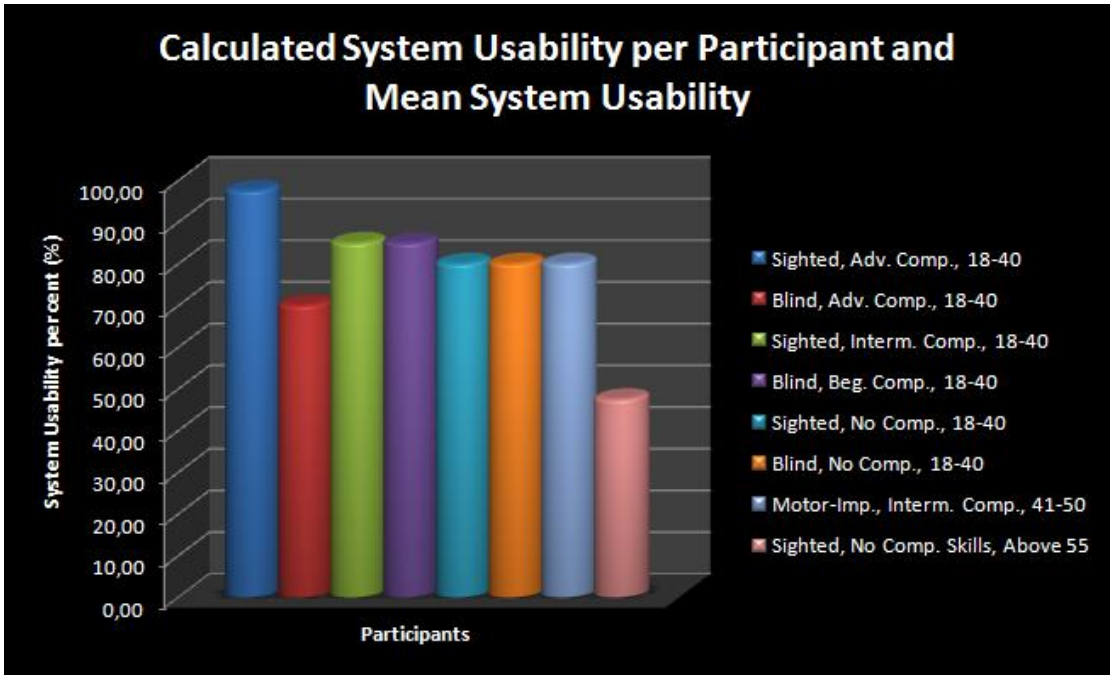


Figure 70. Calculated System Usability per Participant and Mean System Usability Diagram

Qualitative Results

Apart from using the SUS questionnaire for collecting quantitative data, the think-aloud method was also employed in order to collect rich qualitative feedback, by encouraging the participants to comment on their experience using CAMILE. A digest of their comments and thoughts, as well as the evaluators' observations and redesign considerations is provided below.

Usefulness: Overall, all participants found the application very useful. The sighted users liked the simplicity of the interface and particularly the fact that they could control all of its features with a touch of an icon. They also liked the remote control functionality that allowed them to control the lighting from another room. The visually impaired users also thought that the application was useful. When asked if this was an application that they would use, they all said yes and commented on the fact that currently the task of lighting is quite challenging for them, since they have difficulty locating the switches and also they often can not tell the current state of a light. The fact that they could control one aspect of the house in such detail by using just a simple remote controller was very appealing to them and they were excited about the possibilities of such a system for the control of other devices around the house. The motor-impaired user liked the fact that all light switches could be controlled from a single point without having to physically move around the room.

