Supporting the Design of Network-Spanning Applications

Stefan Rennick Egglestone
Mixed Reality Laboratory
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
sre@cs.nott.ac.uk

Andy Law
Edinburgh College of Art
Edinburgh, EH3 9DF, UK
a.law@eca.ac.uk

Andy Boucher
Interaction Research Studio
Department of Design
Goldsmiths College
London, SE14 6NW, UK
dts02ab@gold.ac.uk

Jan Humble
jan.c.humble@googlemail.com

Tom Rodden
Mixed Reality Laboratory
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
tar@cs.nott.ac.uk

Chris Greenhalgh
Mixed Reality Laboratory
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
cmg@cs.nott.ac.uk

CHI 2009, April 4–9, 2009, Boston, Massachusetts, USA
ACM 978-1-60558-247-4/09/04.

Abstract
In this case study, we describe our use of ECT, a tool intended to simplify the design and development of network-spanning applications. We have used ECT throughout the course of a two-year collaboration, which has involved individuals with expertise in a variety of fields, including interaction design and computer systems engineering. We describe our experiences with this tool, with a particular focus on its emerging role in helping us to structure our collaboration. We conclude by presenting lessons that we have learned, and by suggesting future directions for the development of tools to support the design of network-spanning applications.

Keywords
Interaction design, network-spanning applications, toolkit, component-orientation.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

Introduction
Recent years have seen a rapid increase in the capability of computational devices that are used in the home. In the developed world at least, many

residences now contain multiple personal computers, and many of these are either periodically or permanently connected to each other, and to the global internet, through wireless and broadband networking technologies. At the same time, a wider range of devices than ever before are capable of providing general-purpose computational facilities to users. Mobile phones are a prime example here. Once limited to just making calls and sending texts, devices such as Apple's iPhone are now network-enabled, end-user programmable devices in their own right. "Even my toaster has a microprocessor" comments Perlman [3], who argues that "computing is not just for computers any more". And as the technological infrastructure in our homes becomes ever smarter and better connected, the potential for creating applications that are both interesting and that *span* multiple homes, whilst making innovative uses of the devices and networks that are deployed in them, seems likely to increase. Examples of a number of these applications can be found in published literature (for an interesting example, see [4]). Furthermore, this is a growing research field, and as time progresses, it seems likely that new examples of these types of applications will emerge.

However, technological and organizational issues mean that the design and deployment of such *network-spanning* applications can be a difficult process. Even working with individual devices, especially those that are still not fully-featured computers in their own right, can require substantial expertise in electronics, software and hardware engineering. Integrating such devices into the home, and studying their use, requires a broader set of skills. Historically, design teams have addressed such difficulties through multi-disciplinary collaboration, often combining the skills of interaction designers with those of "engineers, who know about hardware, software and electronics" [3], and anthropologists, who know about studying the behaviour of humans "in the wild" [3]. Yet, faced with a complex design space, and with a rapidly-changing technology base, the assembly of such teams is not a "cure-all" for the creation of networking-spanning applications. Tools and methodologies are still required to support the work of these teams, and the design of these technologies is still an open research question. A number of authors have described tools that have been specifically designed to support the development of network-spanning applications, with many of these papers focusing on either technological aspects of the implementation of these tools, or on systems that have been constructed using them. However, a review of existing research papers has revealed that there is little published research that describes these tools in use. We believe that this is a significant omission, and that the study of the use of tools provides an important source of information for their future improvement. We are therefore attempting to make a contribution to this area of research by providing a case study of a particular tool in use. We hope that this will be of interest to both the design and computing communities.

More specifically, we present a case-study illustrating the adoption of an existing tool, known as the Equator Component Toolkit (ECT), into a multi-disciplinary design team which was composed of the authors of this paper. ECT is a tool which is designed to support a well-understood development methodology called component-orientation, an overview of which is provided below. Collectively, we have been using ECT

to design, implement and deploy a number of innovative network-spanning applications, through a collaboration involving individuals with a background in interaction design, industrial design, human-computer interaction and computer systems engineering. We used ECT throughout the course of our collaboration, which took place over a two-year period, and we made modifications to it where necessary.

