Balancing Autonomy and User Control in Context-Aware Systems - a Survey

Bob Hardian, Jadwiga Indulska and Karen Henricksen
School of Information Technology and Electrical Engineering, The University of Queensland
{bhardian, jaga, karen}@itee.uq.edu.au

Abstract

Application autonomy can reduce interactions with users, ease the use of the system, and decrease user distraction. On the other hand, users may feel loss of control over their applications. A further problem is that autonomous applications may not always behave in the way desired by the user. To mitigate these problems, autonomous context-aware systems must provide mechanisms to strike a suitable balance between user control and software autonomy. In this paper, we present a survey of research on balancing user control and system autonomy in context-aware systems. We address various issues that are related to the control-autonomy trade-off, including issues in context modelling, programming models and tools, and user interface design.

1. Introduction

The proliferation of mobile and embedded computing devices requires a change in the nature of interactions between users and computers. One of the goals of pervasive computing is to reduce user interactions with computing applications: i.e., to make applications more autonomous and proactive. To become autonomous and proactive, pervasive computing applications need to place greater dependence on context information in order to dynamically adapt their behaviour to suit the environment and user requirements.

Application autonomy can reduce interactions with users, ease the use of the system, and decrease user distraction. On the other hand, users may feel loss of control over their applications. A further problem is introduced by trade-offs between prescription and freedom, which may result in autonomous applications not behaving in the way desired or expected by the user [1]. To mitigate these problems, autonomous context-aware systems must provide mechanisms to strike a suitable balance between user control and software autonomy. This involves providing mechanisms to make users aware of reasons for application adaptations by selectively revealing aspects of the application state, such as context information, user preference information and adaptation logic used in decision making processes. By providing this information, users may be able to correct undesirable actions (for example, by changing context information or preferences appropriately). However, users can have varying levels of expertise, and this affects their understanding of system operations. Therefore, the challenge is not only to identify what application state information should be exposed, but also in what manner (e.g., with what level of explanation).

In this paper we describe requirements for achieving balance of control between users and context-aware applications, and evaluate existing solutions for addressing these requirements. There are few existing solutions because the challenge of designing applications to provide appropriate control to users has traditionally taken a back seat to more fundamental problems in context-aware systems, like sensing and interpreting context. At the end of the paper, we also provide a brief road-map for future work in this area.

2. Balance of control

The design space for providing user control in context-aware applications can be characterised in terms of the continuum shown in Figure 1.

At the leftmost end (A), users are given full control over application behaviour, and applications have very little autonomy. Applications designed in this way are the most interactive. Conversely, at the other end (C),

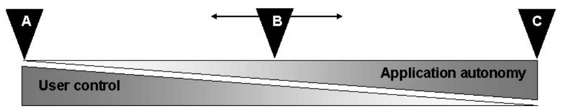

Figure 1. A continuum of user control versus application autonomy

applications only require a small amount of user control, while pro-activity and autonomy play important roles in reducing interactions with users.

Applications can also occupy any intermediate position on the continuum (position B). However, it cannot be assumed that one position is always better than another for a given application. The appropriate position along the continuum will be dictated by the user's needs, situation and expertise.

In traditional applications, the trade-off between user control and software autonomy has been fixed at design-time. In contrast, context-aware applications may need to adjust the balance of user control and software autonomy at run-time. However, models for designing applications to support this type of adaptation do not yet exist. In the remainder of this paper, we survey the work in the field of context-awareness that is most closely related to this problem, with the goal of demonstrating the current state-of-the-art and highlighting key issues for future work.

3. Studies on user control

User control can be conceptualised as the level of user intervention that is required to operate a system [2]. By placing a greater dependence on contextual information, a context-aware system can operate with limited user intervention. However, van der Heijden [2] argues that the transfer of control from the user to the system results in an increase in user anxiety (that is, the personal discomfort that a user associates with the use of the system), so that the lower the level of user control, the more anxiety people exhibit after using the system. However, one of the objectives of pervasive computing is to decrease *mental effort* in the interaction with the system. Van der Heijden also argues that, the lower the level of control, the easier to use people perceive the system to be. This is because highly autonomous systems require less input and thought on the part of the user.

In general, the user's sense of control decreases when the autonomy of the application increases; however, as discussed by Barkuus and Dey [3], users are willing to accept a large degree of autonomy from applications as long as the application's usefulness is greater than the cost of limited control.