Effectiveness: The sighted users found the touch screen very easy to use. They showed high confidence in using it during the tasks and they expressed no confusion or difficulties. However, some of them got confused with the functionality of the controller's left and right arrow buttons when they used it while looking at the screen. More specifically, it seems that they subconsciously associated the arrow keys to moving left and right, instead of their actual functionality, i.e., selecting the next or previous item. This situation emerged when they were interacting with the lower half part of the interface, where for example, when the

right arrow button is pressed, the selection box moves to the left, since there is located the next item. In order to overcome this counterintuitive situation, in the next version of CAMILE it is planned to experiment with controllers with alternative layouts. When the same users used the remote controller without looking at the screen, they didn't face any problem, since they couldn't see the actual movement of the selection box on the screen. The elderly person used the touch screen as efficiently as any other user, but had difficulty at first in understanding the way that remote control operated in combination with scanning. Surprisingly, this person found much more easy to use the non-visual version, since it seems to fit better to his mental model.

The visually impaired users found the remote controlled version easy to understand and use, after a couple of attempts. The only thing that got them confused, which also confused the sighted users when they used the system blindly, was the naming conventions that was used for describing the position of the spotlights. After three tests, it was apparent that the description of the spotlights location was too general and confusing and with the help of the testers a better description was elaborated. The respective changes were made to the system, and this problem did not appear in any of the following tests. In addition, two out of the three blind users said that it would have been better for them if the system was used in a way similar to the way that they user their mobile phone, i.e., using two buttons for browsing all the options in a specific level, a selection button for issuing an action or moving down in the menus hierarchy, and a cancel button for moving one level up in the hierarchy. Also, when an item is read, its relative position in the current level should also be described, e.g., 3 out of 7. This approach will also be implemented in the next version of CAMILE. The motor-impaired user found the use of the switch buttons very effective. The user also seemed to get confused at first when having to click the right switch even though the selection box was moving the opposite direction. Still, when asked about this fact, the user said that it should not be considered as a problem, since it is a case that generally occurs while using switches, and it is really quick and easy to overcome.

An initial design decision was that, when users select an option from a circular menu, this does not close in order to easily compensate for erroneous selections. As resulted by observing all the participants this system feature actually constituted a major usability problem, since users often could not perceive that they had successfully completed their indented action.

Learnability: In general, the instruction time that was required to introduce to the sighted users on how CAMILE worked was less than a minute for the touch screen mode, and 3-4 minutes for the remote controller or switch modes. The only exception was when the elderly person was introduced to the way that the remote control operated, that required a tutorial of about 10 minutes. For the visual impaired users more time was spent since the actual room and the layout of the lights had to be described first, so that they could create a mental model of the space. After that it took us about 5-10 minutes to explain how the controller and the system work. In general, CAMILE was very easy to learn and required little instruction before and during the tests from the evaluators.

Likability: In the question, 'Do you think you would be using this system often if you had it in your house?', 6 out of 8 users answered "I absolutely agree" (5 out of 5 in the Likert scale). The testers thought that the system was not only easy to use, but also useful and the majority of them saw the potential of a centralized control system for various home appliances. They also liked the fact that there were different ways of using it. The sighted users found the touch screen the easiest of all, but also liked the remote controller because it allowed them to control the lights from another room without having to watch the screen. The motor-impaired user could make a very coarse use of some parts of the touch screen, with great difficulty and no accuracy, so he really liked the fact that switches were available, supporting easy, accurate but most of all effortless interaction.

6.2.3 Discussion

Overall, the evaluation of CAMILE confirmed that the AmI Designer and the AmI Player in combination provide a quick and easy instrument for delivering accessible and usable user interfaces to a diverse end user population, as required in the Smart Home domain.

Overall, the CAMILE case study demonstrated that the availability of a design tool such as the AmI Designer allows effective fast prototyping and supports iterative design and evaluation cycles, thus greatly contributing also to the overall usability of the designed applications.

