Enabling End Users to Program for Smart Environments

Markus Funk
University of Stuttgart
Pfaffenwaldring 5a
70569 Stuttgart
markus.funk@vis.uni-stuttgart.de

Oliver Korn
University of Stuttgart
Pfaffenwaldring 5a
70569 Stuttgart
oliver.korn@vis.uni-stuttgart.de

Albrecht Schmidt
University of Stuttgart
Pfaffenwaldring 5a
70569 Stuttgart
albrecht.schmidt@vis.uni-stuttgart.de

Abstract
In the Internet of Things area, sensor-based smart environments are becoming more and more ubiquitous. Smart environments can support user’s cognitive abilities and support them in various tasks e.g. assembling, or cooking. However, programming applications for smart environments still requires a lot of effort as many sensors need to be programmed and synchronized. In this work, we present a novel approach for programming procedures in smart environments through demonstrating a task. We define abstract high-level areas that are triggered by the user while performing a task. According to the triggered areas, projected instructions for performing the task again are automatically created. Those instructions can then be transferred to other users e.g. to learn how to assemble a product or to cook a meal. We present a prototypical implementation of a smart environment using optical sensors and present how it can be used in a smart factory and in a smart kitchen.

Author Keywords
Programming by Example, Augmented Reality, Smart Environments

ACM Classification Keywords
H.5.m [Information interfaces and presentation (e.g., HCI)]: Miscellaneous.
Introduction and Background

In the area of the Internet of Things (IoT), sensors are becoming more and more ubiquitous. As sensors are getting cheaper and need less battery, it is feasible to equip nearly everything with sensors. By equipping sensors with Internet connectivity and therefore connecting sensors to each other, smart environments can be created. These smart environments are finding their way into many areas of people’s lives. The measured values of deployed sensors can be used to check the properties of the environment, e.g. the temperature in one’s living-room. But more interestingly, the measured values can also be used as an input for automatically triggering context sensitive actions. For example in the living room, the Clapper\(^1\) enables the user to switch the light on and off by just clapping their hands. In more public areas, e.g. in the hallway, motion sensors automatically trigger the lighting when a person is walking by. More recently, sensors and smart home technology are connected to each other to enable controlling systems in one’s home from anywhere. Savant\(^2\) for example, enables controlling the heating, light, or electronic devices from anywhere using a smart phone.

Interconnecting sensors in smart environments to offer cognitive support for end users was also the topic of many projects. Antifakos et al. [1] proposed to equip assembly parts of an IKEA PAX wardrobe with sensors. They use instrumented tools to infer a user’s current action and suggest proactive instructions for assembling. Compared to a printed manual, their sensor-based system can dynamically react upon a user’s action as it is aware of all possible assembly orders rather than printing one fixed order. Other projects use camera based approaches to detect movement in important areas e.g. boxes that are holding parts [2]. The camera is connected to a computer system that can react to picking from the box.

In this work, we present an approach for enabling the end user to program instructions for complex procedures using smart environments by just demonstrating the procedure. In our prototypical system, we use a camera-based approach, similar to Bannat et al. [2]. Using this approach, we are able to define high level actions (c.f. [3]) from which instructions can be automatically created. Our concept enables the user to combine, alter, and transfer created instructions between different users and across smart environments.

Concept

The main concept of this work is to enable the user to create procedures for smart environments, without having to write code, by just demonstrating a procedure to the system by performing it once. Deployed optical sensors are used to automatically sense performed actions, while they are performed. Later, the system is able to playback the procedure in the order it was performed.

Our prototypical system (also see [4]) consists of a depth camera and a projector that is mounted over the area of application (see Figure 1). In an initial step, the user has to define important areas using a GUI (see Figure 2). The system can distinguish between three types of areas: a movement area, a tool area, and a storing area. In the movement area, the system just detects if somebody entered the area. For example in the kitchen, this can be used for detecting if the user’s hand entered a compartment. A tool area is used to mark the regular position of a tool, while a storing area is used to mark an area, where something has to be placed or assembled.

---

\(^1\)www.chia.com/index.php/83-other-products/94-the-clapper (last access 01-05-2015)
\(^2\)www.savant.com (last access 01-05-2015)
For detecting performed activities and creating instructions, we define a high-level representation of performed actions (c.f. [3]). We call this high-level representation a workflow, which consists of a finite number of working steps. Each working step has an initial state and a trigger condition (trigger) for advancing to the next step. A trigger is activated when the action according to the type of the area is performed. A movement area triggers if a movement is detected. A tool area is triggered if a registered tool is taken and put back again, i.e. the tool was used. And a storing area is triggered if an object is correctly placed in the area.

**Recording procedures**

After defining the important areas, the system is ready to record procedures. Thereby, the instructions for procedures in a smart environment are created using a programming by example approach. The user performs a task the same way he or she would do regularly. While performing the task, the system constructs a workflow from the performed actions. This workflow can be played back afterwards.

**Playing back**

For visualizing the actions that the user has to perform, the system uses the projector that is mounted next to the depth camera. The visualization for the areas differ for each type of area. The movement area is just highlighted using a green light. Depending on what is highlighted, the user should infer the action that needs to be done. When an item should be used, the item is highlighted with a yellow color. If an object should be placed at the storing area, the target position of the object is highlighted in red. The shape of the projection indicates the orientation and position of the object that is to be placed (see Figure 3).

**Areas of Application**

We present two different areas of application for our prototype. First, in manual manufacturing to cognitively support workers during assembly. And second, in a smart home environment to support the user with interactive cooking instructions.

**Manual Assembly Task**

Our setup can be used at an assembly working place to survey the construction and detect user interactions. The system can detect if a worker is picking a part from boxes which hold the spare parts using a movement area. Further, the system can detect if the worker is currently working with a certain tool using a tool area. For determining where the worker is assembling a previously picked part, the system uses a storing area. The interaction with those areas can be recorded and saved in an instruction for the smart environment. Later, the instruction can be played back to show the recorded assembly to an untrained worker.

**Interactive Cooking**

Another area of application is creating and consuming cooking instructions in a smart environment. There, the environment can also be divided into logical areas (see Figure 4). In our experimental kitchen, we define a tool area, an ingredients area, and a cooking area. A user can, e.g. teach the system how to cook rice. Therefore, the rice needs to be removed from its position on a shelf, which is defined using a movement area (see Figure 5). Then, the rice needs to be put into a pot on the cooking plate, which is implemented as a storage area. The system then saves the meta information into an interactive cooking-instruction, which can be played back. Furthermore, the instruction can be sent to a friend. As the system is using meta information for displaying the
feedback, the instruction can be easily integrated into another person’s kitchen. There, the rice might be at another position, but the instruction can be adapted by highlighting the person’s specific rice area.

**Conclusion & Workshop Contribution**

In this paper, we presented an approach for enabling end users to program complex procedures for smart environments. We integrated our camera-based prototype into an assembly scenario, and a smart kitchen scenario. As the instructions created by our prototype are based on abstract zones, they can be recorded at one environment and then easily transferred and played back at another environment. We believe that enabling users to create, share, and alter content for the smart environments, may have a similar impact on the Internet of Things as it had on the Internet in the area of Web 2.0.

We believe that our work can significantly contribute to the workshop as we provide a user-centered approach for enabling end-users to program smart environments. In our presentation, we will present videos showing the full potential of our system. Further, we believe that presenting our project will foster interesting discussions at the workshop.

**References**


Utilizing Programmer Communities for End User Programmer Feedback

Abstract

In this paper, we describe a system for large scale feedback and the potential role it could play in programmer communities. Imagine a system that crowdsources feedback from experienced programmers and automatically distributes it to less experienced end user programmers. We believe a system like this may be helpful for communities like maker spaces and open source software, which have diverse groups of contributors who collaborate and support each other. With an expected increase in end user programming communities for customization in the Internet of Things, such feedback systems would likely benefit end user programmers and the code they produce.

Author Keywords
End user programming; crowdsourcing; feedback

ACM Classification Keywords
H.5.2. User Interfaces: User-centered design

Introduction

The Internet of Things is likely to increase the already large population of end user programmers. The growing number of “smart” devices and possible features suggests that niche programmer communities could potentially form around specific interests and projects. We expect that these communities, like current open
source and maker communities, will include programmers of varying skill levels, motivated by the desire to improve products that they use [8]. However, end user programmers often find various aspects of programming confusing, such as program behavior and how to create complex functionalities [7]. Current systems focus on either 1) helping end user programmers solve known problems, or 2) automatically locating issues, without providing help toward solving those problems. We propose a crowdsourced feedback system to provide both of these types of support for end user programmers.

Consider a novice end user programmer, Jamie, who is working on an application to control home access for visitors like nannies and maids. Jamie can ask questions of more experienced programmers through forums, but when Jamie implements a security feature based on information she found on a website, she may not realize that a more secure implementation exists. Experienced programmers may spend time answering questions, but they will not necessarily review every line of code. Now imagine a system that Jamie can have “crawl” her code to find opportunities for improvement, as suggested by more experienced developers in her community. These suggestions could provide example code designed to help Jamie learn and improve her applications. We have begun to explore a crowdsourced system for suggesting this type of code improvement to novice programmers.

**Related Work**

Two existing types of systems help end user programmers improve their code: 1) systems that support programmers who can identify their issues, and 2) systems that can identify issues or opportunities for improvement in code.

**Overcoming programming problems**

Online forums and crowdsourced bug fixes help programmers to solve problems that they have found in their own code. StackOverflow and similar forums can help programmers to overcome problems as long as the end user programmers know which questions to ask [9]. Tools also exist to provide support for fixing bugs, such as HelpMeOut, which collects bug fixes from a population, has experts annotate the fixes, and then presents them to users who have the same issues [3]. Currently, this type of assistance can only help when the programmer already knows they have a problem or when an error alerts them to an issue.

**Identifying programming problems**

Static code analysis and code smells can help programmers to identify issues in code. A variety of static code analysis tools, such as FindBugs [4], check programs for common issues and even allow programmers to author their own checks. Code smell detection systems similarly automatically find potentially problematic code [2]. Yet, these types of systems do not provide information about how to fix the problems or how to improve programming skills.

**A Crowdsourced Feedback System**

With existing systems in mind, we wanted to design a system that would find issues that end users do not realize they have, as well as provide them with information about how to fix those problems. Since programmers often search the web for examples of code to solve their problems [1], we operationalize “feedback” as example code that can be used to solve a
We believe that making the system crowdsourced could reduce the amount of time each experienced programmer would need to spend to provide feedback at a large scale. The model for this system has three main components, as shown in Fig. 1: 1) create a code example and author a rule script for that example, 2) community review of feedback, and 3) feedback presentation to the end user programmer.

1) Code Example Creation and Rule Script Authoring
First, an experienced programmer creates a code example by improving an existing program and then annotates the example to provide assistance for using it. The annotated code example may focus on proper programming practices, such as a more efficient way to code a function. A code example could also show an example of a similar, but more advanced code snippet that might improve a basic program, such as improved security for a login feature. Additionally, end user programmers could likely create code examples as they begin to gain skills.

An experienced programmer then authors a rule script that codifies the conditions under which this code example might help other users improve their program. These rules enable experienced programmers to define which programs qualify for specific feedback, such that the system could then automatically distribute feedback.