Designing user interactions with computers involves deciding how to divide functions between humans and computers [4]. Not all tasks can (or should) be delegated to the system. There are *residual tasks* that are left to human users when a system is automated. In order to ensure that the residual tasks can be carried out effectively by users, the system should reveal its current understanding of the automated function, and allow users to correct this understanding whenever the system produces undesirable outcomes.

In addition to applications exposing their understandings of context information to users, context-aware applications need to ensure that actions they take on behalf of users are both intelligible and accountable [5, 6]. That is, the systems cannot simply be trusted to take action on behalf of users. Bellotti and Edwards suggest that users should be involved in system actions as follows (reproduced from [5]):

- If there is only slight doubt about what the desired outcome might be, the user must be offered an effective means to *correct the system action;*
- If there is significant doubt about the desired outcome, the user must be able to *confirm the action* the system intends to take; and
- If there is no real basis for inferring the desired outcome, the user must be offered available *choices for system action*.

As discussed by Bellotti and Edwards, accountability of a context-aware system can be achieved by informing the user of the system's capabilities and its understanding of the current context, disclosing actions taken by the system, providing feedback to the user, and providing mechanisms for user control. However, Bellotti and Edwards only suggest general design principles for context-aware systems; their work does not extend to recommending design approaches or methodologies that can be used to put these principles into practice.

4. Context modelling and reasoning

Providing a sufficient understanding of a context-aware system's view of the context and its corresponding actions as discussed in the previous section requires selectively exposing some of the internal application state to users. As discussed in Section 1, this state information includes the available context information (as well as the means by which it was derived), the system's current knowledge of the user's preferences, and additional knowledge and logic used to arrive at adaptation decisions. By exposing these types of information, users are better positioned to understand and correct inappropriate actions than when context-aware systems are developed in the form of "black boxes". This, in turn, increases user control, as users may be able to trace inappropriate behaviours back to incorrect context or preference information (or even to failed or mis-configured sensors), and to correct this information accordingly.

To date, the context-awareness community has placed much more emphasis on modelling context information than on modelling preferences and adaptation logic. (Preferences and adaptation logic are typically handled directly within the application source code, in a manner that makes the logic relatively difficult to change or expose to users.) The context modelling techniques developed so far provide a natural starting point for research on generic (i.e., application-independent) models for exposing context information, and the means by which it was derived, to users.

Context models vary from very simple models to advanced models. Advanced context models can capture not only basic facts but also high-level situations derived from facts. They can also model relationships between context facts and quality of context information. Many efforts have been made to develop common context models and representations, as well as reasoning algorithms that facilitate context sharing and interoperability among context-aware applications. A recent survey by Strang and Linnhoff-Popien [7] classified context modelling approaches into several categories: key-value models, mark-up scheme models, graphical models, object-oriented models, logic-based models, and ontology based models. For a broad discussion of approaches, we refer the reader to that paper. Here, we describe a subset of the approaches for illustrative purposes. At the end of our discussion on context modelling, we use these selected models to suggest possible approaches for revealing context information to users. Our motivation for choosing the particular context modelling approaches that are covered in this section is to highlight the wide variations in current context modelling approaches, and to illustrate how different models naturally lend themselves to different means of exposing context information.

One context modelling approach, which we developed in our work on software engineering techniques for pervasive computing [8], is the Context Modelling Language (CML). CML assists designers with the tasks of exploring and specifying the context requirements of a context-aware application. CML provides a graphical notation for describing types of information. The model provides two levels of abstraction: facts, which capture fine-grained information, and situations, which describe abstract classes of context described in terms of facts. The model also distinguishes various types of context information, based on persistence and source (i.e., sensed, static, profiled and derived context), and allows quality of context information to be modelled. In addition, we have defined additional models for describing user preferences and adaptation logic, both of which are based on our context modelling approach. These models will be described further in Section 5.2.

Thomson et al. [9] show another approach to model situations in context-aware systems using information retrieval (IR) techniques. This approach builds on the fact that there are similarities between the tasks of text classification and situation determination. A vector space model can be used to describe situations in terms of a multi-dimensional Euclidean space, where each axis corresponds to a term. A situation snapshot is treated as a document of terms. The vector space model can be applied to the representation of contextual information to position a situation snapshot in situation space. Additionally, the situation of a snapshot can be identified using text classifiers, such as Support Vector Machine (SVM) techniques.