7. Conclusion & Future Work

In the next few years, it will be possible to control a large number of standard electrical and electronic devices residing in the home environment through computing technologies. The vision of Ambient Intelligence promises to deliver more natural ways for controlling these devices (e.g., by using hand gestures and speech) or even offer proactive, non user-initiated, control (e.g., a "smart" room knows when and how to change the state of each device in it). But, even when this is achieved, there will still be a need for tangible (i.e., "non-transparent") highly usable interfaces that will allow direct and more importantly universally accessible user control and programming, and will also provide quick and intuitive overviews of the current state of the computer-controllable environment. The two tools reported in this thesis, AmI Designer and AmI Player, have been specifically created to address this need, as they enable automatic generation of accessible graphical user interfaces in AmI environments without the need of programming.

The tool, as it has been described in previous Chapters, can be extended easily and quickly. Some basic extensions of the AmI Designer are already planned which are intended to further improve the easy, fast and functional generation of accessible user interfaces.

The AmI Designer, as showed from the evaluation's results, is a functional application that provides the user the possibility of design accessible user interfaces. The basic changes that are planned in the short term concern its user interface. Certain processes, such as the binding of items and services and the item's attachment to panel, should become more automated and provided in a more visible way in the user. Also the property grid should provide to the user more functionality, such as easier ways of finding colors and pictures.

The widgets that are used in the AmI Designer are detected dynamically. In order for a designer to create a new widget, the required code should be written according to the architecture of the Flash stage and add the new item to the Widget Manager. Three additional facilities could be provided in the future to facilitate this process: (a) a widgets' editor for creating new widgets, (b) creation of complex widgets through saving new interfaces as widgets; and (c) automatic integration of flash objects in the AmI Designer without the need of following specific architectural guidelines and updating the Widget Manager.

One additional feature which it would be useful to the designer is the selection of language or languages to be used at run time of the generated interface. This would allow, for example, to easily customize the system in order for foreign visitors to easily control the smart environment.

As reported in Chapter 5, different input devices can be used for controlling the accessible user interfaces produced through the AmI Designers. In the specific case study, these were touch screen, binary switches and cordless presenter. A useful additional function of the AmI Designer could be support the selection of the input device, and the association of specific tasks to specific device actions. Apart from the input device (Wii remote control, mouse, keyboard), the user could select a way of recognition, such as speech or gestures.

The AmI Player only generates application from AmI Designer's description, and therefore is less amenable to future extensions.

The main foreseen evolution of AmI Player is to support the transport of generated application in different operating system platforms. Currently, the AmI Player does not support multi-platform due to the C# layer. The C# layer is responsible only for the communication with the middleware and the parsing of the XML file and most of the functionality of the generated application is in the Flash stage. Thus, if the communication and parsing are implemented in another programming language, such as JAVA, the AmI Player might run in different platforms and device, such as PDAs. Because the stage is implemented in Flash, another interesting extension would be for the AmI Player to support displaying user interfaces through a web browser.

The findings of the user-based evaluations provide strong evidence that the design approach of CAMILE can be effectively used for creating environmental control interfaces that are accessible and highly usable by diverse user groups, including people with disabilities. In order to further investigate and extend this approach, future work will develop in three main steps. First, it is planned to improve and upgrade the current version of CAMILE based on the evaluation findings. Then, and the new version will be evaluated with a much larger and more diverse user audience. Finally, following a step by step approach, the system will gradually scale up so that it is able to control a much larger number of various devices (i.e., air-condition, doors, windows) residing in different rooms.

As a final remark, the work presented in this thesis has made clear the need for easy and fast creation of accessible user interface in AmI Environments. In such a context, with the rapid spreading of ambient technologies in the home environment, it will become necessary to decouple user interface provision from extensive programming, and facilitate the emergence of new professional roles targeted to quickly and effectively set-up smart environments for their diverse inhabitants, including people of all ages, abilities and backgrounds, and provide users with appropriate means to manage easily and quickly all the intelligent devices that exist in the complex smart environment. Tools such the ones reported in the thesis could significantly contribute to such an evolution.