2) Community Review of Feedback
Other experienced programmers would then review feedback to ensure quality and generalizability. This process would involve a number of experienced programmers contributing a very small amount of time to check over existing feedback.

3) Code Example Presentation
The system would then use the rule scripts (created in step 1) to determine which programs qualify for certain code examples. A code example would then be shown to the end user programmer as a suggested way for them to improve their code. A user could, hypothetically, receive these suggested code examples automatically, or select to see them on request.

Studies
We ran three studies on crowdsourcing feedback: a study of the feasibility of experienced programmers creating examples and authoring rules, a study on the workflow of a tool for this type of system, and a study looking at the presentation of examples.

Our exploratory study of example creation and rule authoring showed that experienced programmers often created useful code example suggestions and authored rules in pseudocode [6]. Their pseudocode rule scripts also demonstrated how experienced programmers conceptualized authoring rule scripts, providing insight into how to design a tool to support rule-authoring.

Communities of programmers will likely include programmers of varying skill levels, all of whom we believe can be valuable contributors to a crowdsourced feedback system. We developed a tool prototype, which involves four sub-tasks: I) explore an end user’s program, II) evaluate existing feedback for that program, III) create feedback for that program, and IV) author a rule script for that feedback [5]. Preliminary results showed that a broad population of programmers
A. Specific Note
The 'repeat 4 times' loop makes the 'soccerBall bounce' action happen 4 times in a row.

![Repeat 4 times SoccerBall bounce loop]

B. Summary
The ball bounces across the park.

![Repeat 4 times SoccerBall bounce loop]

C. Simple Highlighting
![Repeat 4 times SoccerBall bounce loop]

Fig. 2 We found that these three types of example annotations did not significantly affect novice programmers’ ability to complete a task using an example.

(97% of participants) can evaluate and create feedback, while 71% can effectively author new rules. One concern, however, is that participants spent about 5.5, 11.5, 9 and 14 minutes on each of the four sub-tasks, respectively. While these times may be reasonable with respect to answering questions thoroughly on a forum, ideally, each sub-task would be completed in a shorter amount of time.

We may be able to reduce feedback creation time by changing how experienced programmers annotate example code. We ran a preliminary study to investigate how different annotation styles affect novice programmers’ abilities to complete tasks. Results showed that simple highlighting, as shown in Fig. 2-C, can focus a programmer’s attention on the critical aspect of a code example just as effectively as two types of textual descriptions (Fig. 2-A and Fig. 2-B). This supports the use of crowdsourcing for providing feedback, since simple annotations likely require less time and revision than textual annotations.

Conclusion and Future Directions
As communities of workers form with the Internet of Things, we believe it will be important to harness the knowledge of experienced programmers to improve the programming skills of the end user programmer population. End user programmers may not always know which questions to ask or how to improve their code, so we believe that leveraging the crowd of experienced programmers may improve the code and skills of end user programmers.

Acknowledgements
This material is based upon work supported by the National Science Foundation under Grant No. 1054587.

References
Managed End-User Development That Scales

Jan Kučera
Newcastle University
Newcastle upon Tyne, UK
j.kucera1@ncl.ac.uk

Abstract
This paper presents the current state of .NET Micro Framework, the Internet of Things technology developed and deployed before it was cool, and .NET Gadgeteer, a rapid prototyping platform built on top of it, which anyone from K-12 students to industrial researchers can use to design and develop their own things on the edge.

Author Keywords
End-User Development, Modular hardware, Microsoft .NET Gadgeteer

ACM Classification Keywords
D.2.6 [Software Engineering]: Programming Environments.

Introduction
As a software developer, I am disappointed how user interfaces are becoming less and less configurable and customizable, and I believe that taking this responsibility and capability away from users, we are not only losing their interest in technologies but also making them less credible and aware of how they work, eventually resulting in issues like reduced privacy and/or increased fear. These concerns are only going to increase with Internet of Things, and putting people back in control of devices they are surrounded with and
allow them to understand how technology works is an effective way to rebut this trend.

To achieve this goal, we need an end-user development environment that is both powerful and reliable for advanced users and easy to learn and fun to use for inexperienced users. I believe that .NET Micro Framework and .NET Gadgeteer frameworks coming from Microsoft Research, which are both open source, meet these criteria and allow for simple yet effective end-user development. I am happy to demonstrate my experience with these platforms to participants of the workshop.

.NET Micro Framework
The .NET Micro Framework is a Common Language Runtime running directly on hardware without need for an underlying operating system. It shipped commercially in 2004 in couple of consumer products like smartwatches or home appliances, and in 2006 it was introduced to general public as a development platform.

Currently, it is targeting devices of at least 64 kB RAM and 256 kB FLASH memory, and several processors and development kits spanning from small one like FEZ Mini on Figure 1 to very powerful like Blackfin signal processors were introduced on the market by various vendors.

One of the core feature of .NET Micro Framework is the managed runtime environment, which makes development much simpler for end-users, since a garbage collector takes care about the memory and the hardware architecture is abstracted away. Both C# and Visual Basic.NET languages are supported, so that users that have no programming experience can have an easy starting point.

Compared to other hardware development platforms, the main advantage would be the full-featured Visual Studio IDE for live on-device debugging and advanced features like XML parser, Web Services for Devices or reflection, which enables the unique scenario of devices supporting plug-ins and other code deployable at runtime.

However, the framework has also its weak points, such as suboptimal speed since the user applications are interpreted, inability to fulfill any real-time requirements or lack of USB host support. Remote firmware updates are currently limited. It would be interesting to compare framework’s features and issues with other emerging end-user development platforms.

.NET Gadgeteer
Prototyping devices costs large amount of money, time and expertise. Microsoft Research solved this issue in 2011 by introducing the .NET Gadgeteer [1].

.NET Gadgeteer is built on top of the .NET Micro Framework and one of its goals was rapid hardware prototyping. This is accomplished by standardizing the hardware form factors of individual components like sensors and actuators and providing managed drivers conforming to common interfaces. For example, users can then grab any temperature sensor module, display module and WiFi module, connect them together without any soldering and develop and deploy a simple monitoring thing with notification capabilities in couple lines of code.

Figure 1. FEZ Mini by GHI Electronics. One of the smallest development boards with an LPC2387 system on chip. 72MHz, 96KB RAM, 512kB FLASH.
One such commercially available kit with several sensors and modules is shown on Figure 2.

Apart from rapid prototyping, the .NET Gadgeteer comes with a very appealing and simple to use visual designer. Not only it generates the required initialization code for users, but it can also automatically suggest how to connect modules to the mainboard, so that technical details of the hardware involved does not need to be known (see Figure 3).

The .NET Gadgeteer platform therefore became an interesting way how to learn programming for students on primary and secondary schools as well as universities. A pilot studies had been run in the United States and the United Kingdom, which showed interest and enthusiasm among students [2], and the platform can be used as a part of delivering the UK’s national curriculum for computer science [3]. I believe that students without any previous experience in hardware or software development represent an important target group of end-user development systems in home and hobbyists deployments of Internet of Things.

I have participated on developing the .NET Gadgeteer core and helped to run several workshops with both
students and researchers in various countries, gathering feedback from community of end-users related to the topics of this workshop.

**Finalizing things**

While there exists a considerable amount of platforms for end-user development of hardware and software, the options of designing the final product housing and appearance seem to be still limited.

The .NET Gadgeteer as a prototyping platform is often connected with 3D printing or laser cutting [1], but these are still quite expensive and inaccessible technologies to most households and usually require specialized expertise to be used. I would like to have the opportunity to discuss alternative approaches or suggestions for introducing the same amount of modularity, feasibility and ease of use into finalizing the things of Internet of Things.

**Conclusion**

In this paper I have presented free and open source tools that users of various age and experience can use to build their own devices. Although these tools offer a significant step towards user friendliness and straightforward development process, there are still issues and scenarios where they are not a perfect fit. I have also questioned the existence of affordable and accessible options and tools for physical design of devices. I hope that participation in the workshop would give me incentives about how to further improve the platforms I have experience with as well as share their possibilities with others interested in the field.

Jan Kučera has just started his PhD at Culture Lab in Newcastle University, UK, with background in mathematics and computer graphics at Charles University, Prague. As an intern of Sensors and Devices team at Microsoft Research, he helped developing .NET Micro Framework technologies and preparing teaching materials that use .NET Gadgeteer.

**References**


End-User Empowerment in Lifelogging Activities

Abstract
With the widespread of Internet of Things’ devices, sensors, and applications, the quantity of collected data grows enormously and the need of extracting, merging, analyzing, visualizing, and sharing it paves the way for new research challenges. This ongoing revolution of how personal devices are used and how they are becoming more and more wearable has important influences on the most well established definitions of end user and end-user development. The paper presents an analysis of the most diffused applications that allow end users to aggregate quantified-self data, originated by several sensors and devices, and to use it in personalized ways. From the outcomes of the analysis, we present a new EUD paradigm and language that extends the ones existing in the current state of the art Internet of Things.

Author Keywords
Internet of Things, End-User Development, Quantified self, Lifelogging, Unwitting developers.

ACM Classification Keywords
H.1.2 User/Machine Systems: Human information processing. H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

Stefano Valtolina
Dipartimento di Informatica,
Università degli Studi di Milano,
Italy
valtolin@di.unimi.it

Barbara Rita Barricelli
Dipartimento di Informatica,
Università degli Studi di Milano,
Italy
barricelli@di.unimi.it

Stefano Valtolina
Dipartimento di Informatica,
Università degli Studi di Milano,
Italy
valtolin@di.unimi.it

Barbara Rita Barricelli
Dipartimento di Informatica,
Università degli Studi di Milano,
Italy
barricelli@di.unimi.it

Author Keywords
Internet of Things, End-User Development, Quantified self, Lifelogging, Unwitting developers.