Ranganathan et al. [10] propose a model that is based on first order predicate calculus. First order logic can be used to perform inductive and deductive logic reasoning on contexts. This approach allows the deduction of higher-level contexts from low-level sensed contexts using rule-based techniques. Ranganathan et al.'s modelling approach also makes use of an ontology to specify both basic context types (in terms of classes) and the structure of context predicates for use in checking predicate validity.

These modelling techniques are designed primarily for internal use by context-aware applications and supporting middleware components, as well as use by application designers and developers. Very little work has been done so far on developing conceptual models or user interfaces for explaining the information expressed using these modelling approaches to users. Naturally, the different modelling approaches lend themselves to different styles of explanation. For example, Thomson et al.'s approach, which models situations in terms of vectors, may lend itself to a graphical style of explanation, in which relationships between situations are depicted diagrammatically in terms of overlaps, proximity, and so on. In contrast, ontology-based approaches may be able to leverage easily understandable ontology concepts (e.g., "sameAs", "differentTo", and "AllDifferent" in the case of OWL [11]) to show relationships and explain terms. In our modelling approach, it is possible to show users run-time traces of situation, preference and adaptation rule evaluations. These can help to explain the use of context information by the application, and can be used to identify only the relevant context information to expose to users (i.e., the subset actually used in a given decision made by a context-aware application, as opposed to the entire set of context information available to the application). We discuss this idea further in Section 5.2.

In future work, it will be necessary not only to explore techniques for exposing context information to users for each modelling approach, but also to compare the approaches to determine which are the most appropriate (in general, as well as in specific application domains) for building the "intelligible and accountable" systems advocated by Bellotti and Edwards.

Another related problem is the development of appropriate techniques for presenting different *classes* of context information to users. The problem here is to take each class of information (for example, activity information, location information), and to identify the most appropriate *modes* of presentation. For some classes, a graphical presentation may be the most natural, while for others, audio or textual presentation will be more appropriate. Early work on visualisation of location information, using graphical, map-based approaches, has already been carried out by Aaltonen and Lehikoinen [12] and Li et al. [13] (as well as others), but more work is still needed in this area.

The issue of presenting context information is complicated by the fact that separate presentation of different classes of information may not always be helpful. When a context-aware application combines several classes of information in complex ways to make decisions, it may be the combined result, rather than the individual pieces of context information, that is relevant to the user. In these cases, better results may be achieved by showing direct links between the application behaviours and the combined context inputs, rather than showing pieces of context information in isolation.

5. Programming models and tools

Developing context-aware applications that provide appropriate levels of feedback and control to users requires innovative programming models and tools. In this section, we briefly review the following relevant solutions: end-user programming techniques, a preference-based decision support mechanism, and an extension to Dey et al.'s Context Toolkit [14]. Other programming models and tools have been proposed for context-aware systems, but they are not covered, as their emphasis is not on supporting user control.

5.1. End-user programming

End-user programming allows users, rather than application developers, to define the behaviour of context-aware systems. It aims to ensure a closer match between user requirements and system behaviours than is often achieved with traditional software engineering approaches, and allows users to add functionality that could not have been anticipated by system designers.

One example of an end-user programming system is *a CAPpella* [15], which supports a paradigm called *programming by demonstration*. *a CAPpella* requires the user to demonstrate desired system behaviours by carrying out actions manually. It relies on machine learning techniques to build *recognizers* to detect the situations in which the actions should occur. Once the system has been trained, it is able to carry out actions automatically without prompting from the user.

The Topiary tool [13] is a second solution that incorporates a simpler form of programming by demonstration for recording scenarios for prototyping purposes. However, it is not truly an end user programming solution – instead, it aims to support application designers by allowing them to create map-based location models, develop scenarios by moving objects around a location map, and create and run storyboards. Topiary is focused exclusively on location-based applications.

The Jigsaw editor [16] differs from *a CAPpella* and Topiary in that it supports end-user programming by *assembly*, rather than programming by demonstration. It provides a visual interface based on a jigsaw puzzle metaphor, which enables end-users to connect together components, such as sensors, displays and applications, to carry out tasks. Jigsaw primarily targets domestic environments – for example, by enabling users to construct doorbells or simple surveillance systems from input and output devices in the home.