8. References

[1] Ark, W., Dryer, D.C., Selker, T., and Zhai, S. 1998. Landmarks to aid navigation in a graphical user interface. In Proceedings of the Workshop on Personalized and Social Navigation in Information Space. (Stockholm, March 16-17, 1998). Available at: http://www.almaden.ibm.com/u/zhai/papers/sweden/final.html

[2] Albinsson, P. and Zhai, S. 2003. High precision touch screen interaction. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Ft. Lauderdale, Florida, USA, April 05 - 10, 2003). CHI '03. ACM, New York, NY, 105-112.

[3] Bender, G., T. 1999: Touch screen performance as a function of the duration of auditory feedback and target size. Ph.D. Dissertation, Wichita State University.

[4] Bonino, D., and Garbo, A. 2006. An Accessible Control Application for Domotic Environments. In Developing Ambient Intelligence: Proceedings of the First International Conference on Ambient Intelligence Developments. AmID'06. Antonio Maña and Volkmar Lotz, Eds. Springer, Paris, 11-27.

[5] Booth, P. 1989. An Introduction to Human-Computer Interaction. Lawrence Erlbaum Associates, London.

[6] Brooke, J. 1996. SUS: A "quick and dirty" usability scale. In Usability Evaluation in Industry, P. W. Jordan, B. Thomas, B. A. Weerdmeester & A. L. McClelland, Eds. Taylor and Francis, London, 189–194.

[7] De Salces, F. J., England, D., and Llewellyn-Jones, D. 2005. Designing for all in the house. In Proceedings of the 2005 Latin American Conference on Human-Computer Interaction (Cuernavaca, Mexico, October 23 - 26, 2005). CLIHC '05, 124. ACM, New York, NY, 283-288.

[8] Greenstein, J.S. and Arnaut, L.Y., 1988. Input Devices. In Handbook of Human-Computer Interaction, M. Helander, Ed., North Holland, Amsterdam, 495–519.

[9] ISO 9241-9. 2000. Ergonomic requirements for office work with visual display terminals (VDTs) - Part 9: Requirements for non-keyboard input devices.

[10] Jin, Z. X., Plocher, T., and Kiff, L. 2007. Interfaces for Older Adults: Button Size and Spacing. In Universal Access in HCI: Coping with Diversity – Volume 5 of the Proceedings of the 12th International Conference on Human-Computer Interaction. HCI International 2007. C. Stephanidis, Ed. (LNCS 4554). Springer, Berlin Heidelberg, 933–941.

[11] Koskela, T. and Väänänen-Vainio-Mattila, K. 2004. Evolution towards smart home environments: empirical evaluation of three user interfaces. Personal Ubiquitous Comput. 8, 3-4 (Jul. 2004), 234-240.

[12] Lucero A., Lashina T., and Terken, J. 2006. Reducing Complexity of Interaction with Advanced Bathroom Lighting at Home. I-COM Journal 5, 1, 34-40.

[13] MacKenzie, S. I. 1992. Fitts' Law as a Research and Design Tool in Human-Computer Interaction. Human-Computer Interaction, 7, 1, 91-139.

[14] Maguire, M. 1999. A Review of User-Interface Design Guidelines for Public Information Kiosk Systems. International Journal of Human-Computer Studies, 50, 3 (March 1999), 263-286.

[15] Mills, Z. and Prime, M. 1990. Are All Menus the Same? – An Empirical Study. In Proceedings of the 3rd IFIP International Conference on Human-Computer Interaction (Cambridge, UK, August 27-31, 1990). INTERACT '90. D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, Eds. North-Holland, Amsterdam, 423-427.

[16] Ntoa, S., Savidis, A., & Stephanidis, C. 2004. FastScanner: An accessibility tool for motor impaired users. In Proceedings of the 9th International Conference on Computers Helping People with Special Needs. ICCHP 2004. (LNCS 3118). Springer, Berlin Heidelberg, 796-804.