ACM Classification Keywords
H.1.2 User/Machine Systems: Human information processing. H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.
**Introduction**

The Internet of Things (IoT) concept was coined in 1999/2000 by Kevin Ashton and his team at MIT's Auto-ID Center [1] and rapidly spread around the world thanks to the evolution of sensor technology and its use that is becoming more and more mobile and pervasive [2]. To connect uniquely identified everyday objects in a network allows to send and receive data and at the same time to influence the behavior of the objects in two ways: automatic, on the basis of the collected data, and semi-automatic/manual, according to users’ needs and/or preferences. Recent studies [3, 4] show that the coming of IoT changed the way people use the Internet, and mobile and sensor-based devices. This tendency is more relevant in domains that present pervasive characteristics where the integration of data could help in improving quality of life and in offering an even richer and satisfying experience of use of everyday objects. This type of integration is what characterizes the so-called lifelogging: keeping track of the collected data through all the everyday or occasional activities that may influence people’s quality of life. Lifelogging, initially conceived in the 70s as a 24/7 broadcasting of self-videos, has become today a wide spreading phenomenon, called quantified-self movement, that allows people to keep track of their habits, health conditions, physiological data, and behavior, and to monitor conditions and quality of the environments in which they work and live. Today, a continuously increasing number of lifelogging devices are on the market and become more and more affordable to the masses. Some of the most advanced IoT devices offer solutions based on artificial intelligence and expert systems for avoiding to prompt users too often and risking to bother them with too many questions. The idea to make objects and environments able to take decisions on behalf of the users aims at not disturbing and overwhelming people in their everyday lives. Although these automatic suggestions avoid to bother users by helping them in managing objects more easily, we believe that the user control over connected objects is a crucial element for IoT success. According to this consideration, IoT allows the end users to manage physical devices, interactive systems, and quantified-self data by deciding how to create new usage scenarios and this empowers them more than ever, making them evolve from passive end users to active end-user developers [5]. Although, the definitions of EUD given in [5,6,7] still sound valid to describe the end user as someone interested in using digital devices just for the sake of it and not with the idea of becoming expert in the technology, these definitions do not reflect anymore the current scenario of IoT due the missing of considerations about time, space, and social dimensions. The broadening of the space dimension in the use of digital devices leads to a revision of all those definitions of end users that consider the context of use as fundamental. Another problem with these definitions is that the notion of time in today’s life and the way in which we manage it have deeply changed. The growing computational performance of the digital devices leads towards a growing speed in user’s performing actions and take decisions. Moreover, when dealing with sensors and temporal data, there is the need to make a distinction between valid time and transaction time. The first refers to the instant in which an event actually occurs, while the second is linked to the instant in which the event has been registered in the system. Another aspect that changed in the last decade is the concept we have of the social dimension in which we live: the digital devices have become not only tools to satisfy the
need of getting jobs done but also the key for taking care of social relationships (real or virtual). According to this need to shift the EUD definition towards more time, space and social-centric aspects, in the next sections we present the definition of a new EUD paradigm and language in IoT domain. Specifically, we propose a sensor-based rule language able to support the end user in aggregating and combining data originated by several sensors/devices and in creating personalized use of the quantified self-data. This language aims at enabling end user for unwittingly developing personalized IoT environments according to specific temporal and spatial conditions that may affect the elements in the IoT environment.

A New EUD Paradigm and Language for IoT

The most diffused applications for IoT that exploit EUD principles allow users to define sets of desired behaviors in response to specific events rules definition-wizards that rely on the states of sensors/devices. Such strategy is adopted by those applications that use automated rules-based engines like Atooma (http://www.atooma.com/) and IFTTT (https://ifttt.com) – by using the programming statement IF this DO that, and by Wewiredweb (https://wewiredweb.com/) with the statement WHEN trigger THEN action. Instead, other applications stems from the outstanding work done with Yahoo's Pipes (https://pipes.yahoo.com/pipes/) and they typically use EUD strategies as formula languages and/or visual programming. Applications like Bipio (https://bip.io/) and DERI pipes (http://pipes.deri.org/) offer engine and graphical environment for data transformation and mashup. They are based on the idea of providing a visual pipeline generator for letting the end user creating aggregation, filtering, and porting of data originated by sources. An advanced use of such visual paradigm is offered by WebHooks (https://developer.github.com/webhooks/) that allows the end users to even write their personal API for enabling connections with new sources of data. Both presented typologies of EUD strategies, adoptable in the context of the IoT applications, offer a solution able to gather information from across the net and trigger specific actions when certain things happen. The adoption of the IF-THIS-DO-THAT/WHEN-TRIGGER-THEN-ACTION patterns are not enough to deal with more sophisticated rules based on time and space conditions. On the other hand, the second type of applications offers a too complex solution for supporting the end user in expressing their preferences. Pretending that the end users are able to deal with APIs of several sensors/devices put at risk the success of the EUD approach. Another problem with the current state of the art regards the fact that in the most diffused applications the social dimension is commonly taken care of, while time and space dimensions are almost never considered. To face these problems, we propose an extension of the IF-THIS-THEN-THAT paradigm by presenting a sensor-based rule language able to support the end user in defining rules in a more articulated way but keeping the complexity at an acceptable and accessible level. The idea is to define a paradigm able to allow end users to design triggers that depend also on time and space and not only on social media content, like most of the applications in the current state of the art. The introduction of time dimension allows end users to set triggers that can be fired at some specific time, delayed in case of certain conditions are verified, and may be repeated until some event happens. The space dimension gives end users the chance of linking
triggers to the place/area where they currently are, where they will possibly be in the future, where they are moving into, or where some events are taking place. The EUD paradigm we propose aims at supporting the end user in composing space/time-based rules for extending the well-established but not powerful IF-THIS-THEN-THAT paradigm. Our Sensor-based Rule Language follows syntax, semantics, and grammar of a Policy Rule Language proposed in [8], and is based on the ECA (Event, Condition, Action) paradigm [9]. Our language allows to specify rules stating policies for triggering actions (one or a set). The general format of a rule is the following (square brackets denote optional components):

RuleName: ‘‘MY RULE’’
ON Sensor[s]
[WHENEVER ‘‘Condition’’]
Action: ‘‘Some Actions’’
[VALIDITY: Validity_Place-Interval]

A rule consists of several components. The RuleName component represents the rule identifier. Users can retrieve rules by means of such identifier for visualizing, sharing, dropping, or modifying them. Sensor[s] represents the sensor or set of sensors upon which data the rule is triggered. Each sensor exposes a set of parameters which can be used for expressing the conditions. Condition is an optional conditional expression. Action is an expression that states what happens when the condition is verified. Validity_Place-Interval is a special spatial and/or temporal condition also expressed by means of the condition language we developed, representing the space and time period during which the rule is enabled. In this statement, the keyword IN is used for specifying rules that need to be triggered if the data streams refer to a specific geographical place/area. The keyword AT is used to indicate a rule that is triggered at a well-defined time, while the keyword EVERY could be combined with an expression of type PERIOD for repeating execution of a particular action regularly after a fixed period of time has passed. For example, if the interval [EVERYDAY EXCEPT SATURDAY] is specified we know that a rule is enabled every day of the week but not on Saturday. But if Validity_Place-Interval IS not specified, we know that the rule is always enabled. By means of Validity_Place-Interval it is possible to state that certain rules are not always enabled; rather, they are enabled only if an event happens in a specific place or during specific temporal intervals. Such a feature is not provided by conventional apps for IoT.

Conclusion
In this paper, from a study of the most diffused applications for IoT that offer EUD tools, we identified and discuss some open problems and proposed a new EUD paradigm and language to solve them. Further developments of this research will consist in the design and development of an interactive visual system aimed at implementing the paradigm and language proposed and at testing its validity.

References
connected devices. Master Thesis at Uppsala University, Department of Information Technology (2014).


Sociable Technologies for Supporting End-Users in Handling 3D Printer

Thomas Ludwig  
Institute for Information Systems  
University of Siegen  
Kohlbettstr. 15  
57072 Siegen  
thomas.ludwig@uni-siegen.de

Prof. Dr. Volkmar Pipek  
Institute for Information Systems  
University of Siegen  
Kohlbettstr. 15  
57072 Siegen  
volkmar.pipek@uni-siegen.de

Abstract
Recently, digital fabrication technologies such as 3D printers have become more and more common at semi- or non-professional settings, such as university or private households. Such technologies show a high complexity and the close link between hardware and software in this field pose challenges for users how to operate them. With this paper we present first steps towards Sociable Technologies, a concept that encompass hardware with an integrated appropriation infrastructure, for supporting end users in using and understanding such rising 3D printing technologies.

Author Keywords
End-User Development; Appropriation; 3D-printer

ACM Classification Keywords
H.5.2. User Interfaces

Introduction
Over the last years, digital fabrication technologies like 3D printers or laser-based cutting machinery could only be found in specialized industrial companies with expert staff trained in modeling, fabricating and finishing professional artifacts and prototypes. But recently, entry level 3D printers such as RepRaps or MakerBots have also become more and more common in various semi- or non-professional settings such as universities, small businesses as well as in the private hobbyist and Maker scene. While professionals had an excellent training in how to use the machinery, semi-as well as non-professionals often struggle with hardware breakdowns, unexpected effects concerning the printing material, unintuitive modeling tools and complex configurations, which often make the handling finicky and difficult to understand. Especially the mix of IT, hardware as well as the material make it almost impossible to adapt established use concepts from existing IT ecologies like desktop software, mobile devices or ‘2D’ printers [3]. In traditional settings, users have found ways to ‘socialize’ these technologies, but handling contemporary 3D printers cannot be compared to the “plug-and-play” operation of common
computer hardware [4]. Users often discover and apply ways of using such technologies that were not anticipated by the designers and manufactures of the machinery while operating 3D printers or laser-based cutters by trying to make sense of those ‘new’ technologies in the context of existing (and changing) practices.

End-User Development (EUD) is defined by the goal of developing “methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact” (Lieberman et al., 2006). To focus on a successful establishment of software within practice, central concepts of EUD are customization and tailorability, which refer to the change of “stable” aspects of an artifact [2]. The concept of appropriation [6] goes deeper than that of customization or tailoring of software in that it can encompass fundamental changes in practice and embraces the possibility of users adopting and using the technology in ways not anticipated by its designer and therefore enables the discovery and sense making of an artifact while using it in practice.

There is an ongoing discourse in HCI by researchers studying appropriation and related activities as well as how to design technological support for these appropriation activities [1,5]. The integration of so called appropriation infrastructures directly in the information systems which it is intended to support has been thoroughly investigated, tested and seems to have merit [6,7,8]. However, those studies and concepts usually address software ecologies and omit the physical-material issues coming with technologies like 3D printers. Ludwig et al. [3] therefore try to overcome this gap by envisioning the concept of “Sociable Technologies” that aims at supporting users by integrated appropriation infrastructure directly into the machinery itself. They have shown that the ability to articulate and discuss use and configuration issues would benefit from ‘Sociable Technologies’ that describe themselves on three context levels [3]:

1. Internal context: Providing information about the inner workings, about their current state as well as about their component and behavioral structure.
2. Socio-material context: Providing information about the environmental data like location, surroundings, room temperature, maintenance or user/usage data.
3. Task/process context: Providing information about technology used to build/prepare printed models, the machine’s position in a production chain or process, and the purpose and goal of machine usage.

**Implementation of Sociable Technologies**

To establish appropriation infrastructures integrated directly within the machinery itself and therefore getting started with the overall goal of establishing Sociable Technolgies, we focus on enabling end users of 3D printers to understand the machinery as well as its functionality and behavior with regard to its three types of context. A serious shortcoming that became apparent is that the 3D printer itself is a kind of lack box for the users and lacks in methods or functionality to visualize how it works. In order to overcome this issue, we provided the users with more details about the current printing process. Providing the right kind of information allows them to get a deeper understanding...
of how the machine works and the users can become more aware of what happens when and where.

Our concept (Figure 1) encompasses three different steps: (1) a gathering information about the environment as well as the 3D printer itself, (2) a real-time visualizing of all context data about the printer to end users and (3) a storing as well as sharing of details about 3D prints with other end users.

To gather information about the socio-material context, we set up an Arduino board with different Arduino sensors for measuring the temperature, brightness, humidity and vibration and integrated it directly within the 3D printer (Figure 2). Further we integrated a webcam that gives details about the progress and the print artifact. By adding those sensors we were able to gather information about the environmental factors of a 3D print. To get insights into the printer’s state itself, we implemented a ReplicatorG plugin that sniffs all data of a print process, for instance, the extruder temperature, the platform temperature, the extruder movements or the 3D model in STL format.