Our aim in publishing this case-study is to provide specific information about the use of ECT in developing network-spanning applications, and at the same time to more generally address the utility of component-orientation as a supporting methodology in this context. We feel that both of these aspects have played an important role in the process of our collaboration, and want to publicly comment on our experiences for the benefit of others. Hopefully, through publishing both our positive and negative experiences in adopting this approach, we can contribute to a debate about determining the best methods for supporting the design of this emerging category of systems. Additionally, we hope to inform the next generation of tools that will be built to support this kind of design. The paper begins by introducing componentization as a concept. It then discusses its relevance to our particular design context. Existing component-orientated tools are reviewed, and the specific tool that we have chosen to use is then introduced. After presenting the core of our case study, the paper then concludes by discussing lessons that we have learnt, and future requirements for tools that we have identified.

Overview of componentization

Componentization as a methodology is built around a very old idea within the field of computing; that the efficient construction of computer systems should involve, as much as possible, pre-fabricated software and hardware components, much as modern-day civil engineering makes use of standard items with well-understood properties such as bolts or reinforced girders. In an early paper on this topic, Randell and Naur [2] suggest that certain organizations would focus on producing components, whilst others would focus on using them. He also introduces the idea of the component as a "black-box", in which users of components are not required to understand their inner working, but instead only the interface that they present. In another analogy with civil engineering, we might note that a user of a bolt does not have to understand the method by which it has been produced, but only the properties that such a method has created – for example, weight, hardness and durability.

Componentization as a methodology has found a substantial number of applications in computer software design and development, and is still in use today. As an example, consider the various *widgets* that are provided with modern programming languages, and which are intended to support the construction of graphical user interfaces. Widgets are general purpose software components that can be used to construct specific software interfaces, and examples include windows, pop-up menus and buttons. In component-orientated terms, one of the key points here is that such widgets have been designed and developed by a small number of expert users (who may work for large software houses such as Microsoft). The widgets are often then assembled into toolkits, which are constructed with the intention of supporting software development by a much larger (and often less-expert) group of individuals. Such use may

incorporate additional tools, such as graphical editors, which allow for the construction of interfaces by dragging and dropping widgets into a variety of interface devices such as forms. Such tools simplify the development of complex applications by less expert users.

Componentization in the construction of network-spanning applications

A prominent example of a component-orientated toolkit that has the capability of supporting network-spanning applications are Phidgets [5]. These are a family of sensing, display and actuation devices, which can be connected into a standard PC. Phidgets sit alongside a supporting software infrastructure, and are intended to be used as simple hardware building blocks, which can be integrated into more complex applications. Examples of Phidgets include small LCD displays (figure 1 below), interface kits (see figure 2) and RFID readers (see figure 3), with the interface kit being a device which allows for the attachment of a wide variety of different sensors. Each Phidget device is provided with a software component that is capable of connecting with, or controlling the device; such components can be either accessed directly (i.e. from code running on the same computer that the Phidget is plugged into), or over the network. Phidgets have been used in a wide variety of prototype and finished systems, with one example being SHIFD (http://www.shifd.com), a system which supports the transference of notes, addresses and links between mobile phones and computers. Phidgets are a commercial product, but have been widely used in academic research.

Figure 1 Phidget LCD

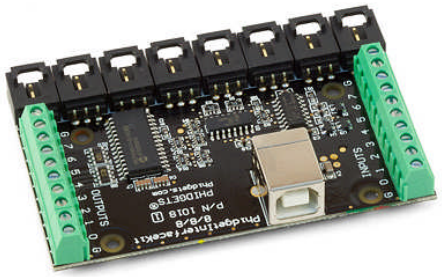

Figure 2 Phidget interface kit

Figure 3 Phidget RFID reader

A second example of a component-orientated system that supports the development of network-spanning applications is provided by iStuff, a product of the HCI research group at Stanford University. Like Phidgets, iStuff also consists of a set of devices and a controlling, component-orientated software infrastructure, but a difference between the two is that iStuff is more focused on providing access to devices for non-programmers.