[17] Sears, A. and B. Shneiderman. 1991. High Precision Touchscreens: Design Strategies and Comparison with a Mouse. International Journal of Man-Machine Studies. 34, 4, 593-613.

[18] Sun, X., Plocher, T., and Qul, W. 2007. An Empirical Study on the Smallest Comfortable Button/Icon Size on Touch Screen. In Usability and Internationalization. HCI and Culture. Volume 8 of the Proceedings of the12th International Conference on Human-Computer Interaction (Beijing, China, July 22-27, 2007). HCI International 2007. (LNCS 4559). N. Aykin, Ed. Springer, Berlin Heidelberg, 615-621.

[19] Tognazzini, B. 2003. First Principles of Interaction Design. AskTog Web Site. Available at: http://www.asktog.com/basics/firstPrinciples.html

[20] Abowd, G.D. (2004) Investigating Research Issues in Ubiquitous Computing: The Capture, Integration, and Access Problem
http://www.cc.gatech.edu/fce/c2000/pubs/nsf97/summary.html

[21] Ahola, J. (2001) "Ambient Intelligence". ERCIM News, No. 47, October
http://www.ercim.org/publication/Ercim_News/enw47/intro.html

[22] Ailisto, H., Kotila, A. and Strömmer, E. (2003) "Ubicom applications and technologies". Presentation slides from ITEA 2003, Oulu, Finland
http://www.vtt.fi/ict/publications/ailisto_et_al_030821.pdf

[23] Anonymous (2002) "Philips' HomeLab: Careful What You Wish For?" *Wireless News*, May, 191

[24] Ark, W.S. and Selker, T. (1999) "A look at human interaction with pervasive computers". *IBM Systems Journal*,Vol. 38, No. 4, 504-508

[25] Bohn, J., Coroama, V., Langheinrich, M., Mattern, F. and Rohs, M. (2004), "Social, Economic, and Ethical Implications of Ambient Intelligence and Ubiquitous Computing,

Institute for Pervasive Computing". ETH Zurich, Switzerland http://www.vs.inf.ethz.ch/publ/papers/socialambient.pdf

[26] Booker, E. (1999) "A Think-Tank Vision: More Comfortable Connectivity". *InternetWeek*, No. 780, September 10:1-8 http://www.internetwk.com/lead/lead091099.htm

[27] Corson, S. and Macker, J. (1999) "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations". *Request for Comments: 2501*, Internet Engineering Task Force, Network Working Group, January http://www.ietf.org/rfc/rfc2501.txt

[28] Davis, G.B. (2002) "Anytime/anyplace computing and the future of knowledge work". *Communications of the ACM*, Vol.12, No. 45, 67-73

[29] Domingo, B.S.S. (2002) "Long-term IT outlook: Internet computing". *BusinessWorld*, November 26, 1

[30] Gupta, M. (2003) "Ambient Intelligence - unobtrusive technology for the information society". *Pressbox.co.uk*, June 17 http://www.pressbox.co.uk/Detailed/7625.html

[31] van Loenen, E.J. (2003) "Ambient intelligence: Philips' vision". Presentation at *ITEA 2003*, Oulu, Finland, January 10 http://www.vtt.fi/ele/new/ambience/evert_van_loenen.ppt (8.5 MB Powerpoint slide presentation)

[32] Lyytinen, K. and Yoo, Y.J. (2002) "Issues and challenges in ubiquitous computing". *Communications of the ACM*, Vol. 45, No. 12, 62-70

[33] Magalhães, M. (2002) "Ambient intelligence in the workplace" (excerpts from speech, translated from Portuguese). Philips USA News Center http://www.newscenter.philips.com/USA/InformationCenter/NewsCenter/FArticleDetail.asp?lArticleId=2455&lNodeId=610

[34] Mish, F.C. and Morse, J.M. (eds.) (1999) *Merriam-Webster's Collegiate Dictionary*, tenth edition (Springfield: Merriam-Webster, Inc.), p. 36