All the gathered data about the socio-material as well as the internal context of the 3D printer were visualized in real-time on a web-based dashboard (Figure 3). After authentication the users can access the dashboard and see all sensor values, the stream of the webcam and the progress / remaining time of the print. The user has further the option to stop a print if an error occurred.

After a successful or unsuccessful print, the details about the entire process including all sensor data, the configurations as well as 3D models and webcam pictures are stored in a print history. We implemented the print history as part of StatusNet (Figure 4), which is open source microblogging software like Twitter. Within StatusNet all end users have a profile where their prints were saved and published. But not only the user have profiles, the 3D printer itself got also a StatusNet profile. Here, the 3D printer published messages each time after a print process with all details about it.

All users can than comment the different print processes and discuss with other users, which is important for asking for help under the provision of all relevant contextual information, getting support and therefore distributing knowledge and experience to other 3D printer operators. Accordingly, the print histories can be send or post to Twitter where they can reach the appropriate audience and discourse.
We evaluated our concept of Sociable Technologies for 3D printers (1) technically by testing whether all sensor work appropriately as well as (2) from a usability perspective with four end users coming from the academia field. Concerning its usability especially the documentation and discussion options were praised because now the print settings with regard to the characteristics of a specific model and the can be discussed directly within a situation.

As seen in Figure 2 our sensors hardware is not a real integrated part of the 3D printer. As a next step we will therefore build a technology kit that encompasses the presented communication functionalities and can be easily integrated into the building of any machinery to transform it to a Sociable Technology.

References
A Spreadsheet Tool for Creating Web Applications Using Online Data

Kerry Shih-Ping Chang  
Carnegie Mellon University  
5000 Forbes Avenue  
Pittsburgh, PA 15213 USA  
kerrychang@cs.cmu.edu

Brad A. Myers  
Carnegie Mellon University  
5000 Forbes Avenue  
Pittsburgh, PA 15213 USA  
bam@cs.cmu.edu

Abstract  
We present a tool called Gneiss that extends the familiar spreadsheet model to support creating data-drive web applications that use data from the Internet. Gneiss allows the user to extract data from arbitrary REST web services to a spreadsheet and send spreadsheet data to web services dynamically without writing conventional code. It provides new spreadsheet functions and interface elements to support using and manipulating structured data and streaming data. Moreover, Gneiss allows users to create not only visualizations but also full-fledged interactive web applications that can present and modify the spreadsheet data.

Author Keywords  
Spreadsheets; end-user programming

ACM Classification Keywords  
H.4.1. Information Systems Applications – Spreadsheets

Introduction  
This paper presents a tool called Gneiss that aims to extend the familiar spreadsheet model for the Internet era to reduce the barrier of using and publishing data online for end-users. Conventional spreadsheets take local, static data and manipulate it to produce local, static graphs and charts. However, many data sources
nowadays are no longer static files but dynamic, real-time data sources on the Internet such as web services and mobile sensors. Further, what many people want to create are also not static document files anymore but rather interactive websites or web applications [6]. While some conventional spreadsheets like Microsoft Excel are being enhanced with some ability to access web data and share content online, they provide little or no support to using structured data (such as JSON and XML which are common formats for web services) and streaming data (such as sensor data, financial data feeds), and can only let the user publish data as tables and visualizations.

Prior research spreadsheet tools have explored extending the spreadsheet language to support programming GUI applications [1,2] and facilitating cleaning data through programing-by-demonstration [7]. Different from these systems, Gneiss focuses on enabling users to create data-driven applications that interact with a variety of data sources.

Gneiss makes contributions by extending the spreadsheet model to support interacting with arbitrary REST web services, using structured and streaming data, and programming data bindings and interactive behaviors in web applications. Here we briefly describe key features with respect to these contributions. Full features of Gneiss are described by publications at VL/HCC’14 [4], UIST’14 [3] and CHI’15 [5]. A screenshot of Gneiss is shown in Figure 1.

Two-way data flow with web services
Gneiss currently supports arbitrary REST web services that return JSON data. To load data from a web service, the user enters an API in the URL bar in the source pane (Figure 1 at 1). The returned JSON data are shown below the URL bar. The user can extract any field in the return data to the spreadsheet by dragging-and-dropping it to any cell in the spreadsheet. Gneiss uses a programming-by-example approach to facilitate extracting similar items in the returned documents, such as the first field of all array items. When the “Extract similar fields to the spreadsheet” checkbox in the source pane is checked, the system will let the user drag a field to the first cell of a spreadsheet column. It will then automatically fill in the rest of the column with similar fields based on the document structure.

The user can also send the value of a cell in the spreadsheet to a web service by replacing any part of the web API in the URL bar with the name of the cell, using the syntax {{cellName}}. When a cell is used in a web API, every time the cell changes, the system will send a new API request using the new cell value and
update any other spreadsheet cells extracted from this web service with the new return data. This makes the spreadsheet program created in Gneiss easily reusable, as the user can retrieve new data from a web service by editing spreadsheet cells.

Supporting structured data with nested tables
Structured fields extracted from web services (such as a JSON object or array) will be shown using nested tables (see Figure 2). Cells in a nested table can be used in formulas like regular cells, using the syntax cell.childCell. For example, in Figure 2, E1.A2.A1 refers to the cell with value “Ewan McGregor”. To assist the user in further manipulating the data in the nested tables, our tool provides a “flatten” function that flattens a nested table column and stores all values in a column. To use the flatten function the user can enter =flatten(columnName) in the top cell of a column, and then our tool will automatically fill the cells below with the appropriate data. The syntax for columnName is similar to referring to a cell in a nested table, which is column.childColumn. Using Figure 2 for example, the user can let column C be all actors in all Star Wars movies by entering =flatten(B.A.A) into cell C (B.A.A is the column that stores the actor names in each abridged cast of each movie). The flatten function populates column C with a flattened list of actor names. The user can then apply filtering to the column to remove duplicate names and get a clean list.

Cell metadata for manipulating streaming data
Gneiss allows the user to stream data from arbitrary REST web services to a spreadsheet by checking the “Stream this source” checkbox in the source pane and then dragging a desired field to a spreadsheet column. The system then starts to stack the column with the latest values of the field pulled from the web service every 3 seconds. By default the values are sorted descending by time, so the newest value appears at the top of the column. The streaming frequency and how the data are sorted can be changed in a dialog box.

Gneiss introduces cell metadata that describe other attributes of a cell’s value and allow users to manipulate spreadsheet data using these attributes. Each cell has metadata of its value’s provenance and fetched time. This enables users to manipulate data using not only their values but also temporal information of when the value is retrieved. For example, to view the 5 highest values of the day, a streamed column can be filtered to show only the data retrieved today, sort them descending by value and filter to show only the top 5 rows. We design new functions FETCHTIME(cellName) and SELECTBYTIME(startTime, endTime, range) to let users get the fetched time of a cell and select values in certain cells that are streamed between certain times. For example, suppose column B holds latest news streamed from a news data source. The formula =COUNTIF(SELECTBYTIME(“2014-09-21 9:00”, “2014-
Programming interactive, data-driven applications

We extend the spreadsheet model to allow a GUI element property (such as “color” and “value” of a heading element) to use spreadsheet formulas as its value and also be used in other spreadsheet cells. The syntax for refereeing a GUI element property is ElementID!PropertyName. To further allow users to program interactive behaviors in web applications, every GUI web element in Gneiss has “interactive properties” whose value will change based on how the user interacts with the element. For example, the “state” property in every GUI element will change based on how the mouse cursor interact with it (such as “idle”, “hovered” and “clicked”). Or the “value” property in a text box will change based on what the user types in the text box.

This design allows data bindings and interactive behaviors in a web application to be programmed using the spreadsheet’s equation-based evaluation model without the need of conventional event-based model. For example, spreadsheet cell A1 can be set to whatever the user types in a text box in a web application by setting its value to =TextBox1!Value. Then A1 can be sent to a web service by replacing the query parameter in a web API with {{A1}}. Suppose the search result extracted from this web service is put in column B. Text in a list element in the web application can then be set to =B:B to show the data. This creates an interactive search web application. Or the sorting rule of a column can be set to =IF(Checkbox1!Checked, ”Descending”, ”Ascending”) to sort the column dynamically based on if the user checks the checkbox. The user can also use Gneiss’s spreadsheet as a database for the created web application by setting a column to stream data from a web input element (such as a text box). The column will then store the user’s input in the element to the spreadsheet as storing a data stream from a web service. This allows users to create data entry applications like a self-tracking application for custom behavior [6].

Future work

We are developing new functions to help users do data transformations and manipulate nested tables in a spreadsheet. We will also enhance Gneiss to support creating mobile web applications and getting data from web pages and mobile sensors to the spreadsheet.

We can bring a demo of Gneiss to the workshop.

References

Abstract
In the era of Internet of Things (IoT), configuring how one’s own devices work together is very similar to programming by the end user. This is challenging to most users and may also pose a challenge to the growth of popularity of IoT. One possible solution is to facilitates prosocial behavior and enhance collaboration between users.

Author Keywords
Device Configuration; Community Collaboration;

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous. H.5.3 Group and Organization Interfaces: Collaborative computing.

Introduction
In the days of Internet of Things (IoT), every device communicates with each other. Potentially, each unit of ‘things’ or devices should have some intelligence that can work on its own, talk to other units, and adjust its own work based on information from other units. One obvious way for manufacturers to allow users to configure these devices without going into the technical details is to provide different ‘work modes’ for the user.
to choose from. The user only needs to set the devices to different modes. As number of devices one person owns increases, the number of possible combinations of different modes increases drastically and such configuration becomes a challenge that is similar to end-user programming. This makes configuration of devices a challenging task. One possible way to enhance the popularity of Internet of Things is to offer community-supported platforms that allow users to share their ‘recipes,’ ‘success stories,’ and solutions to commonly-faced problems.

**Configuration as End-User Programming**

Website ‘If This Then That’ (IFTTT) ([https://ifttt.com/](https://ifttt.com/)) allows users to configure how the website server should respond to certain conditions. The website server detects events in many different channels, such as Facebook or email. The user can configure the IF part in the rule to specify the event when the rule should be triggered, e.g., ‘IF someone tag me on Facebook.’ Then the user configures the ‘THEN’ part, such as sending an SMS to one’s own mobile phone. This is the simplest form of end-user programming. Imagine in the era of IoT, an owners need to configure his/her devices and how these devices work with each other. Such configuration is an end-user programming task.

In the era of IoT, Each unit in a network of devices may offer different 'work modes' for the users to choose. Regular users do not actually program each of them. Instead, they need to choose among the several modes for each unit. For example, your lock at front door talks to the air-conditioner and coffee machine. Whenever there is someone coming back home, the air-conditioner adjusts itself to make sure that the temperature is comfortable (based on its communication with a thermostat or weather website). Coffee machine prepares suitable kind of coffee, depending the amount of milk the refrigerator tells it. Each of these devices has their own intelligence in adjusting their functions depending on their states and information from other devices. However, the user needs to pre-configure how these devices work on its own. For example, the user needs to configure the coffee machine such that it prepares coffee based on the availability of milk. It could have another mode to make coffee based on the weather. Imagine each of 10 appliances in a household kitchen has 3 different working modes. There are $3^{10} = 59,049$ ways to configure them. This ‘configuration process' itself is a programming task needed to be done by regular users.