In general, therefore, these previous efforts have demonstrated that component-orientation does have some relevance to the design of network-spanning applications, and many of the ideas generated by these efforts have been integrated into our collaboration. This case-study does not, therefore, provide a contribution by being the first to suggest this idea. Where we hope to provide a contribution is in providing a detailed study of the use of a component-orientated approach and toolkit in the course of a real design process. Through our work, we have developed a substantial body of expertise in making use of such a methodology. As researchers, we believe that studying the real use of technologies can provide substantial benefits to the future development of such technologies, and it is this that we focus on in this case-study. Firstly, however, we provide a brief overview of ECT, the tool that we have chosen to adopt in our collaboration.

Overview of ECT

Throughout our collaboration, we have chosen to make use of the Equator Component Toolkit (ECT), a pre-existing software solution which, at the start of our collaboration, was already in use by a number of groups worldwide. ECT seemed likely to suit our purposes, as it provides a set of components which can be used to control a wide variety of electronic devices, including the Phidgets which were introduced above. It also provides a graphical editor, which can be used to coordinate these components into more complex systems. Most importantly for us, however, ECT also provides facilities to support the construction of systems that span networks; it can be used to search for components hosted on any network-enabled device, to examine the functionality that they provide, and to request that they perform operations on behalf of the user. It is easy to use ECT, for example, to display information on small screens located around the home using information transmitted over a network; as long as a required set of display and networking hardware is available, such behaviour should work "out of the box". Figures 4 and 5 show screenshots of interfaces provided by ECT. Figure 4 shows the *capability browser*, used to search for components on networked-enabled devices, whilst figure 5 shows the *graph editor*, used to assemble components into systems. There is no space in this paper to provide a complete overview of the process by which systems are constructed in ECT; instead, examples presented in the case studies below will illustrate particularly important features (and the interested reader can refer to [1]). However, one further feature is important to note; this is an open system, into which new components can be added by individuals with sufficient expertise. This is a facility which we used substantially throughout our collaboration, as our understanding of the kinds of systems and components that we wanted to work with progressed. However, we made a number of modifications to increase the flexibility of ECT in this respect, which are outlined in the first section of our case study.