[35] Ribauda, B. (2003) "When the walls have ears". *Computer Technology Review*, Vol. 23, No. 5, May: 31 http://www.findarticles.com/cf_dls/m0BRZ/5_23/103731264/p1/article.jhtml

[36] Siegele, L. (2002) "How about now? (Survey: the real-time economy)". *Economist*, Vol. 362, No. 8257, January 31, 3-18 http://www.economist.com/surveys/displaystory.cfm?story_id=949071

[37] Weiser, M. (1991) "The Computer for the 21st Century". *Scientific American*, Vol. 265, No. 3, September, 94-104 http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html

[38] Wylie, I. (2003a) "Innovation for whom?" *Fast Company*, No. 76, November: 29 http://www.fastcompany.com/magazine/76/innovation.html

[39] Wylie, I. (2003b) "The new AI (Ambient Intelligence)", sidebar in "Can Philips Learn to Walk the Talk?". *Fast Company*, No. 66, January, 44 http://www.fastcompany.com/magazine/66/smartcompany.html

[40] Yang, S. (2003) "Researchers create wireless sensor chip the size of glitter". *UCBerkeley News*, June 4 http://www.berkeley.edu/news/media/releases/2003/06/04_sensor.shtml

[41] Accenture Technology Labs, http://www.accenture.com/xd/xd.asp?it=enweb&xd=services%5Ctechnology%5Ctech_home.xml

[42] Body Area Network - Systemübersicht (in German), Fraunhofer-Gesellschaft, http://www.ban.fraunhofer.de/system/index.html

[43] e2 Home project, http://www.e2-home.com/Folder_16/trials.asp

[44] Georgia Institute of Technology: Aware home, http://www.cc.gatech.edu/fce/ahri/ , Digital Family Portrait, http://www.cc.gatech.edu/fce/ecl/projects/dfp/index.html , Gesture Pendant, http://www.imtc.gatech.edu/projects/archives/gesture_p.html

[45] IBM's Pervasive Computing Lab, http://www-106.ibm.com/developerworks/wireless/library/wi-pvc/

[46] ISTAG, IST Advisory Group, Advisory Group to the European Community's Information Society Technology Program http://www.cordis.lu/ist/istag.htm

[47] MIT Media Laboratory, http://www.media.mit.edu/

[48] Microsoft Research, Hardware Devices, http://research.microsoft.com/hardware/

[49] Philips : HomeLab, http://www.philips.com/InformationCenter/Global/FArticleDetail.asp?lArticleId=2118&lNodeId=818 , Nebula - A human approach to designing interactive systems, http://www.design.philips.com/smartconnections/nebula/ , Pogo - Active tools for a virtual story world, http://www.design.philips.com/smartconnections/pogo/

[50] Microsoft adds XAML to 'Open Specification' list - Software Development Times On The Web, http://www.sdtimes.com/(X(1)S(kw21wu45u03kzpnafqlanyiy))/content/article.aspx?ArticleID=31886&AspxAutoDetectCookieSupport=1

[51] Rob Relyea : January 2004 – Posts, http://www.windows-now.com/blogs/rrelyea/archive/2004/01.aspx

[52] XAML Syntax Terminology, http://msdn2.microsoft.com/en-us/library/ms788723.aspx

[53] MSDN forum post by the WF product manager, http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=218938&SiteID=1#221631

[54] RuleML and BPEL are other examples of XML-based declarative logic languages

[55] Guthrie, Scott (2008-02-22). "Silverlight Tutorial Part 7: Using Control Templates to Customize a Control's Look and Feel", http://weblogs.asp.net/scottgu/pages/silverlight-tutorial-part-7-using-control-templates-to-customize-a-control-s-look-and-feel.aspx

[56] "Microsoft runs into EU Vista charges", http://www.itwire.com/content/view/8988/53/

[57] Foley, Mary Jo (2007-09-25). "Microsoft officially 'extends support' for Novell's Silverlight Linux port", http://blogs.zdnet.com/microsoft/?p=695

[58] Lewis R. J. (1995). IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use, International Journal of Human-Computer Interaction, 7(1), pp. 57-78.