Although end-user programming techniques generally provide better user control than traditional software engineering techniques, they are not appropriate for all application domains. They are most appropriate when the required system behaviours are reasonably straightforward. The simple tasks commonly described in the literature – for example, controlling lights or creating doorbells – are indicative of the level of complexity that can currently be achieved by end-user programming systems. A further problem is that most of the literature on programming by demonstration deals only with how to train the system initially, not with how to later override or modify behaviour (for example, when unexpected actions arise or user requirements change).

5.2. Preference-based decision support

A more traditional approach to provide users with control over their software is to incorporate preference or personalisation mechanisms. This approach is well known, but has not been widely studied specifically in connection with context-aware applications. Personalisation of context-aware applications is more complex than personalisation of traditional applications, because of the potential for dependencies between the context and user preferences (specifically, user preferences may be predicated on the context).

Although there are at least several context-aware applications that allow configuration of user preferences, there are few general programming tools or models that support adaptation of application behaviour based on a combination of context and user preference information. We have developed one solution that addresses this problem [17]. This solution has three parts: (i) a generic preference model used to specify context-dependent user requirements, (ii) a programming model (the *branching* model) in which preferences are combined with context information to support decision making by applications about which action(s) to invoke on behalf the user, and (iii) a supporting programming toolkit. We have developed a variety of context-aware applications using this approach [8], and found that it improves user control, but does not always improve transparency and predictability, particularly when the preferences are complex. Therefore, we are currently developing extensions that provide greater feedback to users about how their preference information is used at run-time. Users will be able to view the particular preferences and context information that led to past application actions, and thereby "debug" and correct recurring problems as discussed in Section 4.

5.3. Context Toolkit extension

Finally, Newberger and Dey [18] propose a very specific solution for user control, based on Dey et al.'s Context Toolkit [14]. This is founded on the idea that, if application state information is exposed in an accessible way, it can be leveraged for building separate user interface components that allow monitoring and control. The solution introduces an *enactor* component into the Context Toolkit. The idea behind this component is to encapsulate application state information and adaptation logic, and to facilitate external access through the provision of a standard API. This allows user interface components, such as Macromedia Flash components, to easily communicate with enactors. These user interface components can support monitoring and control of the enactor, thereby facilitating fine-tuning of the enactor behaviour by the designer. Enactors have the following standard sub-components: *references* (for acquiring context data from widgets), *listeners* (for monitoring changes), and *parameters* (for controlling the behaviour).

Although this solution is primarily intended for use by application designers, it is easy to see that a similar solution could be developed to allow users to monitor and control context-aware behaviour.

6. Conclusions and future work

Systems that sense and use context should selectively reveal relevant context information to users, who can then judge its accuracy. In order to be understood and controlled, context-aware applications need to reveal the elements that influence their behaviours. As well as revealing context information, this can entail exposing other aspects of the internal application state that come into play in decision making processes.

As we discussed in this paper, techniques for providing appropriate explanations and control mechanisms to users, to offset common problems introduced by highly autonomous behaviour, are either primitive or non-existent in current context-aware systems. One of the most advanced approaches to providing user control is offered by end-user programming techniques; however, as discussed in Section 5.1, end-user programming is not appropriate for all application domains. Newberger and Dey's extension to the Context Toolkit is also promising, but has only been explored so far as a tool for the designer to fine-tune application behaviour, not for end-user control. Finally, our preference-based decision support mechanism can potentially be opened up to allow users to inspect traces of past preference and context evaluations, but this remains part of our future work. The use of traces will allow users to easily see where decision making is going wrong. Control/feedback mechanisms can be provided alongside the explanations, so that users are not only passive observers of the system behaviour, but are able to directly manipulate it (for example, by adjusting preferences).

Another important topic for future work involves developing techniques for visualising context. Early work has addressed visualisation of location information, but further work is required, both in relation to location information and other types of context information. As discussed in Section 4, different context modelling approaches lend themselves to different styles of explanation and visualisation. Therefore, there is a need to investigate model-specific techniques, as well as to develop an understanding of which modelling approaches are the most amenable to explanation and visualisation, both in general as well as in particular application domains.

Appropriate techniques for revealing context information and decision processes will lead to new programming models and toolkits for developing context-aware applications. In most application domains, these should eventually replace the current programming models and toolkits which generally lead to inflexible applications developed in a “black box” style.