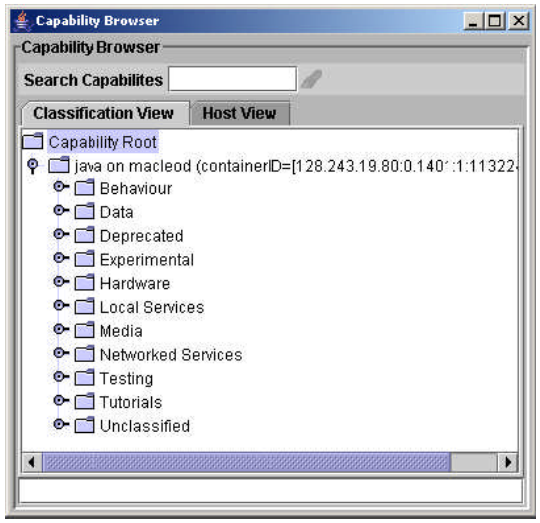

Figure 4 Capability browser

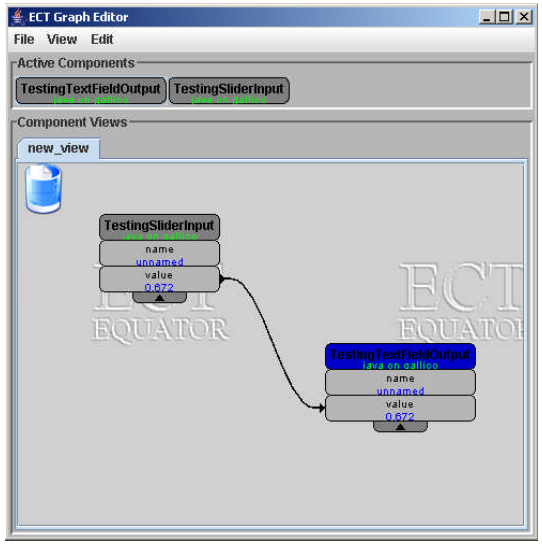

Figure 5 Graph editor

Studying our collaboration

We now move onto the core of our case study, which focuses on a broadly chronological description of our experiences in using ECT as part of a component-orientated methodology to develop network-spanning applications. We have selected three episodes from our collaboration, each of which we describe in detail, and each of which illustrates an important stage in our design process. Throughout our collaboration, we have made modifications to this tool, informed by our design practice, and such modifications have then been used in later design stages. Our exploration therefore sits on the boundary between the design and technical development process, and examines how both have shaped each other during the process of our collaboration. It should be noted that, in developing this account of our experiences, we have made use of a variety of sources of information, including our archives of email records, sketchbooks, progress reports and the debriefing of the designers and developers involved. These sources of information have provided us with a detailed record of our collaboration over the two years of the project, and we hope that they have allowed us to accurately report on our experiences in this period.

Episode one – prototyping of an ambient communication channel between two homes

Our earliest experiments with designing and developing network-spanning systems focused on the use of ECT in the domestic environment. In order to gain experience with the various technologies we wished to use, we decided to use the existing components and facilities provided by ECT to quickly put together an installation that facilitated communication between two homes. In doing so, we gained a significant amount of experience in working with existing domestic infrastructures, such

as wireless networks and the AC power grid. We also learnt more about the challenge of integrating component-orientation into our existing design and development methodologies. In light of this experience, we chose to make a number of modifications to ECT that streamlined its use in our design practice, and these modifications are described later in this section. However we first describe the experiences that informed these modifications, which were orientated around an existing ECT component called *X10 Controller*, which provided us with the ability to control domestic lighting installations.

Our first experiments with this component involved a simple system that allowed a user to use a computer to control a desk lamp in their home through X10. We extended this installation with a facility to control the lamp through a hacked wireless keyboard, and then moved on to an installation that allowed a user to control a lamp in another home, for which we made use of networking facilities that were already built into ECT, alongside a pair of standard domestic broadband connections. Finally, we spent some time experimenting with techniques to turn these technologies into a communication channel, and settled on an installation that created the illusion of a pair of lamps, one in each home, that appeared to be sharing a standard AC power source. Figure 6 below shows the equipment that was used in this deployment, whilst figure 7 shows the installation in the context in which it was installed. Each lamp was provided with a dimmer switch (the unit with two coloured buttons), which was monitored by a component, and which could be used to request more or less virtual power from the shared source. This meant that, if a user in one home requested more power for their lamp, then the lamp in the other home would automatically dim. This installation was deployed into a number of homes, and was used by the residences of these homes for some time.

Figure 6 Equipment

Figure 7 Installation in home

In putting together this installation, we confirmed the utility of ECT in putting together home and network-

spanning applications. However, we also substantially modified our ideas of how to adopt component-orientated development into our design process. Initially, we had decided to adopt a model in which the designers in our collaboration experimented with components, and passed on any requirements for change to the developers, who would make these changes, and then distribute new versions of components to the designers. This method was chosen due to the nature of components in ECT; modifying existing components, or adding new components, requires the ability to develop code in a compiled programming language, a task which requires a substantial amount of technical computing knowledge (and one with which our designers were not familiar). However, in practice, this approach raised a number of issues that caused us difficulties. Firstly, such a dependency upon the developers to make any changes to components slowed down our iterative design process. Secondly, the designers in our collaboration often struggled to learn how to use any components that were provided by the developers, since they had no knowledge of their internal workings. As such, we decided to make a number of adaptations to ECT, which have proved successful throughout the rest of our collaboration.

Our primary adaptation was to integrate the Processing [6] scripting language, in order to allow our interaction designers to develop components themselves. Processing is widely used for prototyping within the interaction design community, and it is one with which our designers were familiar. We have termed components which have been developed through Processing as *lightweight components*, and this terminology is used throughout the rest of this paper.