[59] Grammenos, D., Kartakis, S., Adami, I. & Stephanidis, C. (2008). CAMILE: Controlling AmI Lights Easily. In the *Proceedings of the 1st ACM International Conference on PErvasive Technologies Related to Assistive Environments (PETRA 2008)*, 15-19 July 2008, Athens, Greece. [CD-ROM]. New York: ACM Press

Appendix I – AmI Designer Evaluation Tasks

Task 1. Design a Simple GUI

Steps:

1. Create a new GUI with dimensions 1024 pixels width and 768 pixels height.
2. Create
    a. 3 ImageButtons (ImageButton1, ImageButton2, ImageButton3)
    b. 1 Label (Label1)
    c. 1 Panel (Panel1)

Task 2. Change Widgets' Properties

Steps:

1. Change ImageButtons' images
2. Change Label's text, size and color
3. Change the value of the "lock" property of the Panel as false

Task 3. Bind Items and Services and attach Label to Panel

Steps:

1. Bind the first ImageButton (ImageButton1) with the Panel
2. Bind the second button (ImageButton2) with function "18765_1"
3. Bind the third button (ImageButton3) with function "18765_2"
4. Attach the Label to the Panel

Task4. Signals' Manipulation

Steps:

1. Create two Middleware Signals
    a. "AllLightsOn" signal
    b. "AllLightsOff" signal
2. Create two GUI signals with name "LightsOn" and the second with "LightsOff"
3. Add the ImageButton1 as a child to the two new GUI Signal and change their properties (change image)
4. Add the "LightsOn" GUI Signal to the "AllLightsOn" Middleware Signal
5. Add the "LightsOff" GUI Signal to the "AllLightsOff" Middleware Signal

Task 5. Make a Scanning Order

Steps:

1. Create a new scanning order (ScanningOrder1)
2. Add ImageButton1, ImageButton2, ImageButton3, Panel1 to ScanningOrder1
3. Make the scanning hierarchy as follows:
    a. ImageButton1
    b. ImageButton2
    c. ImageButton3
    d. Panel1

Finally, generate and execute your GUI

Appendix II – AmI Designer Evaluation Questionnaire

The IBM Computer Usability Satisfaction Questionnaires

The After-Scenario Questionnaire (ASQ)

The participant should fill-in this questionnaire after each scenario. For each of the statements below, circle the rating of your choice.

1. Overall, I am satisfied with the ease of completing the tasks in the scenario.

Strongly agree  Strongly disagree N/A

Comments

2. Overall, I am satisfied with the amount of time it took to complete the tasks in this scenario.

Strongly agree  Strongly disagree N/A

Comments

3. Overall, I am satisfied with the support information (messages, documentation) when completing the tasks.

Strongly agree  Strongly disagree N/A

Comments

The Computer System Usability Questionnaire (CSUQ)

The participant should fill-in this questionnaire once at the end of the evaluation (i.e. when all scenarios have been completed). For each of the statements below, circle the rating of your choice.