**Prosocial Behavior in Growth of IoT**

One feature in IFTTT website is that people can share ‘recipes’ – some existing set of configures about the events and responses. This definitely helps new users of the service to get started using it without being frustrated at trying out different combinations. This can save the effort in trying out new things. In order for Internet of Things to be easily adopted by the mass, social elements need to be built in. There are at least three types of social elements that would encourage the adoption of 'Internet of Things': sharing resources, demonstration, and helping in problems.

Tech-savvy users can share their ‘recipes,’ or templates of configuring a set of devices on a public websites. So that new users can copy and easily get going with their new devices. This eases the burden of regular users in thinking about how to ‘program’ their network of devices.
A sharing platform should also allow existing users to share success stories, simply demonstration of different configuration of a set of devices. To people who are considering whether to acquire IoT devices, such visualization of any benefits can encourage them make purchase decisions.

A collaborative platform can also help users to solve their problems faced in programming their network devices. With crowdsourcing mechanism, users can post their questions on such platform and other users can assist them. This is much more effective than static ‘FAQ’ sections because users can receive specific assistance to problems specific to themselves.

**Author’s Work**

I currently work as a freelancer in both industry and academic. I design user experience and user interfaces for clients in the financial sector. I also teach information architecture, interaction design, and interactive digital communication at universities in Hong Kong. My teaching involves helping students without technical background to master the use of various technological platforms as a whole and in order to achieve communication or business objectives.
IoT Programming needs Deixis

Alan F. Blackwell
University of Cambridge
Computer Laboratory
Cambridge CB3 0FD, UK
afb21@cam.ac.uk

Saeed Aghaee
University of Cambridge
Computer Laboratory
Cambridge CB3 0FD, UK
aghaeester@gmail.com

Abstract
In this paper, we argue that the linguistic concept of deixis is an essential consideration in the design of end-user programming (EUP) languages for Internet of Things (IoT) applications. Deixis offers a theoretical account of how physical contexts can be incorporated in the semantics of information systems. By analysing several examples of EUP4IoT, we offer design guidelines for future systems in this area.

Author Keywords
End-user Programming; Internet of Things; Deixis

ACM Classification Keywords
D.1. Programming techniques

Introduction
Deixis is a linguistic term, describing situations in which the meaning of a sentence can only be resolved by reference to the physical context in which it is spoken. A typical example might be “pick that up now!” (said while pointing at an object on the floor). The meaning of the sentence is unclear without the parenthetical explanation that follows it in this text, and the precise interpretation is only available to a person present in the place where it was spoken, able to see the speaker’s finger and the object at which it was pointing.
This simple definition already introduces key concepts of interaction in ubiquitous computing. "Context" is both a key technical challenge in Ubicomp (e.g., inferring context from sensor data in smart environments), and also a key question in the critical philosophy of interaction for Ubicomp[5]. Deixis is similarly a challenging question for computer processing of natural language, because it draws attention to situations in which text alone is insufficient for semantic interpretation.

A further challenge is that computational models of mind do not always account for embodied action. It is necessary to appeal to theories of external representation [7], to describe the ways in which the pointing finger supplements working memory through persistent availability in a perceived scene.

Deixis in Tangible Interaction
These issues of context and representation have been addressed in the field of Tangible and Embodied Interaction (TEI). However, deixis as a feature of language in itself is relatively under-explored. In part, this is because the traditional command line and GUI is kept distinct from new tangible devices. Although designers of tangible systems use conventional UIs to implement them, studies of end users focus on the novel interaction modes.

When considering end-user programming, we must pay attention to deixis. Whereas tangible interaction is an especially direct type of direct manipulation, programming is fundamentally linguistic, rather than physical. When we write programs, we are making statements about the things we want to happen in the future. A key principle of direct manipulation is that the effects of one's actions should be immediately visible. In programming, by contrast, the effects of one's actions will occur in the future, once the program is executed.

One approach to programming in the context of tangible interfaces has been the creation of tangible programming languages [6]. The design intuition is that, if the GUI is more concrete and intuitive than the mathematical abstractions of programming, so 3D tangible UIs (TUIs) will be even more concrete and intuitive. Although it is not immediately apparent how physical objects can be a "language", there is an appealing intuition that programming by assembling software components is like assembling Lego bricks. Many tangible programming languages have been created on the basis of this intuition.

Tangible programming languages
One example of a TUI that is particularly relevant to end-user programming of IoT is the Media Cubes language for scripting interaction between networked devices in a "smart home" automation scenario [3]. The Media Cubes language was motivated by analysis of domestic remote controls, refactoring them so that each control had only a single button. Some Cubes represented state change events (e.g. play/pause), while others represented media content or indexes (e.g. a video stream). Users could specify relations between the Cubes and a range of media devices by placing them in proximity while pressing a button. Scripts were built up by composing these associations. However, using physical objects both for deixis and for language syntax introduced problems.
In tangible programming languages such as Media Cubes, individual cubes function as a reference (after being associated with an appliance) and also as a relation (such as a media stream, index or event). Tangible representation of relations was valuable, because it allowed relations to be composed (for example, specifying that the occurrence of an event should play or pause a media stream). However, this emphasis on tangible representation of abstract entities removed the directness of interaction with the appliances themselves.

A further problem of Media Cubes, often commented on at the time, was that tangible programming languages represented an extreme position in Cognitive Dimensions. Although placing cubes together and pressing a button could naturally compose scripts, the permanent record of that action was not visible, because the configuration of cubes would change as soon as the user put them away. When regarded as a language notation, physical objects have many drawbacks, as analysed in detail by Edge [6], but also anticipated much earlier by the satirist Jonathan Swift in Gulliver's Travels, where the philosophers of Lagado carry baskets of physical objects in order to speak in a universal language without any ambiguity of interpretation.

Integrating direct manipulation with deictic language
As an alternative to the creation of unimodal programming languages, whether tangible, graphical or textual, multimodal languages exploit the relative advantages of different modes in a way that is analogous to deixis in natural language. Just as the introductory example of deixis involved both verbal (speech) and physical (finger) components, so multimodal languages can use a mouse pointer alongside textual or spoken interaction. The classic example of deixis in multimodal interaction is the Put That There system [4]. Aghaee's NaturalMash [1] provides a powerful and intuitive paradigm for web mashups, allowing the user directly to demonstrate actions within a browser, and then compose these actions into a script through use of a constrained natural language in a textual dialogue window.

In our current research, we are extending the interaction paradigm of NaturalMash to IoT, by incorporating real world interactions with internet-enabled devices into the repertoire of services that can be mashed up. However, whereas NaturalMash employs a form of deixis through actions in a (GUI) browser window, interaction with an Internet appliance takes place in the 3D physical world. It is possible to describe these actions indirectly, for example using the onscreen controls of a remote camera app to steer and zoom the camera. This extension is relatively straightforward, and offers an immediately accessible scripting extension to many typical IoT devices. But the argument presented in this paper suggests a more ambitious approach to the use of deixis, integrating the tangible affordances of the physical device with the potential for scripting.

Augmented reality as deixis
We have created a simple mobile augmented reality app that can be used to identify, refer to, and manipulate abstract functions that are associated with physical appliances. In a sense, mobile augmented reality always represents a form of deixis - the phone camera becomes a pointing device, specifying a context to capture and manipulate the context of the physical world.
within which some type of linguistic query or command might be interpreted [8]. Most such systems represent enhanced forms of direct manipulation, but we are interested in turning them into a form of programming language. A programming language can be considered as an abstract layer superimposed on the direct manipulation of data. In the same way, augmented reality can be considered as an imaginary world of interpretation superimposed on the physical environment [2].

In our augmented reality IoT programming interface, this allows the user to distinguish between an immediate “direct manipulation” action in the physical world (e.g. pressing a light switch) and reference to that action for the purpose of incorporating it into a program (e.g. referring to the use of that light switch at a time in the future).

In this paper we have set out a theoretical account of multimodal deictic interaction with digitally-enabled physical devices, illustrated with many examples of past and future systems that demonstrate the advantages and disadvantages of alternative approaches. We believe that this account extends beyond a simple set of design rules, and represents a philosophy of end-user programming for the Internet of Things. The Internet is fundamentally a (very large) information structure. Although realised through engineering infrastructure, information structures are linguistic artefacts. The Internet of Things, however, is not solely linguistic, because physical “things” have an objective reality and embodied context that persist beyond the information they carry. In natural language, the correspondence between information structures and physical contexts is maintained through deixis. However we choose to establish such correspondences for the Internet of Things, the result must also be deictic.

References
Mashing-up smart things: a meta-design approach

Giuseppe Desolda
Università di Bari
via Orabona, 4
70125 Bari, Italy
giuseppe.desolda@uniba.it

Carmelo Ardito
Università di Bari,
via Orabona, 4
70125 Bari, Italy
carmelo.ardito@uniba.it

Maristella Matera
DEI – Politecnico di Milano
Piazza Leonardo da Vinci, 32
20134 – Milano, Italy
matera@elet.polimi.it

Antonio Piccinno
Università di Bari
via Orabona, 4
70125 Bari, Italy
antonio.piccinno@uniba.it

Abstract
Recent technology advances support the interconnection of smart things, enabling their communication according to the Internet of Things (IoT) paradigm. This paper discusses how the opportunities offered by such technologies can be amplified, by offering to the end users the possibility to define the behavior of their smart objects according to paradigms that capitalize on well-established approaches for mashup composition.

Author Keywords
Internet of Things, meta-design, mashups.

General Terms
Design, Human Factors.

Introduction
The current integrated technologies confer intelligence to any type of objects (also called things in this document) and connect them to the network. For example, a pressure sensor can make an office door handle smart, for example alerting a user located in another part of the world every time someone comes into that office. The added value is that the object, being part of a network of smart things (the Internet of Things (IoT)) can communicate with other objects or services, thus triggering dynamic behavior. If the door sensor is able to communicate with a sensor worn by a person, than a remote user can also know who entered the office.

One of the opportunities of the IoT is especially the possibility to collect in real time information concerning events and behaviors happening in the real world. For
example, a potential advantage of the IoT is to anticipate
the needs of a human even before he is aware of it.

Today these scenarios may appear unrealistic since we are
not pervaded with a significant number of sensors. However,
it is estimated that within 10 years the number of sensors
will vary between 20 and 50 billion. There are already many
contexts in which the IoT is adopted. For example, for
creating wearable devices, i.e. clothing and accessories such
as bracelets, watches, T-shirts, rings, shoes, which integrate
sensors capable of detecting physiological parameters.

Domotics is another very active IoT context: pressure,
volumetric and distance sensors, as well as infrared cameras
are able to ensure the security of homes; they permit a
remote user to interact with the devices installed in the
house (e.g., lights, heating, blinds) and assist user to avoid
hazards such as floods, fires or explosions. Other sectors
that are benefiting of the advent of IoT are Smart Cities,
industry and environment through energy saving.