At the same time, however, we continued to make use of more traditional compiled components where necessary, as these still provided a number of benefits, including increased flexibility and performance. In the rest of the paper, these are referred to as *heavyweight components*.

In terms of the power-sharing installation, the two main uses of lightweight components were in developing an interface to the hacked wireless keyboard, and in expressing and prototyping the control logic at the heart of our installation. We also developed lightweight components that allowed the monitoring of the system by third-parties; this facility proved useful as we moved towards trialling our design in real homes. We were aided in the development of these components by the extensive library of scripts that have already been written by those working with Processing. In addition, the development of new features was simplified by the familiarity of the designers in our collaboration with development provided by Processing (see figure 8 for a screenshot of this).

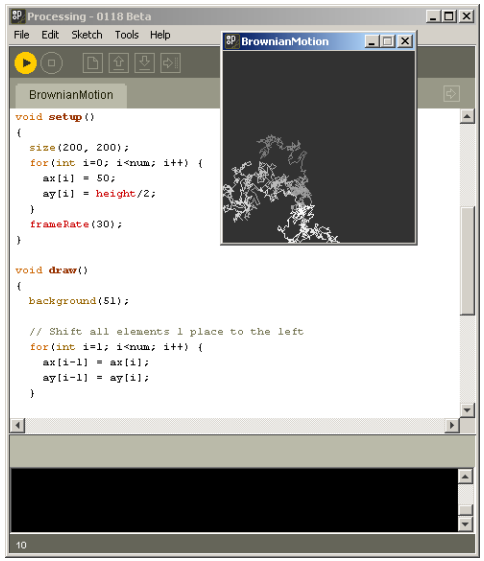

Figure 8 Processing IDE

Episode two – prototyping of components to control small displays

After experimenting with the ambient communication channel described above, the designers in our collaboration became interested in using small displays as a means of communication with domestic residents. However, at the time, there were no components distributed within ECT which were capable of communicating with any small display technologies. As a team, therefore, we decided to develop a new component for this task, and this involved investigations into a variety of different technologies. This episode therefore illustrates a period of collaborative technology design between the members of the team, from which we have learned a number of lessons.

The first technology chosen for investigation as part of this process were LCDs provided as part of the Particle family of technologies. Particle LCDs are text-only, with each being capable of displaying three lines of twelve characters. They are controlled by a Particle base-board, a wireless device which runs an embedded operating system (OS). Communication with this device is by radio transmission, using a packet-based, proprietary protocol. As such, integrating Particle LCDs directly into systems takes substantial expertise in software development in general, and in communications software in particular. Due to the technical difficulty of working with Particle LCDs, this was not a task that the designers could undertake on their own. Development of a component to control Particle LCDs therefore took place in a more traditional manner, with designers providing feedback on early versions of components to developers, in order to influence the design of later versions of components. Since team members were working at different sites, and in order to allow testing of newly-developed components, identical sets of Particle technology were purchased by each site, and each site maintained an installation of ECT into which new components could be installed for testing.

As a first attempt at providing access to these LCDs, the developers produced a heavyweight component, *Particle LCD* which could place an item of text at a particular position on the screen, and distributed a version of this component to the designers, via email. Experimentation with this component, through graphical interfaces provided with ECT, revealed the need to fit more text onto the screen. The designers adopted scrolling as one possibility, and prototyped scrolling text by writing a simple lightweight component

that could be used to control *Particle LCD,* and to move items of text around on its screen. This was then distributed to designers, who experimented with it, and discovered a limitation in the Particle OS, which they corrected by modifying the OS itself. A new version of the OS, along with a new version of the *Particle LCD* component which had been modified to support scrolling, was then distributed by email to the designers, who tested it, and fed back more detailed requirements as to how scrolling should work. These requirements were expressed through a variety of media, including text and video. Requirements were then integrated into *Particle LCD* by the designers, which continued in development until it was suitable for use.

At this point, and despite a working heavyweight component, the Particle platform was abandoned, in favour of a more flexible solution built around Bluetooth communication to repackaged mobile phones (see figure 9 below for a photograph). However, earlier work on developing scrolling routines was useful in developing components to control displays on phones, and code from the *Particle LCD* component was ported across to a new component capable of controlling a phone.