1. Overall, I am satisfied with how easy it is to use this system.

Strongly agree  Strongly disagree N/A

Comments

2. It is simple to use this system.

Strongly agree 

Strongly disagree N/A

Comments

3. I can effectively complete my work using this system.

Strongly agree 

Strongly disagree N/A

Comments

4. I am able to complete my work quickly using this system.

Strongly agree 

Strongly disagree N/A

Comments

5. I am able to efficiently complete my work using this system.

Strongly agree 

Strongly disagree N/A

Comments

6. I feel comfortable using this system.

Strongly agree 

Strongly disagree N/A

Comments

7. It was easy to learn to use this system.

Strongly agree 

Strongly disagree N/A

Comments

8. I believe I became productive quickly using this system.

Strongly agree 

Strongly disagree N/A

Comments

9. The system gives error messages that clearly tell me how to fix problems.

Strongly agree 

Strongly disagree N/A

Comments

10. Whenever I make a mistake using the system, I recover easily and quickly.

Strongly agree 

Strongly disagree N/A

Comments

11. The information (such as on-screen messages, feedback and other documentation) provided with this system is clear.

Strongly agree 

Strongly disagree N/A

Comments

12. It is easy to find the information I need.

Strongly agree 

Strongly disagree N/A

Comments

13. The information provided with the system is easy to understand.

Strongly agree 

Strongly disagree N/A

Comments

14. The information is effective in helping me complete my work.

Strongly agree 

Strongly disagree N/A

Comments

15. The organization of information on the system screen is clear.

Strongly agree 

Strongly disagree N/A

Comments

16. The interface of this system is pleasant.

Strongly agree 

Strongly disagree N/A

Comments

17. I like using the interface of this system.

Strongly agree 

Strongly disagree N/A

Comments

18. This system has all the functions and capabilities I expect it to have.

Strongly agree 

Strongly disagree N/A

Comments

19. Overall, I am satisfied with this system.

Strongly agree 

Strongly disagree N/A

Comments

Appendix III – System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Appendix IV – CAMILE Personal Info

About the present questionnaire

The objective of this questionnaire is to help us to record users' opinions about the CAMILE application. The questionnaire is anonymous and the information that you will provide will be used exclusively in order to ensure that the CAMILE application satisfies the requirements of disabled users or users with particular individual needs.

1. Age

- ☐ Under 18
- ☐ 18 to 40
- ☐ 41 to 55
- ☐ Elder than 55

2. Sex

Please select one from the following:

- ☐ Male
- ☐ Female

3. Frequency of use of computers

How often do you use a computer? Please select one of the following:

- ☐ Less than two times per month
- ☐ Two times per month
- ☐ One or two times per week
- ☐ Three or four times in a week
- ☐ Five or six times in a week
- ☐ Daily

4. Motivations for using computers

Why do you usually use a computer? Select one or more of the following:

- ☐ Text writing
- ☐ Reading and writing e-mails
- ☐ "Surfing" in the World Wide (Internet)
- ☐ Games
- ☐ Other:

5. Use of computer

For each pair of alternatives below, select one degree from 1 until 7, which expresses your opinion concerning computers:

Unpleasant ***Pleasant***

7	6	5	4	3	2	1

Boring ***Fascinating***

7	6	5	4	3	2	1

Useless ***Useful***

7	6	5	4	3	2	1

6. Use of assistive technologies

Do you use any assistive technology, such as:

- ☐ Screen Readers
- ☐ Braille Keyboard
- ☐ Binary switches
- ☐ Other:

Appendix V – Usability Test Consent Form

Usability Test Consent

Please read and sign this form.

In this usability test:

- You will be asked to perform certain tasks on a light controller application.
- You will be asked to comment on the experience of using this application during the evaluation.
- At the end of the evaluation you will be asked to fill in a questionnaire.
- We plan to videotape the session in order to analyze in depth at a later time the observations stemming from the evaluation. The tape will be used only internally within ICS-FORTH. It will not be broadcast or used for any other purposes.
- We will be evaluating the performance of our application, not your performance.

Participation in this usability study is voluntary. All information will remain strictly confidential. The descriptions and findings may be used to help improve our application. However, at no time will your name or any other identification be used. You can withdraw your consent to the experiment and stop participation at any time.

If you have any questions after today, please contact (name) at (phone).

I have read and understood the information on this form and had all of my questions answered

_____ _____

Subject's Signature Date

_____ _____

Usability Consultant Date