**IoT Challenges**

From the technological point of view, such a large number of
things requires an adequate network infrastructure and
efficient communication protocols, especially because
integrated devices have very limited resources (e.g., CPU,
RAM, memory, battery). Further issues, such as privacy
(e.g., in the smart door scenario, if and when it is allowed to
know who entered into the office) and safety (e.g., if a
hacker is able to break a device, then it could access all the
devices it is connected with) must be addressed.

From the HCI point of view, a major challenge is to enable
even non-technical users to manipulate data and
functionality of things in a simple and natural way. Today, in
fact, this is a prerogative almost always reserved to
developers who, through the use of specific programming
languages, provide pre-packaged solutions to users. The
most important challenge is to allow non-technical users to
define and manage the connections between things, which
represent the real benefit of IoT.

Some works in the literature propose mashup techniques for
addressing this issue. For example In [2] and [3], the
authors introduce two systems for the mashup of things for
home automation, both consisting of two design
environments: one is devoted to electrical engineers who
define the behavior of devices through a visual
representation of logic operations and algebraic formulas;
the second one allows non-technical users to create a web
page where they can include widgets to display data coming
from things and synchronize their behavior based on a
"pipes-like" composition paradigm. The problem is however
that several studies have demonstrated that this kind of
composition paradigm is not suitable for non-expert users
[5-7] as it forces them to deal with concepts like data flow
and parameter passing which cannot be mastered by people
who are not expert in programming. One of the few products
available in the market is gluethings
([http://www.gluethings.com](http://www.gluethings.com)), a Web platform for registering
and composing things. Unfortunately, also this system offer
a pipes-based composition paradigm and also require users
to handle json to set parameters for the low-level behavior
of things – such practices are out of reach for laypeople, i.e.,
those users which represent the actual business opportunity.

To determine the success of the IoT, it is necessary to
investigate new approaches that, thanks to high-level
abstractions, can enable non-expert users to compose data
and functionality of things, as well as the communication
among them, by means of “natural” composition paradigm.
Our position is that a meta-design approach can be adopted for the mashup of smart things in a similar way as we already proposed in [1] for the mashup of heterogeneous data sources. Smart things in the end produce data (for example parameters revealing the occurrence of given events). Considering the objects as sources of data is therefore reasonable. Moreover, composition approach for service mashups is based on an event-driven, publish-subscribe paradigms that suits very well the domain of smart objects whose behavior has to be synchronized. We identify the configuration of things as a critical task, at least unless standard technologies and communication protocols are established. In this respect meta-design offer some advantages.

A meta-design approach

The meta-design approach we propose is characterized by three different layers, related to three distinct design phases offered to professional developers, domain experts and end users (see Figure 1). At the top layer, developers design and develop the design environments that enable the design by the end users.

At the middle layer, domain experts with the help of developers customize the platform to a specific domain. In the IoT context this implies performing the following activities:

- **Set of things registration**: sensors and actuators of a specific domain are registered. For example, in the case of home automation cameras, volumetric sensors, anti-burglary sensors, actuators to control the electric shutters, actuators to open and close the gas, etc;

- **Set of things configuration**: specification of which data and functionality are relevant for a specific domain. This activity may concern data sampling frequency definition (the data from the same sensor may be sampled in different ways depending on the context of use) or the definition of interfaces (a thing could be equipped with several sensors and actuators, and different users might have access only to a subset of them; for example a webcam installed in the center of a city can provide citizens with data streaming, while the administrator can also turn it on and off).

- **Visual template definition**: specification of the “visual containers” to be adopted by the end users to display and manipulate data extracted from things (such as charts, lists, maps). The provision of visual templates is a characterizing feature of our approach, as we propose composition paradigms where the customized visual templates provide the abstraction with respect to technical details of service composition practice, which is also adequate with respect to the end users background and skills.

- **Composition style customization**: according to domain elements, such as users’ skills and typical tasks performed with things, the overall composition style is also customized. Also, subsets of available compositions are identified and provided, in order to avoid the overloading the design environment.

At the bottom layer, the final users are provided with an environment that permits not only to include and interact with predefined compositions of things (as in [2, 3]), but also to create new compositions. The user is able to define
select data and functions exposed by sensors/actuators, and define how things have communicate among them.

**Outlook on the composition paradigm**

As mentioned above, things can be treated as services, because sensors and actuators have an URI that identify them on the Internet. In the case of service mashups, platforms implementing an event-driven approach, as the one described in [4], permit to synchronize Web services so that the event produced by/on a service (e.g., selection of a word in a text) triggers an action of another service (e.g., a search using the selected word). We believe that an event-driven paradigm is suitable also for the manipulation of things offering the advantage of enabling the composition of things among themselves and also with other Web services. Similar event-driven platforms (e.g., IFTT - https://ifttt.com/) are now emerging also for **task automation**. They make it easy to connect two services (things or APIs), choose a trigger, and thus create an action. They look very promising thanks to the simple and effective composition paradigm. Our platform will also take advantage of such a simplicity deriving from the event-driven approach, which we already implemented for synchronizing APIs at the UI level. Additionally, we will take advantage of the meta-design methodology and of the intrinsic customizability of the platform. Our ultimate goal is to define a natural composition paradigms fitting the background and skills of the end users.

**Conclusion**

One of the cornerstones of the future of the IoT will be to put in the hands of the end users simple software tools capable of making natural and powerful composition between things. It is unimaginable that this possibility is reserved for a few experts. As already happened in the past with software tools (forums, social networks, CMS) that made users to evolve from simple consumer to prosumer, even for the IoT have to be designed tools suitable even for non-expert users. This paper has illustrated how a generic platform for service mashups can be specialized for the composition of services that enable accessing/controlling smart things. We are currently working on implementing the needed extensions. A demo, illustrating some preliminary results, will be given during the workshop.

**References**


Towards a Toolkit for the Rapid Creation and Programming of Smart Environments

Thomas Kubitza
University of Stuttgart
Pfaffenwaldring 5a
70569 Stuttgart, Germany
thomas.kubitza@vis.uni-stuttgart.de

Abstract
“Smart” environments rely on the interconnection of various devices that are equipped with sensors and actuators and are statically or dynamically deployed in rooms or buildings or worn by users. Although it has become much easier to build and program single device installations with platforms such as Arduino, it still remains a challenging task to build whole environments with heterogeneous interconnected devices. A lot of this effort is due to the implementation of similar functionality in different programming languages for different device platforms and the bridging of the different communication technologies, protocols and formats between devices. The author believes that the right toolkit can widely cover this technical complexity so that designers and users of a smart environment can focus on the interaction design and the programming of intelligent and useful behavior. Such a toolkit is currently being developed within the meSch EU project and is presented and demonstrated by the author.

Author Keywords
Smart Environments, End User Programming, Mashups, Rapid Prototyping
Smart and interactive spaces are based on a common principle; different kinds of devices with sensors and actuators attached are statically installed in rooms, levels, whole buildings or are even worn by users. All these heterogeneous devices need to talk to each other or to an entity that constantly combines system state and generates system reactions. Multiple reasons make the setup of such environments a complex task. Firstly, similar functionality has to be implemented on various platforms ranging from microcontrollers to High-End computers. This requires expert knowledge in very specific programming languages and platforms as well as the management of various development environments (IDEs) and compilation tool chains. Secondly, different communications technologies, protocols and formats have to be bridged so that devices can actually exchange data. Thirdly, devices have to be deployed in their target environment, supplied with electricity and wired or wireless communication infrastructure (e.g. WiFi access points). Especially the first two reasons put up high boundary for non-experts in electronics and programming. This limits its usage to small and mostly only professional user groups.

We believe that the right tools can open up the creation of smart environments to a much larger audience in the same way as physical prototyping platforms such as Arduino made the access to microcontrollers much easier and in the same way as Apps made potentially everyone the programmer of his own cell phone (and the phones of millions of others). By empowering groups such as user experience designers, scientist, designers, artists, makers and hobbyists we envision the creation of a large set of truly useful applications evaluated in realistic environments and addressing a broad range of problems.

In this workshop-paper a toolkit is presented that drastically reduces the technical complexity for creating and programming smart environments. Our approach consists of two main pillars: (1) A client software for each type of end device is provided which allows to remotely access and control all its abilities and to abstract from its specific platform. (2) A server software on a central computer node is provided to bridge between various communication-technologies and to provide unified access to all sensors and actuators of configured devices through a web based JavaScript development environment. This approach allows to quickly implement or change the behaviour of a system without the need to reprogram or physically access any of the associated or deployed devices. JavaScript, one of the most widespread and growing programming languages, is used to define the system behaviour in a single spot.

**System Concept**

Conceptually our toolkit strictly follows a master-slave architecture; the client side software allows to treat each client device as a source of sensor events, a sink for actuation commands, or both. It allows abstracting from the underlying operation system and hardware as well as the communication technology and protocol. For each end device platform a specific client firmware has to be implemented. However, this implementation effort is done just once by experts for this specific platform; after that, users of our toolkit just need to install the firmware once and from that point on they can access and control all its abilities from the central server node.
The central server node provides a web based user interface for the configuration of devices and the creation of behaviour rules. It includes a rule engine that triggers events and runs rules as soon as sensor data is received. Rules may then again trigger actuator commands which are instantly sent to the appropriate devices.

**User Interface**

The toolkit user interface runs in any browser and is structured in three sections: Devices, Events and Rules. New devices automatically appear in the device overview after the installation of their specific client software. In its initial state a device has no modules configured; this means no sensor or actuators are activated yet. In the device configuration view, modules can be selected from a list of supported sensors and actuator modules (figure 3a). By double-clicking on a module, it is automatically assigned to a compatible and free port; this way users don’t need to care about the right attachment of sensors beforehand, the device configuration steps already gives visual hints about the right ports to connect to. After saving a configuration, it is instantly pushed to the device where the selected modules are activated. Client software for a wide range of physical prototyping and mobile platforms is already available.

In the the rules-view an overview of available rules is given. These are structured in groups and consist of a name, a description and an execution priority. Groups allow to tie together rules which are logically associated or only apply to a certain space in the environment (e.g. “Office Floor 1”). Groups and rules can be enabled and disabled individually which allows easy instant switching of behaviour for single spaces or whole environments (figure 3b). Single rules can be edited any time and the changes are applied immediately. Syntax highlighting and auto-completion features help novice users to shimmy along available devices, modules and their individual properties without any previous knowledge (figure 3c); advanced users benefit from the coding speedup and correct referencing of objects. Multiple application setups in the cultural heritage domain (see figures 1 and 2) as well as in office automatisatin have been already realized using
our toolkit – more details on these will be given during the workshop.

**Demonstration during Workshop**
The author will bring all necessary components to the workshop so that multiple groups can independently get hands-on experience with the toolkit. The components will include two smartphones, two tablets, two WiFi projectors, two .NET Gadgeteer kits, two Arduino kits, multiple RPIs and Intel Edisons, various sensors and actuators as well as multiple Bluetooth Low Energy beacons for proximity experiments. Participants will be able to program all these devices through a web based IDE which they can run in the browser of their own PC or tablet. Further, devices can be integrated that are brought by participants (e.g. Android smartphones, fitness trackers, etc.). Quick and simple mashups can be demonstrated as well as more advanced setups that integrate a larger set of heterogeneous devices and more complex logic.