Figure 9 Mobile-phone based displays

In terms of integrating component-orientation into design practice, this episode has raised a number of interesting issues, and has resulted in a number of suggestions for changes to the design of ECT (although, in this case, these designs have not yet been implemented). Importantly, this episode has illustrated that, when components are written with the intention of controlling physical, computational devices, then these components cannot be considered to be self-contained in the traditional sense of the word (as introduced earlier in this paper). In the case of the Particle LCD development process outlined above, the *Particle LCD* component became dependent upon particular modifications to the Particle OS for its successful operation, and even without these modifications, this component would still have been dependent upon particular versions of the Particle hardware platform. Versioning is a major issue here, and at times it has caused us difficulties; we have lost considerable amounts of time in making mistakes such as using a newer version of *Particle LCD* to try to control a Particle with an older version of the operation system. This suggests that, if component-orientation as a technique and technology is to become more useful in the development of network-spanning applications that make use of computational devices, that versioning support needs to become an integrated, automated part of the component-development process (unlike our development process, in which we had to manually track version changes). Potentially, such support might make use of versioning information being embedded into the component itself. It might have been possible, for example, to distribute the *Particle LCD* component with an embedded set of metadata to describe the version of the hardware platform it required, and an embedded version of the Particle OS, for installation

before use of the component (rather than the OS being distributed separately, and installed by the designer). Such changes might improve the efficiency of the design process.

Episode three – assembling a complete system

Episode one demonstrated our ability to construct applications that spanned a network, whilst episode two provided us with a facility to distribute information to small LCDs. Building on these experiences, we began work on the design of a completed system that we intended to deploy into a number of homes. Our choice was to make use of existing components where possible, to write additional components where necessary, and to assemble these components into a working system through use of the ECT graph editor. We then aimed to deploy this system for a lengthy period of time, and to study the impact that its deployment had on domestic life.

Our aim in building this system was, once again, to provide a novel communication channel. This time, however, we decided to focus on providing information about the immediate environment that could be found outside of the home. Having identified a number of websites as being a source of local information, we decided to construct a system that filtered this information by measuring the wind direction on the roof. By measuring this direction, by identifying geographical areas that were upwind from the house, and by gathering information from websites that related to these areas, we hoped to provide residents with interesting knowledge about their local area that they may not have come across otherwise. By distributing a network of LCDs around the home, and by using these to display this information, we hoped to provide an interesting talking point that would be a valuable addition to daily life.

Of course, even once we had developed our ideas to this stage, we were still a long way from a finished system. We needed to refine the design of our system, and we began this process by developing a number of prototypes. These made use of a set of existing heavyweight components with which we had become familiar. Where required, these were augmented with lightweight components which had been constructed from Processing scripts. As in previous episodes, we developed this system as team, and made use of a number of "component-orientated diagrams" to aid us in this process. Figure 10 below shows one of these diagrams, which was produced midway during the process of our prototype development. Each of the ten blocks in this diagram represents one component, with links between these blocks representing information flow between components. Some of these components already existed, whilst others did not. Producing such diagrams allowed us to map out the current state of our system, and our future implementation tasks.