Finally, as we discussed in Section 2, users have different requirements and capabilities when it comes to interacting with context-aware systems. This means that context-aware systems may need to vary the extent and nature of feedback, explanations and control to users at run-time. To support this, models of user expertise are needed that are suitable for use in context-aware systems. In addition, it will be necessary to devise means of incorporating the notion of expertise into visualisation and programming models. A related issue will be how to support the notion of “interruptability”, so that the level of interaction with users can also be adjusted according to the user’s current task.

7. References

[1] K. Cheverst, N. Davies, K. Mitchell, and C. Efstratiou, "Using Context as a Crystal Ball: Rewards and Pitfalls," *Personal Ubiquitous Comput.*, vol. 5, no. 1, pp. 8-11, 2001.

[2] H. van der Heijden, "Ubiquitous computing, user control, and user performance: conceptual model and preliminary experimental design," in *Proceedings of the Research Symposium on Emerging Electronic Markets*. Bremen, 2003.

[3] L. Barkhuus and A. Dey, "Is context-aware computing taking control away from the user? Three levels of interactivity examined," in *Ubicomp 2003: Ubiquitous Computing*, vol. 2864, *Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2003, pp. 149-156.

[4] M. D. Harrison, R. E. Fields, and P. C. Wright, "Supporting Concepts of Operator Control in the Design of Functionally Distributed Systems," in *Proceedings of ALLFN'97*. Galway, 1997.

[5] V. Bellotti and K. Edwards, "Intelligibility and accountability: Human considerations in context-aware systems," *Human-Computer Interaction*, vol. 16, no. 2-4, pp. 193-212, 2001.

[6] S. Anderson, M. Hartswood, R. Procter, M. Rouncefield, R. Slack, J. Soutter, and A. Voss, "Making Autonomic Computing Systems Accountable: The Problem of Human-Computer Interaction," in *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*: IEEE Computer Society, 2003, pp. 718.

[7] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey," in *The Sixth International Conference on Ubiquitous Computing (UbiComp2004). Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management*. Nottingham, England, 2004.

[8] K. Henricksen and J. Indulska, "Developing context-aware pervasive computing applications: Models and approach," *Pervasive and Mobile Computing*, In Press, Elsevier, 2005.

[9] G. Thomson, P. Nixon, and S. Terzis, "Towards Adhoc Situation Determination," in *The Sixth International Conference on Ubiquitous Computing (UbiComp2004). Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management*. Nottingham, England, 2004.

[10] A. Ranganathan and R. H. Campbell, "An infrastructure for context-awareness based on first order logic," *Personal Ubiquitous Comput.*, vol. 7, no. 6, pp. 353-364, 2003.

[11] D. L. McGuinness and F. v. Harmelen, "OWL Web Ontology Language Overview, W3C Recommendation," 2004.

[12] A. Aaltonen and J. Lehikoinen, "Refining visualization reference model for context information," *Personal Ubiquitous Comput.*, vol. 9, no. 6, pp. 381-394, 2005.

[13] Y. Li, J. I. Hong, and J. A. Landay, "Topiary: a tool for prototyping location-enhanced applications," in *Proceedings of the 17th annual ACM symposium on User interface software and technology*. Santa Fe, NM, USA: ACM Press, 2004, pp. 217-226.

[14] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction*, vol. 16, no. 2-4, pp. 97-166, 2001.

[15] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu, "a CAPpella: programming by demonstration of context-aware applications," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. Vienna, 2004, pp. 33-40.

[16] J. Humble, A. Crabtree, T. Hemmings, K.-P. Åkesson, B. Koleva, T. Rodden, and P. Hansson, "Playing with the Bits - User-Configuration of Ubiquitous Domestic Environments," vol. 2864, *Lecture Notes in Computer Science*, 2003, pp. 256-263.

[17] K. Henricksen and J. Indulska, "Personalising Context-Aware Applications," in *OTM Workshop on Context-Aware Mobile Systems*, vol. 3762, *Lecture Notes in Computer Science*: Springer-Verlag, 2005, pp. 122-131.

[18] A. Newberger and A. Dey, "Designer Support for Context Monitoring and Control," Intel Research Berkeley Technical Report IRB-TR-03-017, 2003.