**Conclusion and Future Work**
The toolkit introduced in this workshop-paper aims to drastically reduce the technical effort for setting up a smart environment and to provide a single interface in which its behavior can be programmed using a popular web programming language. We envision this toolkit to be a platform for more advanced research on the programming of smart environments and interactive spaces: We are planning to extend the toolkit with various additional layers that lie on top of the Javascript interpreter and target users with different expertise, e.g. configurable behavior scripts, visual programming, programming by natural language. Further, we are exploring new ways of programming environments apart from writing code on a single desktop PC, e.g. automatic code creation by demonstration as well as proximity based code generation and assistance. We further want to use the power of the community and are therefore developing means that allow easy sharing and integrating single behavior rules or full setups with a single click. The author believes that the presentation and hands-on experience with the toolkit can lead to an active discussion and extremely valuable feedback from the EUD community. This will be an important input into the further development of the toolkit and related projects.
Learning about End-User Development for Smart Homes by “Eating Our Own Dog Food”

Abstract
SPOK is an End-User Development Environment that permits people to monitor, control, and configure smart home services and devices. SPOK has been deployed for more than 4 months in the homes of 5 project team members for testing and refinement, prior to longitudinal experiments in the homes of families not involved in the project. This article reports on the lessons learned in this initial deployment.

Author Keywords
End-user programming; end-user development; connected home; smart home; ubiquitous computing.

ACM Classification Keywords

Introduction
The “Do-It-Yourself” approach to configuring and controlling domestic technology has become increasingly popular. End-User Development Environments (EUDE) have been developed to support this approach. While the Scratch-based programming language [3] used in the ZipaBox and the rule-based IFTTT propose attractive graphical syntax and stylistics,
These environments provide limited debugging aids for non-specialists. At the same time, seminal work from academic research such as Jigsaw [2], CAMP [4], and iCAP [1], has not gone beyond proof of concept. Thus, over the last two years, we have developed SPOK (Simple ProGramming Kit), an EUDE for smart homes. Our goal is to provide a robust, extensible and flexible system that can be used effectively in the home, and that can be evaluated through longitudinal experiments in real life conditions.

In the next section, we explain our technical choice for the baseline middleware for SPOK followed by a presentation of the principal features. In the final section, we summarize the key lessons learned from the initial deployment of SPOK in our own homes.

**Beyond Technical Proof-of-Concept**

Implementing a EUDE for real world smart homes requires choosing the "appropriate" run time infrastructure from a jungle of middleware. From our experience with the development of an earlier environment KISS [5], "appropriate", in a research context, means: (1) license free and robust, (2) support for dynamic discovery, (re)composition and deployment without human intervention, and (3) low entry cost for developers.

OSGi satisfies the two first requirements but is too low-level for non-system developers. OpenHAB [14], which has been chosen by Eclipse as the Eclipse SmartHome, does not, in its current form, support dynamicity. HomeOS is by construction a .net environment, therefore not compatible with iOS and Android platforms. In the absence of a de facto standard middleware for smart homes, we have used ApAM.

ApAM (Application Abstract Machine) is a component-oriented middleware that extends OSGi/iPOJO in two ways: (1) developers describe an application by intention using a dedicated language as opposed to explicitly specifying composition of components and bindings at design time; (2) from the abstract description of the application architecture, a concrete architecture is computed and incrementally updated by resolving the dependencies between the components currently available in the execution environment.

Due to the incremental and dynamic (just-in-time) construction and maintenance made possible by ApAM, SPOK is resilient to the opportunistic installation and disappearance of devices and services. The sidebar shows the overall description of the global architecture of SPOK.

**The Description of SPOK**

SPOK provides the end-user with the following services: (1) A syntax-oriented program editor that enforces the construction of syntactically-correct programs (see sidebar on next page). (2) A program interpreter and a clock simulator to test program execution in "simulated time". (3) Debugging aids to support the detection and correction of programming errors or system malfunctions along with a Trace Manager. (4) A dashboard to remotely control devices and programs in a centralized and uniform manner.

Compared to the state-of-the-art, the key features of SPOK are three-fold: Expressive power of the SPOK language along with a pseudo-natural concrete syntax, dynamic adaptation to the arrival/departure of devices and services, and debugging aids.
Expressive power of the SPOK language

The language supports a mix of imperative and rule-based programming paradigms. For example, the Start-Stop-XmasTree program shown in sidebar, is comprised of an imperative section composed of actions Switch off and Blink, and 2 rules. Conditions can be expressed both in terms of states (e.g., if the blue-lamp is on) and events (e.g., each time the blue-lamp is turned on). Home entities can be denoted by using properties and relations (e.g., all lamps located in bedroom).

Parallelism is supported at multiple grains: several programs can be started simultaneously; within a program, several rules can fire simultaneously and a program can start/stop the execution of another one.

Dynamic adaptation

The grammar of the SPOK language is dynamically updated according to the set of services and devices that are currently available in the home. As a result, the Smart Keyboard, which guides end-users in entering program elements, shows options that are both syntactically valid for the current insertion point, and compliant with the current state of the house (see example in sidebar). Similarly, programs that reference devices and services that are no longer available are flagged so that end-users are made aware that running these programs may result in unexpected behavior. In the example of the Start-Stop-XmasTree shown at the top of the sidebar, the triangle at the top left of the figure indicates that the program is running. The lamp referenced by the Blink action has disappeared from the execution environment. Consequently, its reference has been changed to Unknown and the triangle has turned from green to orange to indicate that execution continues but at our own risk.

Debugging aids

Debugging aids come in three complementary forms: history (by the way of time lines), dependency graph (see sidebar next page), and execution indicators in the source code of SPOK programs. For example, the imperative section of Start-Stop-XmasTree has been executed (counters are set to 1) while no rule has been fired so far (counters are equal to 0) and all of them are waiting for their condition to become true (arrows and light blue background indicate waiting points).

Lessons from Using SPOK in Our Own Homes

SPOK has been deployed for a period of 4 months in 5 distinct homes of project team members. The intent was to test the robustness and usability of the system while refining the system on a weekly basis. Beyond typical bugs, the main findings are the following: improvements for the preparation of field experiments, discovery of new concepts and key issues for future research, and confirmation of findings about our own behavior consistent with results from the literature. (This last issue will not be developed any further here.)

Improvements for field experiments

Harnessing the hardware. Wireless sensors and devices are sensitive to physical conditions (e.g., out of range, lack of power). This sensitivity is generally not "visible". For example, sensors powered with solar cells will fail after a few days without sufficient light. Smart plugs, when installed too far from their dongle, appear and disappear in an unpredictable manner. It is necessary to learn how to build relays between them. The same holds for Hue lamps when blocked by concrete walls. For wireless switches, clicking requires energetic press to generate events. The cover of the DomiCube must be opened to access the led that indicates that it is too
far from its Bluetooth dongle. All these details cannot be discovered in the lab. Work-arounds to such problems must be documented and explained to subjects.

**Taming the technology.** As St-Exupéry wrote for the fox: “If you tame me, then we shall need each other.” Indeed, in the first week after installation, it was common to not find any interesting uses for the system. However the system progressively became an integral part of the home as experience was gained. For example, we have discovered that the hue lamps can provide a tangible representation for temperature and energy consumption. We opportunistically wrote a program to control the lighting of the Christmas tree (as we repeatedly forgot to stop it manually before going to bed). Our conclusion is that the system must be installed in a subject’s home for a minimum of one month to report interesting results. In addition, our programs, which emerged from real life needs, can serve as examples for a forum opened to the subjects.

**Discovery for future research and improvement**

Privacy issues made real. By analyzing the time lines provided by SPOK, it was striking to discover how much the rhythm of daily life can be discovered from data recorded by the Trace Manager: movements in the home, arrival and departure, meal and bed times, etc.

**Some devices are more critical than others.** In the design of SPOK, we missed the notion of ”critical” device. This became clear when we used a smart plug to measure the consumption of our refrigerator, while absent for two days. Such a plug must not fail and cut electricity to the refrigerator – as could possibly through program or user. Similarly, some devices require access control (e.g., TV for kids late at night).

**Conclusion**

We have two take-away messages. First, longitudinal experiments of a EUDE like SPOK in real-world settings require a middleware that reliably supports dynamic software adaptation and automatic deployment. Secondly, an initial deployment using “our own dog food” in homes of project team members provides highly valuable information for tuning the protocols before the start of field studies, thus improving the quality of the evaluation itself while saving time and discovery for future research.

**Acknowledgements**

This work is supported by the European AppsGate CA110 project. We wish to thank our collaborators: R. Barraquand, M. Bidois, S. Caffiau, J.R. Courtois, A. Demeure, J. Estublier, T. Flury, C. Gérard, C. Lenoir, J. Nascimento, K. Pethoukov, C. Roux, G. Vega.

**References**


User Conceptual Models of Event-Action IoT Applications

Paternò Fabio & Gabriella Lucci
CNR-ISTI, HIIS Laboratory
Via Moruzzi 1
56124 Pisa Italy
fabio.paterno@isti.cnr.it

Abstract
In this position paper we discuss why the event action model is important for IoT applications. We report on an analysis about how it is currently supported in some Android apps, the usability and expressiveness of such solutions, and provide indications for the design of new EUD environments for IoT applications able to support it through smartphones.

Author Keywords

ACM Classification Keywords
H.5 Information Interfaces and Presentation; H.5.2 User Interfaces

Introduction
In recent years we have witnessed the use of computers in an increasing number of dynamic contexts, with a huge variety of users having radically different backgrounds and a plethora of diverse tasks to perform. On the one hand, such users share a common requirement for software, to support their common tasks that may vary rapidly, with some of them that cannot be anticipated at design time but discovered only during actual use. On the other hand, slow software development cycles and lack of domain knowledge by software developers are limitations to addressing the requirements of different users. End-User Development (EUD) [1] can help to mitigate this gap. However, another emerging trend is the Internet of Things (IoT) era [2], where ‘smart’ physical objects are thought as networked together, able to interact and communicate with each other, with human beings and/or with the environment to exchange data and information ‘sensed’ about the environment, reacting autonomously to events in the real/physical world, and influencing it by running processes that trigger actions and perform services. According to Gartner1, there will

1 http://www.gartner.com/newsroom/id/2636073
be nearly 26 billion devices on the Internet of Things by 2020: in this scenario IoT applications need to address deeply contextualized user needs.

So far main EUD approaches \[3\] have mainly considered the desktop platform and applications that are unable to control smart objects, and adapt to the changing context of use \[4\]: desktop spreadsheets have been the most used EUD tools so far. Thus, there is a need for a new generation of EUD approaches able to address IoT ubiquitous applications \[5\] and to offer unprecedented opportunities to achieve deeper, more meaningful and faster personalization by putting the user at the centre of informative systems, ambient and personal sensors, communicative tools, and mobile and ubiquitous computing devices.

In particular, our aim is to reach a better understanding of the user mental models when they want to specify how the interactive application should behave according to the objects available in the context of use. A first contribution in this direction \[6\] has only considered some aspects related to practical trigger-action programming in the smart home by using the IFTTT (“If This Then That”) Web environment. Such environment has limited possibilities, since it only supports applications with only one trigger and one action. Thus, we need environment able to allow users to specify more flexible behaviours that can be triggered by various combination of events, and activate multiple effects.