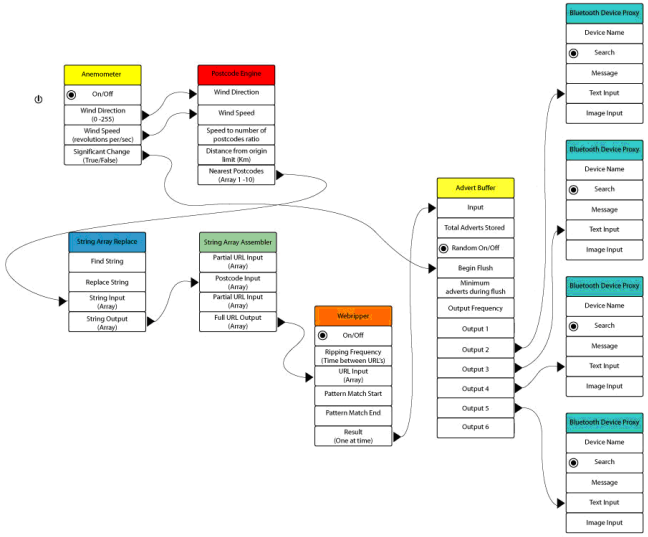

Figure 10

One example of such a task is represented by the component in the top left of figure 10. This particular item was intended to represent a piece of software that was capable of extracting information from an anemometer. Having identified the need for this component, we decided to simulate it by writing a lightweight component which read from a static file of wind simulation data. Later, this component was replaced by a heavyweight component, written by the developers in our collaboration, which was capable of connecting to a real anemometer. A number of other components were also prototyped and improved in this manner. Components intended to filter local information, for example, were prototyped using lightweight components. The performance of these was found to be insufficient, and they were replaced with more efficient heavyweight components which had been crafted by the developers. However, in all of these cases, prototyping work carried out through scripting simplified the later development work, and facilitated the collaboration between members of our team.

Eventually, after a number of iterations, we settled upon a final design for our system, and began a process of testing which we hoped would produce a system which would be sufficiently robust to deploy. However, at this stage, we began to encounter a number of difficult problems with ECT, which delayed our progress. In particular, we struggled to keep any ECT-based system running for more than a few hours; repeatedly, they would all freeze after an unpredictable period, for no apparent reason. After consultation with the authors of the system, we traced our difficulties down to a number of bugs with the implementation of ECT, but fixing these took a lot of effort (and was never satisfactorily completed). Consequently, we decided to implement a stand-alone version of our system, which was not component-orientated, but which implemented all of the behaviour that we had been prototyping throughout our design process. This stand-alone version used much of the code from our existing components, and, as a result, only took a few days to develop. It worked reliably, and was deployed over a substantial time period into a number of homes. There is not the space in this paper to describe these deployments, but they will feature in future publications.

Discussion

This case-study has presented a number of interesting episodes from our collaboration, which has centered on the use of ECT in the prototyping and deployment of

network-spanning applications. We have used ECT throughout a two-year time period, and have developed a substantial amount of experience throughout this time. We have found that it has facilitated our collaboration in many ways, often allowing us to work in more innovative and effective ways than would have been possible otherwise. In addition, however, we have encountered limitations and difficulties with this software, which we have sometimes been able to correct, and which have sometimes caused us difficulties. In this section of the case-study, we reflect on our experiences, drawing out themes, and identifying important lessons. We have grouped these into the following three sub-sections, and hope that they will be useful to others.

ECT as a co-ordination point for collaboration

Our case study has illustrated the capability of ECT as a co-ordination point for our design processes. As a large group of individuals who are distributed across multiple institutions, coordinating our progress has been important, and ECT has supported such coordination in a number of ways. Firstly, it has simplified the processes by which we have collectively learnt about the potential of technologies. This is illustrated in episode one, in which designers made use of existing facilities to distribute components across networks. By deploying such systems into real homes, we gained a substantial amount of knowledge about the capabilities of common wireless networking technologies. Episode two provides another example of this, in which an extended, iterative design process resulted in a finished version of a component that could control a small LCD. Secondly, ECT has simplified the process of articulating our understandings and needs in relation to software and hardware; since ECT allows for the graphical manipulation of components, this makes it easier for designers to point to a component and say "I need it do this" or "I don't understand why it does that". Finally, by supporting the expression of a system as a set of interconnected components, ECT has simplified the process of refining the design of these systems. At various times, team members have collectively looked at a graphical representation of a system, identified tasks that need to be done, and allocated responsibility for these tasks. Such a process is illustrated in episode three, in which components intended to gather wind parameters and use them to filter information were first prototyped as scripts, and then implemented as heavyweight components. Such processes are also illustrated in episode one, in which designers took on the responsibility of prototyping the core logic of the power-sharing installation, whilst developers focused on perfecting the existing *X10 controller* component. Of course, as described in episode 3, we chose not to deploy component-orientated systems into homes, preferring to focus on the development of more efficient and traditionally structured software. This was produced by the developers in our team, in a compiled language, and as such would not be tractable to the designers in our team. However, since so much ECT-orientated prototyping had already taken place, the specification and construction of this software was only a minor task, and there was no need for the modification of this at a later date.

Extending the notion of components

We began our collaboration by adopting the traditional definition of a component as a self-contained unit of software, whose inner workings were hidden, and which would be developed by an expert computer user, for the use of a less-expert computer user. Rapidly, we had

to abandon this notion, in favor of a more flexible definition which suited out purposes better. Firstly, such a definition slowed our design processes, due to the creation of a dependency between developers and designers. Secondly, learning to use components whose inner workings are hidden can be hard, especially if the design of these components is rapidly changing. And finally, as illustrated in episode two, when components are being developed to control individual devices, then such components are effectively dependent upon specific versions of devices and driver software, so are no longer self-contained. Our response to these first two points has been to extend the ways in which components can be defined in ECT, by allowing the use of the Processing scripting language for their development. Effectively, this has allowed the creation of components whose inner workings are visible (since scripts can be dynamically inspected and modified whilst the component is still running). We have not, however, solved the issues of versioning that arise from components no longer being self-contained; we hope that future tools will provide support for such issues.

Augmenting existing design practice

Our description of ECT in this case study has placed it at the centre of the design practice. However, in reality, ECT had to coexist with a variety of other design tools, some of which already had a well-established role in design. On the side of the developers, these tools included programming languages, compilers, and software development environments. On the side of the interaction designers, these included 3D printers and Macromedia Flash. Integrating ECT such an existing context took effort, and potentially involved a steep learning curve, which sometimes delayed our short-term progress. Our key observation here, therefore, is that, although new tools will always require new learning, tool designers are more likely to be successful if they attempt to augment existing practice, rather than replace it. In the case of ECT, limiting ourselves to only heavyweight component development would have represented an attempt to replace existing design practice, and would always be likely to fail. By integrating the Processing scripting language, however, with which our designers were already familiar, we found a way of making use of both their existing experience and an existing library of Processing scripts. Interestingly, although we have presented ECT as being at the core of our collaboration, the designers in particular have tended to see Processing as being at core, with ECT taking on the role of an add on that provided access to a range of hardware devices. This is an interesting example of a conclusion that can only be drawn from studying tools in use, and we would argue that much more work needs to be done to understand other tools in this context.

Conclusion

In this paper we have presented a case-study outlining the use of ECT, a component-orientated design tool, in a two-year collaboration between interaction designers and technologists. We have studied the use of this tool throughout the course of this collaboration, and have presented the results in this paper. Studying the long-term use of this tool has allowed us to develop our understanding of the role of such a tool in an extended, multi-site collaboration, and to provide information which we hope will lead to the development of better tool-support for network-spanning applications in the future.

Acknowledgements
Research described in this paper has been supported through the EPSRC *Equator* project (GR/N15986/01), whilst writing of the paper has been supported through the both the EPSRC *Challenge of Widespread Ubiquitous Computing project* (EP/F03038X/1) and the EPSRC *Wearable Biosensing and the Design, Documentation and Adaptation of Entertainment Experiences* project (EP/F066910/1)

References

[1] Greenhalgh, C., Izadi, S., Mathrick, J., Humble, J. and Taylor, I. A Toolkit to Support Rapid Construction of Ubicomp Environments. Proceedings of UbiSys workshop at UbiComp 2004.

[2] Naur, P. and Randell, B. Software Engineering, Report on a conference sponsored by the NATO Science Committee , Garmisch, Germany, 7th to 11th October 1968. Available at: http://www.cs.dartmouth.edu/~doug/components.txt

[3] Preece, J., Rogers, Y. and Sharp, H. Interaction design: beyond human-computer interaction. Published by Wiley, 2007 (2nd edition)

[4] http://news.bbc.co.uk/1/hi/sci/tech/1264205.stm

[5] http://www.phidget.com

[6] http://www.processing.org