An Analysis of Three Apps for EUD in Smartphones

We started our work with an analysis of how three Android Apps (Tasker\(^2\), Locale\(^3\), and Atooma\(^4\) ) aim to support non-professional developers to create context-dependent applications by exploiting the smartphones’ sensors and capabilities \[7\]. They provide three different solutions according to the event / condition / action model. They have similar structures: categories, elements, actions. We have conducted an analysis of the three environments from two viewpoints: expressiveness (to what extent they support the relevant concepts); usability (for which a user study has been carried out).

They use slightly different vocabularies for the same concepts. In Atooma an application is called Atooma and is structured in an IF and a DO part. Locale supports the development of situations described in terms of Conditions and Settings. Tasker is used to create Profiles structured into Contexts and associated Tasks composed of Actions. In terms of the development process, Atooma is completely sequential: developers have first to indicate the conditions and then the actions. In Locale it is not sequential and developers can freely choose to specify situations and settings in any order. Tasker is semi-sequential in the sense that the starting point can be either the condition or the action, but if a condition is specified then the corresponding actions must be indicated. The three environments differ in terms of how they implement

---

\(^2\) http://tasker.dinglisch.net
\(^3\) http://www.twofortyfouram.com
the events and actions model. Right at the beginning Atooma asks users to select mainly from four main macrocategories, Locale provides a list of elements, which can be extended through plugins, while Tasker structures the selectable events and conditions in terms of six Contexts.

Tasker has the greatest expressiveness (more than double Locale’s), with a number of actions that can be expressed (108) greater than the triggers (83). In Atooma the number of expressible conditions (70) is greater than the actions (48). In both triggers and actions, Locale has the same number of expressible elements (40) and is the one that has the lowest total expressiveness. On a total of 80 features, 58 are obtained through plugins since few elements are directly integrated into the environment.

In the user test, two tasks were accomplished in ascending difficulty order for each app. The first was to specify an adaptation rule composed of one event and one action, the second included an event, a condition and two actions. In the first task, with Locale the mean time of the first task execution (1'29") was about half than Atooma (2'52") and Tasker (3'50"><") was almost a minute longer than Atooma . In the second task, the application that required less average execution time was again Locale (3'35") followed by Atooma (4'00") and Tasker (5'14").

To summarize, the most expressive environment (Tasker) is also the one that was found most difficult to use (as demonstrated by the highest performance time, error numbers, and unsuccessful performance numbers).

Card Sorting for Identifying User Conceptual Mode

In order to better understanding how users classify the concepts that characterize context-dependent applications we carried out a user study by using card sorting and associated cluster analysis techniques. The identification of the cards derived from the analysis the three apps. We proposed cards able to indicate all the events and actions that were supported by the three apps. In the end we obtained 39 cards: 14 referring to only events, 6 to actions, and 19 were used for both events and actions. The card sorting was proposed to 18 users. At the beginning we provided them with some basic concepts to introduce context-dependent applications, then the users had to group logically them and assign a name to each group identified. They had to carry out the exercise twice: once to classify the 33 cards related to events and once for the 25 cards representing the possible actions. During the exercise the groups identified by the users were entered in the UXsort tool that has been used to support the results analysis. By applying hierarchical clustering methods the tool is able to measure the linkage among elements groups and produces a dendrogram that represents the similarity among elements through a tree-like structure. The supports their analysis by using three clustering algorithms (single linkage, complete linkage, average linkage).

In order to select the most interesting results we decided to focus on solutions that offer a number of groups between 5 and 8. Such numbers were identified by the analysis of the numbers of groups supported by current solutions, and also considering that with less than five groups we can obtain groups containing
heterogeneous elements, and with more than 8 groups the solutions tend to separate elements that people would expect together. This generated 12 solutions for grouping the events and 12 for the actions.

By observing the pattern elements in the groupings occurring both in the events and actions classification we have identified the associations between elements and groups that users found more meaningful, and the corresponding group names that were assigned more frequently occurring. We also discarded solutions with an unbalanced number of elements in the resulting groups or with a group name not completely consistent with the actual elements (e.g. archive for a group containing the media-player element).

In the end, we found a solutions with two small variants that differ for the collocation of the GPS element. In one variant, the GPS element was moved from the Connection to the Sensors group and this caused the creation of one further group for the actions. In the other variant, there is one further group (Localization) for both events and actions that contains the GPS and Location elements. A more detailed description of this study is reported in [8].

Conclusions and Future Work
We have presented a study aiming to identify how users classify most recurrent events and actions in IoT context-dependent applications. The study has also considered how such issue has been addressed in current solutions for Android devices.

Future work will be dedicated to designing a new environment for smartphone able to take into account user mental models and support the possibility of specifying adaptation rules through an expressive language, such as the AAL-DL (Advanced Adaptation Logic Description Language) [9] in a platform for context-dependent adaptation of IoT applications.

References
End-User Development in Internet of Things: We the People

Abstract
This position paper considers people aspects of end-user development in the Internet of Things.

Author Keywords
End-User Development, End-User Programming, End-User Software Engineering, Gender HCI, Internet of Things

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

Introduction
The Internet of Things is inherently about people—helping to improve their safety or their experiences in their homes, jobs, and in the world. For example, we can warn people of dangerous situations near their current location, recommend the next painting to look at in a museum, or enable them to remotely "program" objects in their smart homes to talk to each other.

My interest is in the "programming" aspect of this, specifically: (1) how we can enable ordinary users to customize, control, and "fix" Internet of Things applications that are trying to help them, and (2) doing so in ways that allow IoT systems to be inclusive to both males and females.

Margaret Burnett
Oregon State University
Corvallis, OR 97331 USA
burnett@eecs.oregonstate.edu

Todd Kulesza
Oregon State University
Corvallis, OR 97331 USA
kuleszto@eecs.oregonstate.edu

Paste the appropriate copyright/license statement here. ACM now supports three different publication options:
• ACM copyright: ACM holds the copyright on the work. This is the historical approach.
• License: The author(s) retain copyright, but ACM receives an exclusive publication license.
• Open Access: The author(s) wish to pay for the work to be open access. The additional fee must be paid to ACM.

This text field is large enough to hold the appropriate release statement assuming it is single-spaced in Verdana 7 point font. Please do not change the size of this text box.

Every submission will be assigned their own unique DOI string to be included here.
Meet “Grandma”
Imagine “Grandma”. Grandma has lived in the same small town and even the same house for her entire adult life. Even though she is advancing in years, she does not want to leave her home to live with relatives or in a nursing home.

Her granddaughter, Mary, would be happy to have Grandma come live with her in Los Angeles (about 100 miles from Grandma’s home), but Grandma will have nothing to do with this plan. She used to be a teacher, and her former students still recognize her when they run into her at the neighborhood grocery or library. She is very involved in her church, her neighborhood, and her community. In short, she feels like she is where she belongs, and moving would make her feel like she was giving up most of what she cares about.

Together, they decide on an aging-in-place solution, which uses IoT technology. Using this approach, smart technologies around Grandma’s home can alert Mary when Grandma’s activities seem abnormal, or when safety issues around the house arise, like water overflowing from a forgotten faucet, smoke, possible trespassers, and so on. Mary can even control some of the technologies remotely, like turning off the faucet.

End-User Customization, Control, and “Fixing” of IoT Mis-steps
However, smart technologies are never 100% correct, and some of the mistakes the system makes can be very costly to Mary, and even dangerous to Grandma. For example, Mary does not want a lot of false positives from the system, because this will mean a 100-mile drive one-way to Grandma’s, only to find out, for example, that the system’s warnings of intruders oftentimes are just branches blowing against the house. And Mary does not want false negatives, because she cannot risk leaving Grandma in a precarious or dangerous situation simply because the system thinks the situation matches patterns it has seen before.

Mary needs to fix such false positives and negatives, but how? Grandma’s data is in many ways unique to Grandma, so Mary cannot rely on other grandmas’ data to train the system better. She needs some amount of direct control.

We have been working on an approach to help Mary exert this kind of control. We call the approach “Explanatory Debugging” [5]. The approach aims to enable end users like Mary to efficiently influence the predictions that machine learning systems make on their behalf, such as the system powering Grandma’s house. This paper presents an overview of Explanatory Debugging, an interactive machine learning approach in which the system explains to users like Mary how it made each of its predictions, and the user (Mary) then explains any necessary corrections back to the learning system.
The Explanatory Debugging approach presents explanations that people like Mary can then act upon. These explanations need to honor two sets of principles: a set of Explainability principles and a set of Correctibility principles:

- **Explainability:**
  - Principle 1.1: Be iterative, so that the user can learn in increments via many interactions.
  - Principle 1.2: Be sound; explain the system truthfully.
  - Principle 1.3: Be complete; explain how the entire system operates.
  - Principle 1.4: But don’t overwhelm; if the system is too complex to explain, perhaps a different machine-learning model would be more appropriate.

- **Correctibility:**
  - Principle 2.1: Be actionable.
  - Principle 2.2: Be reversible.
  - Principle 2.3: Always honor user feedback.
  - Principle 2.4: Incremental changes matter.

For example, Grandma does not like the sensors in the bathroom to be on when she is actually in there, so she covers them up with a swatch of dark material when she uses it. This causes the system to raise alarms, but Mary and Grandma talk it over, and Mary decides that as long as the bathroom sensors are not “dead” for more than 30 minutes, she will not worry about it.

She needs to explain this to the system in terms that will not work at cross-purposes to the system’s ways of reasoning; this is where Principles 1.2 – 1.4 are especially important. (If the system does not clearly explain how it is working, how can Mary understand how to correct its behavior?) Further, she needs to know what pieces of the explanation she should change (Principle 2.1), be able to tell whether her corrections are on the way to producing the right behaviors (Principles 1.1, 2.3, and 2.4), and be able to undo corrections that did not work out well (Principle 2.2).
One example of Principles 1.2 – 1.4 in a “baseball vs. hockey” text classifier is shown in Figure 1 [5].

**Gender Inclusiveness**

Mary is, obviously, a female. Will the emergent Internet of Things systems fit her style of working and problem-solving as well as they fit that of male adult caregivers with aging-in-place grandmothers?

Judging from the recent past, the answer is likely to be no. Recent research has shown that the way people use software often differs by gender, and further, that many software features are inadvertently designed around the way males tend to work and problem-solve with software [1, 2, 3, 4, 6, 7].

These findings show the importance of taking gender differences into account when designing software for solving problems, such as the software Mary is using in the example above. Fortunately, design remedies are emerging to overcome gender-inclusiveness issues, and such changes need not trade off one gender against the other. In fact, researchers have shown that taking gender differences into account in designing software features can benefit both genders. For example, Tan et al. showed that displaying optical flow cues benefited both females and males in virtual world navigation [7], and Grigoreanu et al. showed that changes to spreadsheet features relating to confidence and feature support reduced gender gaps while improving both genders’ attitudes and feature usage [3].

**Conclusion: Don’t Forget the People**

We hope that this new generation of software powering the Internet of Things will be as helpful to ordinary people as it is to corporate interests, engineers and tech-geeks, and equally inclusive to females and males.

**Acknowledgments**

This work was supported in part by NSF #1240957 and #1314384.

**References**


